Journal of Automated Reasoning (2021) 65:1071–1096
https://doi.org/10.1007/s10817-021-09600-4

Optimization Modulo the Theories of Signed Bit-Vectors and Floating-Point Numbers

Patrick Trentin[1] · **Roberto Sebastiani**[1]

Received: 15 April 2020 / Accepted: 19 October 2020 / Published online: 23 July 2021

Abstract

Optimization modulo theories (OMT) is an important extension of SMT which allows for finding models that optimize given objective functions, typically consisting in linear-arithmetic or Pseudo-Boolean terms. However, many SMT and OMT applications, in particular from SW and HW verification, require handling *bit-precise* representations of numbers, which in SMT are handled by means of the theory of bit-vectors ($\mathcal{BV}$) for the integers and that of floating-point numbers ($\mathcal{FP}$) for the reals respectively. Whereas an approach for OMT with (unsigned) $\mathcal{BV}$ objectives has been proposed by Nadel & Ryvchin, unfortunately we are not aware of any existing approach for OMT with $\mathcal{FP}$ objectives. In this paper we fill this gap, and we address for the first time OMT with $\mathcal{FP}$ objectives. We present a novel OMT approach, based on the novel concept of *attractor* and *dynamic attractor*, which extends the work of Nadel and Ryvchin to work with signed-$\mathcal{BV}$ objectives and, most importantly, with $\mathcal{FP}$ objectives. We have implemented some novel OMT procedures on top of OptiMathSAT and tested them on modified problems from the SMT-LIB repository. The empirical results support the validity and feasibility of our novel approach.

Keywords Optimization Modulo Theories · OMT Satisfiability Modulo Theories · SMT Floating-Point Arithmetic attractor dynamic attractor

1 Introduction

Optimization modulo theories (OMT) [6–9,20–23,28–32,35,36,38–43] is an important extension to satisfiability modulo theories which allows for finding models that optimize one or more objectives, which typically consist in some linear-arithmetic or Pseudo-Boolean function application.

This is an extension of a paper published at CADE 2019 conference [44]. We would like to thank the anonymous reviewers for their insightful comments and suggestions, and we thank Alberto Griggio for support with MathSAT code. The authors are sponsored in part by D-Wave Systems Inc. via project QuASi.

✉ Roberto Sebastiani
roberto.sebastiani@unitn.it

[1] DISI, University of Trento, Trento, Italy

Nevertheless, many SMT and OMT applications, in particular from SW and HW verification, require handling *bit-precise* representations of numbers, which in SMT are handled by means of the theory of bit-vectors ($\mathcal{BV}$) for the integers and that of Floating-Point Numbers ($\mathcal{FP}$) for the reals respectively and their combination ($\mathcal{BV} \cup \mathcal{FP}$). For instance, during the verification process of a piece of software, one may look for the minimum/maximum value of some `int` or `double` parameter causing an SMT($\mathcal{BV} \cup \mathcal{FP}$) call to return SAT—which typically corresponds to the presence of some bug—so that to guarantee a safe range for such parameter; also, one may want to find the maximum relative difference in the `double` values returned by two implementations of the same function.

Example 1 Consider some C/C++ library implementation of some mathematical function f: $\texttt{Double}^N \longmapsto \texttt{Double}$. Suppose one wants to substitute it with a new implementation $f'(...)$ of the same function. Given the ranges $[\underline{l}, \underline{u}]$ for the values of $\underline{x}$, one may want to find the maximum relative difference between the value returned by the two functions. This can be done, e.g., by finding the maximum value of ϵ s.t. the SMT($\mathcal{BV} \cup \mathcal{FP}$) formula

$$\begin{aligned}&(|f(\underline{x}) - f'(\underline{x})| > \epsilon * max\{|f(\underline{x})|, |f'(\underline{x})|\}) \wedge \\ &(f(\underline{x}) = ...) \wedge (f'(\underline{x}) = ...) \wedge \textstyle\bigwedge_{i=1}^{N}((l_i \leq x_i) \wedge (x_i \leq u_i))\end{aligned} \qquad (1)$$

is satisfiable, where $(f(\underline{x}) = ...)$ and $(f'(\underline{x}) = ...)$ are the SMT($\mathcal{BV} \cup \mathcal{FP}$)[1] encodings of the implementations of the functions f and f' respectively.[2] Notice that here it is strictly necessary to use bit-precise representation of numbers provided by $\mathcal{BV} \cup \mathcal{FP}$ —rather than standard non-linear arithmetic—in order to reproduce the truncating and rounding errors and their propagation. (E.g., two C functions computing iteratively $a_0 + a_1 * x + ... + a_n * x^n$ and $a_0 + x * (a_1 + x * (... + x * (a_n))..))$ by floating-point arithmetic may return different values on the same input value x, although they are mathematically equivalent.)

OMT for the theory of (unsigned) bit-vectors was proposed by Nadel and Ryvchin [32], although a reduction of the problem to MaxSAT was already implemented in the SMT/OMT solver Z3 [10]. The work in [32] was based on the observation that OMT on unsigned $\mathcal{BV}$ can be seen as lexicographic optimization over the bits in the bitwise representation of the objective, ordered from the most-significant bit (MSB) to the least-significant bit (LSB). Notice that, in this domain, this corresponds to a binary search over the space of the values of the objective.

In this paper (as in [44]) we address—for the first time to the best of our knowledge—OMT for objectives in the theory of signed Bit-Vectors and, most importantly, in the theory of Floating-Point Arithmetic, by exploiting some properties of the two's complement encoding for signed $\mathcal{BV}$ and of the IEEE 754-2008 encoding for $\mathcal{FP}$ respectively. (We consider the former as a straightforward extension of [32], and the latter as our main contribution.)

We start from introducing the notion of *attractor*, which represents (the bitwise encoding of) the target value for the objective which the optimization process aims at. This allows us to easily leverage the procedure of [32] to work with both *signed* and *unsigned* bit-vectors, by minimizing lexicographically the bitwise distance between the objective and the attractor, that is, by minimizing lexicographically the bitwise-xor between the objective and the attractor.

Unfortunately there is no such notion of (fixed) attractor for $\mathcal{FP}$ numbers, because the target value changes as long as the bits of the objective are updated from the MSB to the LSB, and the optimization process may have to change dynamically its aim, even in the opposite

[1] Notice that the implementation of f, f' may contain also some integers, so that $\mathcal{BV} \cup \mathcal{FP}$ is needed.

[2] Here we use "$|f(\underline{x}) - f'(\underline{x})| > \epsilon * max\{|f(\underline{x})|, |f'(\underline{x})|\}$" to handle the case $f(\underline{x}) = f'(\underline{x}) = 0$.

direction. (For instance, as soon as the minimization process realizes there is no solution with a negative value for the objective and thus sets its MSB to 0, the target value is switched from $-\infty$ to 0^+, and the search switches direction, from the maximization of the exponent and the significand to their minimization.).

To cope with this fact, we introduce the notions of *dynamic attractor* and *attractor trajectory*, representing the dynamics of the moving target value, which are progressively updated as soon as the bits of the objective are updated from the MSB to the LSB. Based on these ideas, we present novel OMT procedures for $\mathcal{FP}$ objectives, which require at most $n + 2$, incremental calls to an $\mathcal{BV} \cup \mathcal{FP}$ solver, n being the number of bits in the representation of the objective. Notice that these procedures do not depend on the underlying $\mathcal{BV} \cup \mathcal{FP}$ procedure used, provided the latter allows for accessing and setting the single bits of the objective.

Notice that, unlike with the $\mathcal{BV}$ domain, this does not simply perform binary search over the space of the values of the objective. Rather, it first performs (a lexicographic bitwise search corresponding to) binary search of the *exponent* values, which very-rapidly converges to the right order of magnitude, followed by binary search on the *significand* values, which fine-tunes the final result.

We have implemented these OMT procedures on top of the OptiMathSAT OMT solver [43]. We have run an experimental evaluation of the procedures on modified SMT problems from the SMT-LIB library. The empirical results support the validity and feasibility of the novel approach.

The rest of the paper is organized as follows. In Sect. 2 we provide the necessary background on $\mathcal{BV}$ and $\mathcal{FP}$ theories and reasoning. In Sect. 3 we provide the novel theoretical definitions and results. In Sect. 4 we describe our novel OMT procedures. In Sect. 5 we present the empirical evaluation. In Sect. 6 we conclude, hinting some future directions.

2 Background

We assume some basic knowledge on SAT and SMT and briefly introduce the reader to the Bit-Vector and Floating-Point theories.

Bit-Vectors A *bit* is a Boolean variable that can be interpreted as 0 or 1. A Bit-Vector ($\mathcal{BV}$) variable $\mathbf{v}^{[n]}$ is a vector of n bits, where $v[0]$ is the Most Significant Bit (MSB) and $v[n-1]$ is the Least Significant Bit (LSB).[3] A $\mathcal{BV}$ constant of width n is an interpreted vector of n values in $\{0, 1\}$. We $\overline{\texttt{overline}}$ a bit value or a $\mathcal{BV}$ value to denote its complement (e.g., $\overline{[11010010]}$ is $[00101101]$). A $\mathcal{BV}$ variable/constant of width n can be *unsigned*, in which case its domain is $[0, 2^n - 1]$, or *signed*, which we assume to comply with the *Two's complement* representation, so that its domain is $[-2^{(n-1)}, 2^{(n-1)} - 1]$. Therefore, the vector $[11111111]$ can be interpreted either as the unsigned $\mathcal{BV}$ constant $\mathbf{255}^{[8]}$ or as the signed $\mathcal{BV}$ constant $-\mathbf{1}^{[8]}$. Following the SMT- LIBv2 standard [4], we may also represent a $\mathcal{BV}$ constant in *binary* form (e.g. $\mathbf{28}^{[8]}$ is written #b00011100). A $\mathcal{BV}$ term is built from $\mathcal{BV}$ constants, variables and interpreted $\mathcal{BV}$ functions which represents standard Register-Transfer Level (RTL) operators: word concatenation (e.g. $\mathbf{3}^{[8]} \circ \mathbf{x}^{[8]}$), sub-word selection (e.g. $(\mathbf{3}^{[8]}[6:3])^{[4]}$), modulo-n sum and multiplication (e.g. $\mathbf{x}^{[8]} +_8 \mathbf{y}^{[8]}$ and $\mathbf{x}^{[8]} \cdot_8 \mathbf{y}^{[8]}$), bit-wise operators (like, e.g., $\mathbf{and}_n$, $\mathbf{or}_n$, $\mathbf{xor}_n$, $\mathbf{nxor}_n$, $\mathbf{not}_n$), left and right shift $<<_n$, $>>_n$. A $\mathcal{BV}$ atom can be built by combining $\mathcal{BV}$ terms with interpreted predicates (either signed or unsigned ones) like $\geq_n$, $<_n$ (e.g.

[3] Although most often in the literature the indexes $i \in [0, ..., n-1]$ use to grow from the LSB to the MSB, in this paper we use the opposite notation because we always reason from the MSB down to the LSB, so that to much simplify the explanation.

$\mathbf{0}^{[8]} \geq_8 \mathbf{x}^{[8]}$) and equality. We refer the reader to [4,25] for further details on the syntax and semantics of Bit-Vector theory.

There are two main approaches for $\mathcal{BV}$ satisfiability, the "*eager*" and the "*lazy*" approach, which are substantially complementary to one another [26]. In the *eager* approach, $\mathcal{BV}$ terms and constraints are encoded into SAT via bit-blasting [17,18,24,25,33,34]. In the *lazy* approach, $\mathcal{BV}$ terms are not immediately expanded—so to avoid any scalability issue—and the $\mathcal{BV}$ solver is comprised by a layered set of techniques, each of which deals with a subportion of the $\mathcal{BV}$ theory [11,16,19,25].

Floating-Point The theory of *Floating-Point Numbers* ($\mathcal{FP}$), [4,14,37], is based on the IEEE standard 754-2008 [5] for floating-point arithmetic, restricted to the binary case. A $\mathcal{FP}$ sort is an indexed nullary sort identifier of the form (_ `FP` $< ebits > < sbits >$) s.t. both $ebits$ and $sbits$ are positive integers greater than one, $ebits$ defines the number of bits in the exponent and $sbits$ defines the number of bits in the significand, including the hidden bit. A $\mathcal{FP}$ variable $\mathbf{v}^{[n]}$ with sort (_ `FP` $< ebits > < sbits >$) can be indifferently viewed as a vector of $n \overset{\text{def}}{=} ebits + sbits$ bits, where $v[0]$ is the Most Significant Bit (MSB) and $v[n-1]$ is the Least Significant Bit (LSB), or as a triplet of bit-vectors $\langle \mathbf{sign}, \mathbf{exp}, \mathbf{sig} \rangle$ s.t. **sign** is a $\mathcal{BV}$ of size 1, **exp** is a $\mathcal{BV}$ of size $ebits$ and **sig** is a $\mathcal{BV}$ of size $sbits - 1$. A $\mathcal{FP}$ constant is a triplet of $\mathcal{BV}$ constants. Given a fixed floating-point sort, i.e. a pair $\langle ebits, sbits \rangle$, the following $\mathcal{FP}$ constants are implicitly defined:

value	**Symbol**	$\mathcal{BV}$ **Repr.**
plus infinity	(_ `+oo` $< ebits > < sbits >$)	`(fp #b0 #b1...1 #b0...0)`
minus infinity	(_ `-oo` $< ebits > < sbits >$)	`(fp #b1 #b1...1 #b0...0)`
plus zero	(_ `+zero` $< ebits > < sbits >$)	`(fp #b0 #b0...0 #b0...0)`
minus zero	(_ `-zero` $< ebits > < sbits >$)	`(fp #b1 #b0...0 #b0...0)`
not-a-number	(_ `NaN` $< ebits > < sbits >$)	`(fp t #b1...1 s)`

where `t` is either 0 or 1 and `s` is a $\mathcal{BV}$ which contains at least a 1.

Setting aside special $\mathcal{FP}$ constants, the remaining $\mathcal{FP}$ values can be classified to be either normal or subnormal (a.k.a. denormal) [5]. A $\mathcal{FP}$ number is said to be *subnormal* when every bit in its exponent is equal to zero, and *normal* otherwise. The significand of a normal $\mathcal{FP}$ number is always interpreted as if the leading binary digit is equal 1, whereas for denormalized $\mathcal{FP}$ values the leading binary digit is always 0. This allows for the representation of numbers that are closer to zero, although with reduced precision. Notice that the absolute value of any subnormal $\mathcal{FP}$ number is smaller than the absolute value of any non-zero normal $\mathcal{FP}$ number, and that the value contribution of the significand bits is always less significant than that of the exponent bits.

Example 2 Let x be the normal $\mathcal{FP}$ constant `(_ FP #b0 #b1100 #b0101000)`, and y be the subnormal $\mathcal{FP}$ constant `(_ FP #b0 #b0000 #b0101000)`, so that their corresponding sort is (_ `FP` $< 4 >< 8 >$). Then, according to the semantics defined in the IEEE standard 754-2008 [5], the floating-point value of x and y in decimal notation is:

$$x = (-1)^0 \cdot 2^{(12-7)} \cdot \left(1 + \sum_{i=1}^{7} \left(x[4+i] \cdot 2^{-i}\right)\right) = 1 \cdot 2^5 \cdot \left(1 + \frac{1}{2^2} + \frac{1}{2^4}\right) = 42$$

$$y = (-1)^0 \cdot 2^{(0-7+1)} \cdot \left(0 + \sum_{i=1}^{7} \left(y[4+i] \cdot 2^{-i}\right)\right) = 1 \cdot 2^{-6} \cdot \left(\frac{1}{2^2} + \frac{1}{2^4}\right) = \frac{5}{2^{10}}.$$

Notice that with (_ FP $<4><8>$) the smallest strictly-positive normal value is 2^{-6}, whereas the greatest subnormal value is $2^{-6} \cdot \sum_{i=1}^{7} 2^{-i}$, which is smaller than 2^{-6}. $\diamond$

The theory of $\mathcal{FP}$ provides a variety of built-in floating-point operations as defined in the IEEE standard 754-2008. This includes binary arithmetic operations (e.g. $+$, $-$, $\star$, $\div$), basic unary operations (e.g. abs, $-$), binary comparison operations (e.g. $\leq, <, \neq, =, >, \geq$), the remainder operation, the square root operation and more. Importantly, arithmetic operations are performed *as if with infinite precision*, but the result is then *rounded* to the “nearest” representable $\mathcal{FP}$ number according to the specified *rounding mode*. Five *rounding modes* are made available, as in [5].

The most common approach for $\mathcal{FP}$-satisfiability is to encode $\mathcal{FP}$ expressions into $\mathcal{BV}$ formulas based on the circuits used to implement floating-point operations, using appropriate under- and over-approximation schemes—or a mixture of both—to improve performance [15,45–47]. Then, the $\mathcal{BV}$-Solver is used to deal with the $\mathcal{FP}$ formula, using either the *eager* or the *lazy* $\mathcal{BV}$ approach. An alternative approach, based on *abstract interpretation*, is presented in [12,13,27]. With this technique, called *Abstract CDCL* (ACDCL), the set of feasible solutions is over-approximated with floating-point intervals, so that intervals-based conflict analysis is performed to decide $\mathcal{FP}$-satisfiability.

3 Theoretical Framework

We first present our generalization of [32] to the case of signed Bit-Vector Optimization (Sect. 3.1), and then move on to deal with Floating-Point Optimization (Sect. 3.2).

3.1 Bit-Vector Optimization

Without any loss of generality, we assume that every objective function $f(...)$ is replaced by a variable obj of the same type by conjoining “obj $= f(...)$” to the input formula. We use the symbol n to denote the bit-width of obj, and obj$[i]$ to denote the i-th bit of obj, where obj$[0]$ and obj$[n-1]$ are the Most Significant Bit (MSB) and the Least Significant Bit (LSB) of obj respectively.[4] We define the *Bit-Vector Optimization problem* as follows.

Definition 1 (OMT$_{[\mathcal{BV}]}(\mathcal{BV} \cup \mathcal{T})$) Let φ be a SMT($\mathcal{BV} \cup \mathcal{T}$) formula for some (possibly empty) theory $\mathcal{T}$ and obj be a—signed or unsigned—$\mathcal{BV}$ variable occurring in φ. We call an **Optimization Modulo $\mathcal{BV}$problem for** $\mathcal{BV} \cup \mathcal{T}$, OMT**($\boldsymbol{\mathcal{BV}} \cup \boldsymbol{\mathcal{T}}$)**, the problem of finding a $\mathcal{BV} \cup \mathcal{T}$-model $\mathcal{M}$ for φ (if any) whose value of obj is a minimum wrt. the total order relation $\leq_n$ for signed $\mathcal{BV}$s if obj is signed, and the one for unsigned $\mathcal{BV}$s otherwise. (The dual definition where we look for the *maximum* follows straightforwardly)

[4] Same as with Footnote 3.

Notice that the definition is independent on the extra theory $\mathcal{T}$, provided that obj is a $\mathcal{BV}$ term. (In practice $\mathcal{T}$ may be empty, or contain $\mathcal{FP}$ or/and other theories like e.g. that of arrays.) Hereafter, unless otherwise specified and when it is not necessary to make $\mathcal{T}$ explicit, we will abbreviate "$\text{OMT}_{[\mathcal{BV}]}(\mathcal{BV} \cup \mathcal{T})$" into "$\text{OMT}_{[\mathcal{BV}]}$".

We generalize the unsigned $\mathcal{BV}$ maximization procedures in [32] to the case of signed and unsigned $\mathcal{BV}$ optimization. To this extent, we introduce the novel notion of *$\mathcal{BV}$ attractor*.

Definition 2 (*Attractor, attractor equalities*). When minimizing [resp. maximizing], we call **attractor** for obj the smallest [resp. greatest] $\mathcal{BV}$-value $attr$ of the sort of obj. We call **vector of attractor equalities** the vector A s.t. $A[k] \overset{\text{def}}{=} (\text{obj}[k] = attr[k])$, $k \in [0..n-1]$.

Example 3 If obj$^{[8]}$ is an *unsigned $\mathcal{BV}$* objective of width 8, then its corresponding attractor $attr$ is $\mathbf{0}^{[8]}$, i.e. [00000000], when obj$^{[8]}$ is minimized and it is $\mathbf{255}^{[8]}$, i.e. [11111111], when obj$^{[8]}$ is maximized. When obj$^{[8]}$ is instead a *signed $\mathcal{BV}$* objective, following the two's complement encoding, the corresponding $attr$ is $-\mathbf{128}^{[8]}$, i.e. [10000000], for minimization and $\mathbf{127}^{[8]}$, i.e. [01111111], for maximization. ◇

In essence, the *attractor* can be seen as the target value of the optimization search and therefore it can be used to determine the desired improvement direction and to guide the decisions taken by the optimization search. By construction, if a model $\mathcal{M}$ satisfies all equalities $A[i]$, then the evaluation of obj in $\mathcal{M}$ is $attr$.

We use the symbol μ_k to denote a generic (possibly partial) assignment which assigns at least the k most-significant bits of obj. We use the symbol τ_k to denote an assignment to the k most-significant bits of obj. Given $i < k$, we denote by $\mu_k[i]$[resp.$\tau_k[i]$] the value in $\{0, 1\}$ assigned to obj[i] by μ_k[resp.τ_k]. Moreover, we use the expression $[\![\mu_k]\!]_i$ where $i \le k$ to denote the restriction of μ_k to the i most-significant bits of obj, obj[0], ..., obj[$i-1$]. Given a model $\mathcal{M}$ of φ and a variable v, we denote by $\mathcal{M}(v)$ the evaluation of v in $\mathcal{M}$. With a little abuse of notation, and when this does not cause ambiguities, we sometimes use an attractor equality $A[i] \overset{\text{def}}{=} (\text{obj}[i] = attr[i])$ to denote the single-bit assignment obj[i] := $attr[i]$ and we use its negation $\neg A[i]$ to denote the assignment to the complement value obj[i] := $\overline{attr[i]}$.

Definition 3 (lexicographic maximization) Consider an $\text{OMT}_{[\mathcal{BV}]}$ instance $\langle \varphi, \text{obj} \rangle$ and the vector of attractor equalities A. We say that an assignment τ_n to obj **lexicographically maximizes** A **wrt.** φ iff, for every $k \in [0..n-1]$,

- $\tau_n[k] = \overline{attr[k]}$ if $\varphi \wedge [\![\tau_n]\!]_k \wedge A[k]$ is unsatisfiable,
- $\tau_n[k] = attr[k]$ otherwise.

where $A[k]$ is the attractor equality (obj[k] $= attr[k]$). Given a model $\mathcal{M}$ for φ, we say that $\mathcal{M}$ lexicographically maximizes A wrt. φ iff its restriction to obj lexicographically maximizes A wrt. φ.

Starting from the MSB to the LSB, $\tau_n[resp.\mathcal{M}]$ in Definition 3 assigns to each obj[k] the value $attr[k]$ unless it is inconsistent wrt. φ and the assignments to the previous obj[i]s, $i \in [0..k-1]$.

Notice that this corresponds to the minimization of $\sum_{k=0}^{n-1} 2^{n-1-k} \cdot (\text{obj}[k]\ \mathbf{xor}_1\ attr[k])$ $[resp. maximization\ of\ \sum_{k=0}^{n-1} 2^{n-1-k} \cdot (\text{obj}[k]\ \mathbf{nxor}_1\ attr[k])]$—where $\mathbf{xor}_n$ is the bitwise-xor operator and $\mathbf{nxor}_n$ is its complement—because $2^{n-1-i} > \sum_{k=i+1}^{n-1} 2^{n-1-k}$ for every $n > i \ge 0$.[5]

[5] This reduces to the well-known inequality $2^m > \sum_{j=0}^{m-1} 2^j$ for $m \overset{\text{def}}{=} n-i-1$ and $j \overset{\text{def}}{=} n-1-k$.

The following fact derives from the above definitions and the properties of two's complement representation adopted by the SMT- LIBv2 standard for signed $\mathcal{BV}$. [6]

Theorem 1 *An optimal solution of an OMT$_{[\mathcal{BV}]}$ problem $\langle \varphi, obj\rangle$ is any model $\mathcal{M}$ of φ which* lexicographically maximizes *the vector of attractor equalities A.*

Proof (We investigate the minimization case, since the maximization case is dual.)

In the case of minimization with *unsigned $\mathcal{BV}$*, *attr* is [00...00], so that the lexicographic maximization of A corresponds to minimize $\sum_{k=0}^{n-1} 2^{n-1-k} \cdot \text{obj}[k]$ which is the standard minimization for unsigned $\mathcal{BV}$.

In the case of minimization with *signed $\mathcal{BV}$*, *attr* is [10...00], so that the lexicographic maximization of A corresponds to minimize $2^{n-1} \cdot \overline{\text{obj}[0]} + \sum_{k=1}^{n-1} 2^{n-1-k} \cdot \text{obj}[k]$ which—by means of subtracting the constant value 2^{n-1}—is equivalent to minimize $-2^{n-1} \cdot \text{obj}[0] + \sum_{k=1}^{n-1} 2^{n-1-k} \cdot \text{obj}[k]$, which is the standard minimization for two's complement $\mathcal{BV}$. □

Definitions 2 and 3 with Theorem 1 suggest thus a direct extension to the minimization/maximization of *signed $\mathcal{BV}$* of the algorithm for unsigned $\mathcal{BV}$ in [32]: *apply the unsigned-$\mathcal{BV}$maximization [resp. minimization] algorithm of* [32] *to the objective* $\text{obj}' \overset{\text{def}}{=} (\text{obj } \mathbf{nxor}_n\ attr)[resp.\text{obj}' \overset{\text{def}}{=} (\text{obj } \mathbf{xor}_n\ attr)]$ *instead of simply to* obj $[resp.\overline{\text{obj}}]$.

Example 4 Let $\text{obj}^{[3]}$ be a signed 3-bit $\mathcal{BV}$ goal to be minimized and $attr \overset{\text{def}}{=} [100]$ (i.e. $-\mathbf{4}^{[3]}$) be its attractor, and $A \overset{\text{def}}{=} [\text{obj}[0] = 1, \text{obj}[1] = 0, \text{obj}[2] = 0]$ be the corresponding vector of attractor equalities. Consider the three assignments

$$\tau_3 \overset{\text{def}}{=} \{A[0], \neg A[1], \neg A[2]\} \quad (\text{for which}\text{obj}^{[3]=[111],\text{i.e}-\mathbf{1}^{[3]}})$$
$$\tau_3' \overset{\text{def}}{=} \{\neg A[0], A[1], A[2]\} \quad (\text{for which}\text{obj}^{[3]} = [000], \text{i.e}-\mathbf{0}^{[3]})$$
$$\tau_3'' \overset{\text{def}}{=} \{A[0], \neg A[1], A[2]\} \quad (\text{for which}\text{obj}^{[3]} = [110], \text{i.e } -\mathbf{2}^{[3]})$$

Then τ_3 is lexicographically better than τ_3', because τ_3 satisfies the *attractor equality* corresponding to the MSB whereas τ_3' does not; τ_3 is lexicographically worse than τ_3'' because–all the rest being equal—τ_3'' makes the *attractor equality* ($\text{obj}[2] = 0$) true. Indeed, τ_3 is nearer in value to the attractor than τ_3' and is farther in value than τ_3''. ◇

3.2 Floating-Point Optimization

We define the *Floating-Point Optimization problem* as follows.

Definition 4 (OMT$_{[\mathcal{FP}]}(\mathcal{FP} \cup \mathcal{T})$) Let φ be a SMT($\mathcal{FP} \cup \mathcal{T}$) formula for some (possibly empty) theory $\mathcal{T}$ and obj be a $\mathcal{FP}$ variable occurring in φ. We call an **Optimization Modulo $\mathcal{FP}$problem for $\mathcal{FP} \cup \mathcal{T}$,OMT$_{[\mathcal{FP}]}(\mathcal{FP} \cup \mathcal{T})$** the problem of finding a $\mathcal{FP} \cup \mathcal{T}$-model $\mathcal{M}$ for φ (if any) whose value of obj, is either

- minimum wrt. the usual total order relation $\leq$ for $\mathcal{FP}$ numbers, if φ is satisfied by at least one model $\mathcal{M}'$ s.t. $\mathcal{M}'$(obj) is not NaN,
- some binary representation of NaN, otherwise.

(The dual definition where we look for the *maximum* follows straightforwardly.)

[6] If the standard adopted were the sign-and-magnitude binary encoding, then Theorem 1 would not hold. Nevertheless, in such a case we could use a simplified version of the technique for $\mathcal{FP}$ optimization in Sect. 3.2.

As with $\mathcal{BV}$, the definition is independent on the extra theory $\mathcal{T}$, provided that obj is a $\mathcal{FP}$ term. In practice $\mathcal{T}$ may be empty, or contain $\mathcal{BV}$ or/and other theories like e.g. that of arrays. Hereafter, unless otherwise specified and when it is not necessary to make $\mathcal{T}$ explicit, we will abbreviate "$\text{OMT}_{[\mathcal{FP}]}(\mathcal{FP} \cup \mathcal{T})$" into "$\text{OMT}_{[\mathcal{FP}]}$".

Definition 4 is necessarily convoluted because obj can be NaN. In fact, in the SMT- LIBv2 standard the comparisons $\{\leq, <, \geq, >\}$ between NaN and any other $\mathcal{FP}$ value are always evaluated false because NaN has multiple representations at the binary level (see Table 1). Also, requiring the optimal solution to be always different from NaN makes the resulting $\text{OMT}_{[\mathcal{FP}]}$ problem $\langle\varphi \wedge \neg\mathsf{IsNaN(obj)}, \text{obj}\rangle$ unsatisfiable when φ is satisfied only by models $\mathcal{M}$ s.t. $\mathcal{M}(\text{obj})$ is NaN. For these reasons, we admit NaN as the optimal solution value for obj if and only if φ is satisfied only by models $\mathcal{M}$ s.t. $\mathcal{M}(\text{obj})$ is NaN.

In the rest of this section we assume that we have already checked, in sequence, that

(i) the input formula φ is satisfiable—by invoking an SMT($\mathcal{FP}$) solver on φ. If the solver returns UNSAT, then there is no need to proceed;

(ii) φ is satisfied by at least one model $\mathcal{M}'$ s.t. $\mathcal{M}'(\text{obj})$ is not NaN—by invoking an SMT($\mathcal{FP}$) solver on $\varphi \wedge \neg\mathsf{IsNaN(obj)}$ if the model $\mathcal{M}$ returned by the previous SMT call is s.t. $\mathcal{M}(\text{obj})$ is NaN. If the solver returns UNSAT, then we conclude that the minimum is NaN.

Thus, we can safely focus our investigation on the restricted $\text{OMT}_{[\mathcal{FP}]}$ problem $\langle\varphi_{\mathsf{noNaN}}, \text{obj}\rangle$, where $\varphi_{\mathsf{noNaN}} \overset{\text{def}}{=} \varphi \wedge \neg\mathsf{IsNaN(obj)}$, knowing it is satisfiable.

In Sect. 3.1, we have introduced the concept of a $\mathcal{BV}$ attractor, showing how this value can be used to drive the optimization search towards the optimum value, when minimizing or maximizing a *signed* or *unsigned* $\mathcal{BV}$ goal. However, in the case of floating-point optimization, it is not possible to statically determine the attractor value in advance, before the search is even started. This is due to the more complex representation of $\mathcal{FP}$ variables, which uses three separate Bit-Vectors (i.e. sign, exponent and significand), and the presence of various classes of special values (i.e. zeros, infinity, NaN), which make Definition 2 ambiguous for $\mathcal{FP}$ optimization. We illustrate this problem with the following example.

Example 5 Let $\langle\varphi_{\mathsf{noNaN}}, \text{obj}\rangle$ be an $\text{OMT}_{[\mathcal{FP}]}$ problem where obj is a $\mathcal{FP}$ objective, of sort `(_ FP 3 5)`, to be minimized. To make our explanation easier to follow, we show in Table 1 a short list of sample values for an $\mathcal{FP}$ variable of the same sort as obj. Each $\mathcal{FP}$ value is represented as a triplet of bit-vectors $\langle$**sign**, **exp**, **sig**$\rangle$—following the SMT- LIBv2 conventions described in Sect. 2—and also in decimal notation.

From Table 1, we immediately notice that the binary representation of both the exponent and the significant of a Floating-Point number grows in opposite directions in the positive and in the negative domains. In addition, by sorting the values according to their binary representation, we observe that $-\infty$ [resp. $+\infty$] is not the smallest [resp. greatest] representable $\mathcal{FP}$ value in the negative [resp. positive] domain. In fact, both extreme ends of the table are occupied by NaN, which has multiple binary representations.

In what follows, we temporarily disregard the effects of unit-propagation, which might assign some (or all) bits of obj as a result of some constraints in φ_{noNaN}, and pick some values as candidate attractors for an $\mathcal{FP}$ goal to be minimized.

Assume that the optimal value of the $\mathcal{FP}$ goal is the sub-normal $\mathcal{FP}$ value `(fp #b1 #b000 #b1111)` (i.e. $\frac{-15}{64}$). Suppose that the attractor is chosen to be equal to the value $-\infty$ listed at row 9 in Table 1, which is the smallest $\mathcal{FP}$ value wrt. total order relation $\leq$ for $\mathcal{FP}$ numbers. Then, it can be seen that after both the sign and the exponent bits have been decided to be equal `#b1` and `#b000` respectively, the remaining bits of the attractor pull the search in the wrong direction, that is, towards 0^-.$\diamond$

Table 1 Sample values for a $\mathcal{FP}$ variable with sort `(_ FP 3 5)`

	Sign	Exp	Sig	Value
1	#b0	#b111	#b1111	NaN
	...	...	...	NaN
2	#b0	#b111	#b0000	$+\infty$
3	#b0	#b110	#b1111	$\frac{31}{2}$
	...	...	...	...
4	#b0	#b000	#b0001	$\frac{1}{64}$
5	#b0	#b000	#b0000	0^+
6	#b1	#b000	#b0000	0^-
7	#b1	#b000	#b0001	$-\frac{1}{64}$
	...	...	...	...
8	#b1	#b110	#b1111	$-\frac{31}{2}$
9	#b1	#b111	#b0000	$-\infty$
	...	...	...	NaN
10	#b1	#b111	#b1111	NaN

Selecting a different $\mathcal{FP}$ value as candidate attractor would not solve the problem; rather, it would result in a different set of issues. For instance, an attractor equal to the NaN value listed at row 10 in Table 1, which is the smallest representable $\mathcal{FP}$ value according to the binary ordering, would solve the problem for the previous case in which the optimum $\mathcal{FP}$ value is `(fp #b1 #b000 #b1111)`. However, this attractor would remain an unsuitable choice for an OMT$_{[\mathcal{FP}]}$ instance where obj is forced to be positive, because after the sign bit of the objective function has been decided to be equal `#b0` the remaining bits of the attractor drive the search in the wrong direction, that is, towards $+\infty$. $\diamond$

Since there is no statically-determined $\mathcal{FP}$ value that can be used as an attractor when dealing with floating-point optimization, we introduce the new concept of *dynamic attractor*.

Definition 5 (Dynamic attractor) Let $\langle \varphi_{\mathsf{noNaN}}, \text{obj}\rangle$ be a restricted OMT$_{[\mathcal{FP}]}$ problem, where $\varphi_{\mathsf{noNaN}} \overset{\text{def}}{=} \varphi \wedge \neg\mathsf{IsNaN}(\mathsf{obj})$ is a satisfiable SMT($\mathcal{FP}$) formula and obj is a $\mathcal{FP}$ objective to be minimized [resp. maximized]. Let $k \in [0..n]$ and τ_k be an assignment to the k most-significant bits of obj.

Then, we say that an $\mathcal{FP}$-value $attr_{\tau_k}$ for obj is a **dynamic attractor for** obj**wrt.** τ_k iff it is the smallest [resp. largest] $\mathcal{FP}$ value different from NaN s.t. the k most-significant bits of $attr_{\tau_k}$ have the same value of the k most-significant bits of obj in τ_k. We call **vector of attractor equalities** the vector A_{τ_k} s.t. $A_{\tau_k}[i] \overset{\text{def}}{=} (\text{obj}[i] = attr_{\tau_k}[i])$, $i \in [0..n-1]$.

The following fact derives from the above definitions and the properties of IEEE 754-2008 standard representation adopted by SMT- LIBv2 standard for $\mathcal{FP}$.

Lemma 1 *Let $\langle \varphi_{noNaN}, obj\rangle$ be a restricted minimization $[resp.maximization]$ OMT$_{[\mathcal{FP}]}$ problem, let τ_k be an assignment to $obj[0]...obj[k-1]$ and $attr_{\tau_k}$ be its corresponding dynamic attractor, for some $k \in [0..n-1]$. Let $\tau_{k+1} \overset{def}{=} \tau_k \cup \{obj[k] := attr_{\tau_k}[k]\}$ and $\tau'_{k+1} \overset{def}{=} \tau_k \cup \{obj[k] := \overline{attr_{\tau_k}[k]}\}$, and let $\mathcal{M}$, $\mathcal{M}'$ two models for φ_{noNaN} which extend τ_{k+1} and τ'_{k+1} respectively.*

Then $\mathcal{M}(obj) \le \mathcal{M}'(obj)$ $[resp.\mathcal{M}(obj) \ge \mathcal{M}'(obj)]$.

Proof (We prove the case of minimization, since the case of maximization is dual wrt. the value of the sign bit.) We distinguish three cases based on the value of k.

Case $k = 0$ **(sign bit)**. Then $attr_{\tau_0}[0] = 1$, $\tau_1 = \{\text{obj}[0] = 1\}$ and $\tau_1' = \{\text{obj}[0] = 0\}$, where obj[0] is the MSB of obj and represents the sign of the floating-point value. Then obj is smaller or equal zero in every model $\mathcal{M}$ and larger or equal zero in every model $\mathcal{M}'$ of φ_{noNaN}, so that $\mathcal{M}(\text{obj}) \leq \mathcal{M}'(\text{obj})$ is verified.

Case $k \in [1..ebits]$ **(exponent bits)**, where $ebits$ is the number of bits in the exponent of obj. Then, $attr_{\tau_k}[k]$ is 1 if $\tau_k[0] = 1$ and 0 otherwise.

In the first case, obj can only be negative-valued in both $\mathcal{M}$ and $\mathcal{M}'$. More precisely, $\mathcal{M}(\text{obj})$ can be either $-\infty$ or a normal negative value, whereas $\mathcal{M}'(\text{obj})$ can be either a normal or a sub-normal negative value. Hereafter, we consider only the case in which both have a normal negative value, because the case in which $\mathcal{M}(\text{obj}) = -\infty$ or $\mathcal{M}'(\text{obj})$ is sub-normal are both trivial, given that the absolute value of any sub-normal $\mathcal{FP}$ number is smaller than the absolute value of any non-zero normal $\mathcal{FP}$ number. Furthermore, we disregard the significand bits in $\mathcal{M}$ and $\mathcal{M}'$ because their contribution to the value of obj is always less significant than that of the bits in the exponent. Given these premises, the exponent value of obj in every possible $\mathcal{M}$ is larger than the exponent of obj in every possible $\mathcal{M}'$ by a value equal to $2^{ebits-k}$ and therefore, given that both $\mathcal{M}(\text{obj})$ and $\mathcal{M}'(\text{obj})$ are negative-valued, $\mathcal{M}(\text{obj}) \leq \mathcal{M}'(\text{obj})$.

The case in which $\tau_k[0] = 0$, that is when obj can only be positive-valued in both $\mathcal{M}$ and $\mathcal{M}'$, is dual.

Case $k > ebits$ **(significand bits)**. Then there are three sub-cases.

If for every $i \in [1..ebits]$ the value of $\tau_k[i]$ is equal 1, then the only possible value of $\mathcal{M}(\text{obj})$ for every possible $\mathcal{M}$ is $+\infty$, and therefore $attr_{\tau_k}[k] = 0$. On the other hand, there exists no possible model $\mathcal{M}'$ of φ_{noNaN}, because the assignment $\text{obj}[k] = 1$ would imply obj being equal to NaN, so that the statement $\mathcal{M}(\text{obj}) \leq \mathcal{M}'(\text{obj})$ is vacuously true.

If instead there is some $i \in [1..ebits]$ s.t. $\tau_k[i] = 0$, then $attr_{\tau_k}[k]$ is 1 if $\tau_k[0] = 1$ (i.e. obj is negative-valued) and 0 otherwise (i.e. obj is positive-valued). In both cases, we can disregard the exponent bits in $\mathcal{M}$ and $\mathcal{M}'$ because their contribution to the value of obj is the same in either model. For the same reasons, since $\mathcal{M}(\text{obj})$ and $\mathcal{M}'(\text{obj})$ can only be either both normal or both sub-normal, we can ignore the contribution of the leading hidden bit and focus on the bits of the significand.

When $\tau_k[0] = 1$ and obj must be negative-valued, the decimal value of the significand in $\mathcal{M}$ is larger than the decimal value of every possible significand in $\mathcal{M}'$ by exactly $2^{-(k-ebits)}$. Given that both $\mathcal{M}(\text{obj})$ and $\mathcal{M}'(\text{obj})$ are negative-valued, we have that $\mathcal{M}(\text{obj}) \leq \mathcal{M}'(\text{obj})$.

The case in which $\tau_k[0] = 0$, that is when obj can only be positive-valued in both $\mathcal{M}$ and $\mathcal{M}'$, is dual. □

Notice that Lemma 1 states "$\mathcal{M}(\text{obj}) \leq \mathcal{M}'(\text{obj})$" and not "$\mathcal{M}(\text{obj}) < \mathcal{M}'(\text{obj})$" because, e.g., we may have $\mathcal{M}(\text{obj}) = 0^-$ and $\mathcal{M}'(\text{obj}) = 0^+$, and $(0^- < 0^+)$ is false in $\mathcal{FP}$.

Lemma 1 states that, given the current assignment τ_k to the k most-significant-bits of obj, $\text{obj}[k] = attr_{\tau_k}[k]$ is always the best extension of τ_k to the next bit (when consistent). A dynamic attractor $attr_{\tau_k}$ can thus be used by the optimization search to guide the assignment of the $k+1$-th bit of obj towards the direction of maximum gain which is allowed by τ_k, so that to obtain the "best" extension τ_{k+1} of τ_k. Once the (new) assignment τ_{k+1} is found, the OMT solver can compute the dynamic attractor $attr_{\tau_{k+1}}$ for obj wrt. τ_{k+1} and then use it to assign the $k+2$-th bit of obj, and so on.

Let $\langle \varphi_{\mathsf{noNaN}}, \text{obj} \rangle$ be an $\text{OMT}_{[\mathcal{FP}]}$ instance, s.t. obj is a $\mathcal{FP}$ variable of n bits, and τ_0 be an initially empty assignment. If at each step of the optimization search the assignment of the k-th bit of obj is guided by the dynamic attractor for obj wrt. τ_k, then the corresponding sequence of n dynamic attractors (of increasing order k) is unique and depends exclusively on φ_{noNaN}. Intuitively, this is the case because the (current) dynamic attractor always points in the direction of maximum gain. We illustrate this in the following example.

Example 6 Let $\langle \varphi_{\mathsf{noNaN}}, \text{obj}\rangle$ be an $\text{OMT}_{[\mathcal{FP}]}$ problem where obj is a $\mathcal{FP}$ objective, of sort `(_ FP 3 5)`, to be minimized, as in Example 5. At the beginning of the search, nothing is known about the structure of the solution. Therefore, $\tau_0 = \emptyset$ and, since obj is being minimized, the dynamic attractor $attr_{\tau_0}$ for obj wrt. τ_0 is `(fp #b1 #b111 #b0000)` (i.e. $-\infty$), which gives a preference to any feasible value of obj in the negative domain.

If we discover that the domain of the objective function can only be positive, so that the first bit of obj is permanently set to 0 in τ_1, then the new dynamic attractor for obj wrt. τ_1 (i.e. $attr_{\tau_1}$) is equal to `(fp #b0 #b000 #b0000)` (i.e. 0^+). Otherwise, $attr_{\tau_i}$ remains $-\infty$ until, e.g., we discover there is no solution ≤ -8 so that the second bit in the exponent is forced to 0. Then $attr_{\tau_3}$ becomes `(fp #b1 #b101 #b1111)` (i.e., $\frac{-31}{4}$).) Notice that all significand bits in the attractor pass from 0 to 1 because now we have a finite solution. ⋄

Definition 6 (Attractor trajectory $\mathcal{A}_\varphi$) We consider the restricted $\text{OMT}_{[\mathcal{FP}]}$ problem $\langle \varphi_{\mathsf{noNaN}}, \text{obj}\rangle$ s.t. $\varphi_{\mathsf{noNaN}} \overset{\text{def}}{=} \varphi \wedge \neg\mathsf{IsNaN(obj)}$ as in Definition 5, a triplet of inductively-defined sequences $\langle\{\tau_0, \tau_1, ..., \tau_n\}, \{attr_{\tau_0}, attr_{\tau_1}, ..., attr_{\tau_n}\}, \{A_{\tau_0}, A_{\tau_1}, ..., A_{\tau_n}\}\rangle$—where each τ_k is an assignment to the first k most-significant bits of obj s.t. $\tau_k \subset \tau_{k+1}$, $attr_{\tau_k}$ is its corresponding dynamic attractor and A_{τ_k} is its corresponding *vector of attractor equalities*—so that, for every $k \in [0..n-1]$:

(i) $\tau_{k+1}[k] = \overline{attr_{\tau_k}[k]}$ if $\varphi_{\mathsf{noNaN}} \wedge \tau_k \wedge A_{\tau_k}[k]$ is unsatisfiable,
(ii) $\tau_{k+1}[k] = attr_{\tau_k}[k]$ otherwise.

Then we define the **attractor trajectory** $\dashv_\varphi$ as the vector $[A_{\tau_0}[0], ..., A_{\tau_{n-1}}[n-1]]$.

The attractor trajectory $\dashv_\varphi$ contains those attractor equalities ($\text{obj}[k] = attr_{\tau_k}[k]$) which are of critical importance for the decisions taken by the optimization search. Intuitively, this is the case because the value of the k-th bit of obj (i.e. obj$[k]$) is still undecided in τ_k.

Example 7 Let $\langle \varphi_{\mathsf{noNaN}}, \text{obj}\rangle$ be a restricted $\text{OMT}_{[\mathcal{FP}]}$ problem where obj is a $\mathcal{FP}$ objective, of sort `(_ FP 3 5)`, to be minimized, as in Example 5. We consider the case in which the input formula φ_{noNaN} requires obj to be larger or equal $\frac{29}{2}$ and it does not impose any other constraint on the value of obj. Given the sequence of (partial) assignments $\tau_0, ..., \tau_8$ in Fig. 1, the corresponding list of dynamic attractors and the corresponding vectors of attractor equalities, then the attractor trajectory $\mathcal{A}_\varphi$ is equal to the vector $[\text{obj}[0] = 1, \text{obj}[1] = 0, \text{obj}[2] = 0, \text{obj}[3] = 0, \text{obj}[4] = 0, \text{obj}[5] = 0, \text{obj}[6] = 0, \text{obj}[7] = 0]$. ⋄

Lemma 2 *Consider* $\langle \varphi_{noNaN}, obj\rangle$, $\tau_0, ..., \tau_n$, $attr_{\tau_0}, ..., attr_{\tau_n}$, $A_{\tau_0}, ..., A_{\tau_n}$, *and* $\dashv_\varphi$ *as in Definition 6. Then* τ_n *lexicographically maximizes* $\dashv_\varphi$ *wrt.* φ_{noNaN}.

Proof By Definition 6, we have that, for each $k \in [0..n-1]$,

(i) $\tau_{k+1}[k] = \overline{attr_{\tau_k}[k]}$ if $\varphi_{\mathsf{noNaN}} \wedge \tau_k \wedge A_{\tau_k}[k]$ is unsatisfiable,
(ii) $\tau_{k+1}[k] = attr_{\tau_k}[k]$ otherwise.

By construction, $\tau_k = [\![\tau_n]\!]_k$. Therefore, we can replace τ_k with $[\![\tau_n]\!]_k$ so that

(i) $[\![\tau_n]\!]_{k+1}[k] = \overline{attr_{[\![\tau_n]\!]_k}[k]}$ if $\varphi_{\mathsf{noNaN}} \wedge [\![\tau_n]\!]_k \wedge A_{[\![\tau_n]\!]_k}[k]$ is unsatisfiable,
(ii) $[\![\tau_n]\!]_{k+1}[k] = attr_{[\![\tau_n]\!]_k}[k]$ otherwise.

$$
\begin{array}{llll}
\tau_0 = \emptyset & attr_{\tau_0} = \texttt{(fp #b1 #b111 #b0000)} = [\underline{1}.111.0000] & [-\infty] & \Rightarrow \text{UNSAT} \\
\tau_1 = \tau_0 \cup \{\mathsf{obj}[0] = 0\} & attr_{\tau_1} = \texttt{(fp #b0 #b000 #b0000)} = [0.\underline{0}00.0000] & [0^+] & \Rightarrow \text{UNSAT} \\
\tau_2 = \tau_1 \cup \{\mathsf{obj}[1] = 1\} & attr_{\tau_2} = \texttt{(fp #b0 #b100 #b0000)} = [0.1\underline{0}0.0000] & [+2] & \Rightarrow \text{UNSAT} \\
\tau_3 = \tau_2 \cup \{\mathsf{obj}[2] = 1\} & attr_{\tau_3} = \texttt{(fp #b0 #b110 #b0000)} = [0.11\underline{0}.0000] & [+8] & \Rightarrow \text{SAT} \\
\tau_4 = \tau_3 \cup \{\mathsf{obj}[3] = 0\} & attr_{\tau_4} = \texttt{(fp #b0 #b110 #b0000)} = [0.110.\underline{0}000] & [+8] & \Rightarrow \text{UNSAT} \\
\tau_5 = \tau_4 \cup \{\mathsf{obj}[4] = 1\} & attr_{\tau_5} = \texttt{(fp #b0 #b110 #b1000)} = [0.110.1\underline{0}00] & [+12] & \Rightarrow \text{UNSAT} \\
\tau_6 = \tau_5 \cup \{\mathsf{obj}[5] = 1\} & attr_{\tau_6} = \texttt{(fp #b0 #b110 #b1100)} = [0.110.11\underline{0}0] & [+14] & \Rightarrow \text{SAT} \\
\tau_7 = \tau_6 \cup \{\mathsf{obj}[6] = 0\} & attr_{\tau_7} = \texttt{(fp #b0 #b110 #b1100)} = [0.110.110\underline{0}] & [+14] & \Rightarrow \text{UNSAT} \\
\tau_8 = \tau_7 \cup \{\mathsf{obj}[7] = 1\} & attr_{\tau_8} = \texttt{(fp #b0 #b110 #b1101)} = [0.110.1101] & [{}^{29}/_{2}] &
\end{array}
$$

$$
\begin{array}{ll}
A_{\tau_0} = & [\underline{\mathsf{obj}[0] = 1}, \mathsf{obj}[1] = 1, \mathsf{obj}[2] = 1, \mathsf{obj}[3] = 1, \mathsf{obj}[4] = 0, \mathsf{obj}[5] = 0, \mathsf{obj}[6] = 0, \mathsf{obj}[7] = 0] \\
A_{\tau_1} = & [\mathsf{obj}[0] = 0, \underline{\mathsf{obj}[1] = 0}, \mathsf{obj}[2] = 0, \mathsf{obj}[3] = 0, \mathsf{obj}[4] = 0, \mathsf{obj}[5] = 0, \mathsf{obj}[6] = 0, \mathsf{obj}[7] = 0] \\
A_{\tau_2} = & [\mathsf{obj}[0] = 0, \mathsf{obj}[1] = 1, \underline{\mathsf{obj}[2] = 0}, \mathsf{obj}[3] = 0, \mathsf{obj}[4] = 0, \mathsf{obj}[5] = 0, \mathsf{obj}[6] = 0, \mathsf{obj}[7] = 0] \\
A_{\tau_3} = & [\mathsf{obj}[0] = 0, \mathsf{obj}[1] = 1, \mathsf{obj}[2] = 1, \underline{\mathsf{obj}[3] = 0}, \mathsf{obj}[4] = 0, \mathsf{obj}[5] = 0, \mathsf{obj}[6] = 0, \mathsf{obj}[7] = 0] \\
A_{\tau_4} = & [\mathsf{obj}[0] = 0, \mathsf{obj}[1] = 1, \mathsf{obj}[2] = 1, \mathsf{obj}[3] = 0, \underline{\mathsf{obj}[4] = 0}, \mathsf{obj}[5] = 0, \mathsf{obj}[6] = 0, \mathsf{obj}[7] = 0] \\
A_{\tau_5} = & [\mathsf{obj}[0] = 0, \mathsf{obj}[1] = 1, \mathsf{obj}[2] = 1, \mathsf{obj}[3] = 0, \mathsf{obj}[4] = 1, \underline{\mathsf{obj}[5] = 0}, \mathsf{obj}[6] = 0, \mathsf{obj}[7] = 0] \\
A_{\tau_6} = & [\mathsf{obj}[0] = 0, \mathsf{obj}[1] = 1, \mathsf{obj}[2] = 1, \mathsf{obj}[3] = 0, \mathsf{obj}[4] = 1, \mathsf{obj}[5] = 1, \underline{\mathsf{obj}[6] = 0}, \mathsf{obj}[7] = 0] \\
A_{\tau_7} = & [\mathsf{obj}[0] = 0, \mathsf{obj}[1] = 1, \mathsf{obj}[2] = 1, \mathsf{obj}[3] = 0, \mathsf{obj}[4] = 1, \mathsf{obj}[5] = 1, \mathsf{obj}[6] = 0, \underline{\mathsf{obj}[7] = 0}] \\
A_{\tau_8} = & [\mathsf{obj}[0] = 0, \mathsf{obj}[1] = 1, \mathsf{obj}[2] = 1, \mathsf{obj}[3] = 0, \mathsf{obj}[4] = 1, \mathsf{obj}[5] = 1, \mathsf{obj}[6] = 0, \mathsf{obj}[7] = 1]
\end{array}
$$

Fig. 1 An example of $\mathcal{FP}$ optimization using the dynamic attractor. ("[...]" denotes the value of the attractor $attr_{\tau_i}$. "$\Longrightarrow$ SAT/UNSAT" denotes the satisfiability of $\varphi_{\mathsf{noNaN}} \wedge \tau_k \wedge A_{\tau_k}[k]$. For ease of illustration, we have underlined the critical bit $attr_{\tau_k}[k]$ in the attractors and each attractor equality of the attractor trajectory $\dashv_\varphi$ inside the vectors of attractor equalities.)

We notice the following facts. For each $k \in [0..n-1]$, $[\![\tau_n]\!]_k \subset \tau_n$. Furthermore, for each $k \in [0..n-1]$, $\mathcal{A}_\varphi k = A_{[\![\tau_n]\!]_k}[k]$ because $\mathcal{A}_\varphi k = A_{\tau_k}[k]$ by the definition of attractor trajectory, and $A_{\tau_k}[k] = A_{[\![\tau_n]\!]_k}[k]$ by the equality $\tau_k = [\![\tau_n]\!]_k$. Thus, we can replace $[\![\tau_n]\!]_{k+1}$ with τ_n and $A_{[\![\tau_n]\!]_k}[k]$ with $\mathcal{A}_\varphi k$, as follows. For each $k \in [0..n-1]$,

(i) $\tau_n[k] = \overline{attr_{\tau_n}[k]}$ if $\varphi_{\mathsf{noNaN}} \wedge [\![\tau_n]\!]_k \wedge \mathcal{A}_\varphi k$ is unsatisfiable,
(ii) $\tau_n[k] = attr_{\tau_n}[k]$ otherwise.

Hence, τ_n lexicographically maximizes $\mathcal{A}_\varphi$ wrt. φ_{noNaN}. □

Finally, we make the following two observations. The first is that the sequence $\tau_0, ..., \tau_n$ in Definition 6 can be iteratively constructed using its list of requirements, for instance, by means of a sequence of incremental calls to an SMT solver. The second, more important, observation is that τ_n corresponds to the assignment of values which makes obj optimal in φ_{noNaN}. Using the above definitions, we show that the following fact holds.

Theorem 2 *Let $\langle\varphi_{noNaN}, obj\rangle$, $\tau_0, ..., \tau_n$, $attr_{\tau_0}, ..., attr_{\tau_n}$, $A_{\tau_0}, ..., A_{\tau_n}$, and $\dashv_\varphi$ be as in Definition 6. Then, any model $\mathcal{M}$ of φ_{noNaN} which lexicographically maximizes the attractor trajectory $\dashv_\varphi$ is an optimal solution for the OMT$_{[\mathcal{FP}]}$ problem $\langle\varphi_{noNaN}, obj\rangle$.*

Proof (We prove the case of minimization, since the case of maximization is dual.)

By Lemma 2 we have that τ_n lexicographically maximize $\mathcal{A}_\varphi$. Let $\mathcal{M}$ be a model of φ_{noNaN} which lexicographically maximizes $\mathcal{A}_\varphi$, and let μ be its restriction to obj. Since both τ_n and

$\mathcal{M}$ lexicographically maximize $\mathcal{A}_\varphi$, from the uniqueness of τ_n, we immediately notice that $\mu = \tau_n$, so that $\tau_k = [\![\mu]\!]_k$ for each $k \in [0..n]$ and μ lexicographically maximize $\mathcal{A}_\varphi$.

By definition, $\mathcal{M}$ is an optimal solution for $\langle \varphi_{\mathsf{noNaN}}, \mathrm{obj} \rangle$ iff there exists no other model $\mathcal{M}'$ for it s.t. $\mathcal{M}'(\mathrm{obj}) < \mathcal{M}(\mathrm{obj})$. Hence, we show by contradiction that no such $\mathcal{M}'$ can exist.

Assume (for the sake of contradiction), that there exists a model $\mathcal{M}'$ for φ_{noNaN}, s.t. $\mathcal{M}'(\mathrm{obj}) < \mathcal{M}(\mathrm{obj})$, and let μ' be the restriction of $\mathcal{M}'$ to obj. Then there must be at least one index i for which $\mu[i] \neq \mu'[i]$. Let m be the smallest such index. Recalling that $\tau_m = [\![\mu]\!]_m$ and $\tau_{m+1} = [\![\mu]\!]_{m+1}$, we set $\tau'_{m+1} \overset{\text{def}}{=} [\![\mu']\!]_{m+1}$. Then, $\tau_m \subset \tau_{m+1}$, $\tau_m \subset \tau'_{m+1}$, $\tau_{m+1} \neq \tau'_{m+1}$. In particular, $\tau_{m+1}[m] = \overline{\tau'_{m+1}[m]}$ and therefore $\tau_{m+1}[m] = attr_{\tau_m}[m]$ if $\tau'_{m+1}[m] = \overline{attr_{\tau_m}[m]}$, and vice versa.

Then, we distinguish two cases.

In the first case, $\tau_{m+1}[m] = \overline{attr_{\tau_m}[m]}$ and $\tau'_{m+1}[m] = attr_{\tau_m}[m]$. From $\tau_{m+1}[m] = \overline{attr_{\tau_m}[m]}$ and the fact that μ lexicographically maximizes $\mathcal{A}_\varphi$, we derive that $\varphi_{\mathsf{noNaN}} \wedge \tau_m \wedge \dashv_\varphi[m]$ is unsatisfiable, where $\mathcal{A}_\varphi m \overset{\text{def}}{=} (\mathrm{obj}[m] = attr_{\tau_m}[m])$. Since $\tau_m \subset \tau'_{m+1} \subseteq \mu'$ and $\tau'_{m+1}[m] = attr_{\tau_m}[m]$, we conclude that $\varphi_{\mathsf{noNaN}} \wedge \mu' \models \bot$, so that $\mathcal{M}'$ cannot be a model of φ_{noNaN}, contradicting the initial assumption.

In the second case, $\tau_{m+1}[m] = attr_{\tau_m}[m]$ and $\tau_{m+1}[m] = \overline{attr_{\tau_m}[m]}$. Therefore, by Lemma 1, for every pair of models $\mathcal{M}_1, \mathcal{M}_2$ for φ_{noNaN} which extend respectively τ_{m+1} and τ'_{m+1} we have that $\mathcal{M}_1(\mathrm{obj}) \leq \mathcal{M}_2(\mathrm{obj})$. Since $\tau_{m+1} = [\![\mu]\!]_{m+1}$ and $\tau'_{m+1} = [\![\mu']\!]_{m+1}$, it follows that $\mathcal{M}'(\mathrm{obj}) \not< \mathcal{M}(\mathrm{obj})$, contradicting the initial assumption. □

4 OMT$_{[\mathcal{FP}]}$ Procedures

In this paper, we consider two approaches for dealing with OMT$_{[\mathcal{FP}]}$: a baseline linear/binary search, based on the inline OMT schema for $\mathcal{LAA}$ objectives presented in [39], and *Floating-Point Optimization with Binary Search* (OFP- BS), a brand-new engine inspired by the OBV- BS algorithm for unsigned bit-vectors in [32] and by Theorem 2 and relative definitions in Sect. 3.2.

4.1 OMT-Based Approach

The OMT-based approach for OMT$_{[\mathcal{FP}]}$ adapts the linear- and binary-search schemata for OMT with $\mathcal{LAA}$ objectives presented in [39] to deal with $\mathcal{FP}$ objectives.

In the basic linear-search schema, the optimization search is advanced by means of a sequence of linear cuts, each of which forces the OMT solver to look for a new model $\mathcal{M}'$ which improves the value of obj wrt. the most recent model $\mathcal{M}$. In the binary-search schema, instead, the OMT solver learns an incremental sequence of cuts which bisect the current domain of the objective function. For clarity, we recap here the essential elements of the binary-search schema presented in [38,39]. At the beginning of the optimization search and following each update of the lower- (lb) and upper- (ub) bounds of obj, the OMT solver computes a pivoting value $\mathsf{pivot} \overset{\text{def}}{=} \mathtt{floor}(\rho \cdot ub + (1-\rho) \cdot lb)$, for some value of ρ (e.g. $\frac{1}{2}$). If pivot lies inside the range $]lb, ub]$, a cut of the form $(\mathrm{obj} < \mathsf{pivot})$ is learned. Otherwise, if—due to rounding side-effects of $\mathcal{FP}$ operations—pivot lies outside the range $]lb, ub]$, a cut of the form $(\mathrm{obj} < ub)$ is learned instead. If the cut is satisfiable, the upper-bound of obj is updated with a new model value of obj. Otherwise, the lower-bound is made equal to pivot

[resp. ub]. The algorithm terminates when the search interval $[lb, ub[$ becomes empty. In general, it is reasonable to expect the binary-search schema to converge towards the optimal solution faster than the linear-search schema, because the feasible domain of a $\mathcal{FP}$ goal can be comprised by an exponentially large number of values (wrt. the bit-width of the cost function).

In either schema, whenever the optimization engine encounters for the first time a solution s.t. obj $=$ NaN, the OMT solver learns a unit-clause of the form $\neg$(isNaN(obj)) so as to look for an optimal solution different from NaN (if any).

When dealing with $\mathcal{FP}$ objectives, differently from the case of $\mathcal{LRA}$ in [39], it is not necessary to implement a specialized optimization procedure within the $\mathcal{FP}$-Solver in order to guarantee the termination of the optimization search. Indeed, such procedure is not available when Floating-Point terms are bit-blasted into bit-vectors *eagerly*, or when the ACDCL $\mathcal{FP}$-Solver is used, because by the time the optimization procedure is called the domain interval of any $\mathcal{FP}$ term contains a singleton value. Conversely, such a minimization procedure could be envisaged when the OMT solver uses a *lazy* $\mathcal{FP}$-Solver as back-end, so as to speed-up the convergence towards the optimal solution[7].

4.2 Floating-Point Optimization with Binary Search

The *Floating-Point Optimization with Binary Search* algorithm, OFP- BS, is a new engine for $\text{OMT}_{[\mathcal{FP}]}(\mathcal{FP} \cup \mathcal{T})$–hereafter simply $\text{OMT}_{[\mathcal{FP}]}$–which is inspired by the OBV- BS algorithm for $\text{OMT}_{[\mathcal{BV}]}$ [32] and implements Definition 6 and Theorem 2. Here $\mathcal{T}$ may be empty, or contain $\mathcal{BV}$ and other theories (e.g. that of arrays). We assume that an $\text{SMT}(\mathcal{BV} \cup \mathcal{FP} \cup \mathcal{T})$-solving procedure is available—hereafter simply "SMT"—even when $\mathcal{BV}$ is not part of $\mathcal{T}$, because we need accessing explicitly to each bit in obj, which is not possible with plain $\mathcal{FP}$.

The optimization search tries to lexicographically maximize the (implicit) *attractor trajectory* vector $\mathcal{A}_\varphi$, which is incrementally derived from the current value of the dynamic attractor. The raw value of the dynamic attractor's bits drive the optimization search towards the direction of maximum gain at any given point in time, without disrupting any decision that has been already made. The dynamic attractor is incrementally updated along the search, based on the outcome of the previous rounds of the optimization search. At each round, one bit of the objective function is assigned its final value. The first round decides the sign, the next batch of rounds decides the exponent, and the remaining rounds decide the fine-grained details of the significand.

The pseudo-code of OFP- BS is shown in Fig. 2. The arguments of the algorithm are the input formula φ and the $\mathcal{FP}$ objective obj, where obj is a $\mathcal{FP}$ variable with $ebits$ bits in the exponent, $sbits - 1$ in the significand and $n \overset{\text{def}}{=} ebits + sbits$ bits overall.

The procedure starts by checking whether the input formula φ is satisfiable and immediately terminates if this is not the case (rows 1–3). If $\mathcal{M}(\text{obj}) = \text{NaN}$, then the procedure checks whether there exists a model $\mathcal{M}'$ for $\varphi \wedge \neg\mathsf{IsNaN}(\mathsf{obj})$ (rows 4–5). If this is not the case, the procedure terminates immediately and returns the pair $\langle \text{SAT}, \mathcal{M} \rangle$ (row 7). Otherwise, the model $\mathcal{M}$ is updated with the new model $\mathcal{M}'$ (row 9). In every case, φ is permanently extended with the constraint $\neg\mathsf{IsNaN}(\mathsf{obj})$ (row 10).

At this point, the procedure initializes the value of the dynamic attractor by invoking an external function UPDATE_DYNAMIC_ATTRACTOR() with the empty assignment τ as parameter, so that the returned value is equal to $-\infty$ when minimizing and $+\infty$ when maximizing

[7] Currently, there is no such specialized optimization procedure embedded within the *lazy* $\mathcal{FP}$-Solver of OptiMathSAT, so we won't describe this approach any further.

```
function OFP-BS ($\varphi$, obj)
 1: $\langle res, \mathcal{M} \rangle := \text{SMT.CHECK\_UNDER\_ASSUMPTIONS}(\varphi, \emptyset)$
 2: if ($res == \text{UNSAT}$) then
 3:     return $\langle res, \emptyset \rangle$                          // $\varphi$ is unsatisfiable
 4: if ($\mathcal{M}(\text{obj}) == \text{NAN}$) then
 5:     $\langle res, \mathcal{M}' \rangle := \text{SMT.CHECK\_UNDER\_ASSUMPTIONS}(\varphi \wedge \neg\text{IsNaN}(\text{obj}), \emptyset)$
 6:     if ($res == \text{UNSAT}$) then
 7:         return $\langle \text{SAT}, \mathcal{M} \rangle$                  // obj can only be NAN
 8:     else
 9:         $\mathcal{M} := \mathcal{M}'$
10: $\varphi := \varphi \wedge \neg\text{IsNaN}(\text{obj})$     // from now on, obj cannot be equal to NAN
11: $\tau := \emptyset$
12: $attr_\tau := \text{UPDATE\_DYNAMIC\_ATTRACTOR}(\tau, -1)$
13: for $i := 0$ up to $n - 1$ do
14:     $eq := (\text{obj}[i] = attr_\tau[i])$                  // attractor equality $A_\tau[i]$
15:     if ($\mathcal{M} \models eq$) then
16:         $\tau := \tau \cup \{eq\}$
17:     else
18:         $\langle res, \mathcal{M}' \rangle := \text{SMT.CHECK\_UNDER\_ASSUMPTIONS}(\varphi, \tau \cup \{eq\})$
19:         if ($res == \text{SAT}$) then
20:             $\tau := \tau \cup \{eq\}$
21:             $\mathcal{M} := \mathcal{M}'$
22:         else
23:             $\tau := \tau \cup \{\neg eq\}$
24:             $attr_\tau := \text{UPDATE\_DYNAMIC\_ATTRACTOR}(\tau, i)$
25: return $\langle \text{SAT}, \mathcal{M} \rangle$
```

Fig. 2 OFP- BS Algorithm for floating-point optimization

```
function UPDATE_DYNAMIC_ATTRACTOR ($\tau$, $i$)
 1: static $attr_\tau = -\infty$                                // track $-\infty$
 2: if ($\tau \neq \emptyset$ and $i \geq 0$) then
 3:     $attr_\tau[i] = \overline{attr_\tau[i]}$        // flip unfeasible value of current bit
 4:     if ($\tau[0] == 0$) then
 5:         for $j := i + 1$ up to $n - 1$ do
 6:             $attr_\tau[j] = 0$              // track smallest positive value
 7:     else
 8:         if ($i \leq ebits$) then
 9:             for $j := i + 1$ up to $n - 1$ do
10:                 $attr_\tau[j] = 1$          // track largest negative value
11· return $attr$
```

Fig. 3 The function UPDATE_DYNAMIC_ATTRACTOR()

(rows 11–12). Then, the execution moves to the section of code implementing the core part of the OFP- BS algorithm (rows 13–24), which consists of a loop over the bits of obj, starting from the MSB obj[0] down to the LSB obj[$n - 1$] (Fig. 3).

Inside this loop, OFP- BS first checks whether the value of obj[i] in $\mathcal{M}$ matches the i-th bit of the (current) dynamic attractor $attr_\tau$. If this is the case, then the i-th bit is already set to its "best" value in $\mathcal{M}$. Thus, the assignment τ is extended so as to permanently set obj[i] $= attr_\tau[i]$ (row 16), and the optimization search moves to the next iteration of the loop. If instead obj[i] $\neq attr_\tau[i]$ in $\mathcal{M}$, we need to verify whether the value of the objective function in $\mathcal{M}$ can be improved by forcing the i-th bit of obj equal to the i-th bit of the dynamic attractor. To do so, we incrementally invoke the underlying SMT solver, this time checking the satisfiability of φ under the list of assumptions $\tau \cup \{\text{obj}[i] = attr_\tau[i]\}$ (row 18). If the SMT solver returns SAT, then the value of the objective function has been successfully improved. Hence, τ is extended with an assignment setting obj[i] equal to $attr_\tau[i]$, and $\mathcal{M}$ is

$$\begin{array}{llll}
\tau_0 = \emptyset & attr_{\tau_0} = \texttt{(fp \#b1 \#b111 \#b0000)} = [\underline{1}.111.0000] & [-\infty] & \Rightarrow \textsc{sat} \\
\tau_1 = \tau_0 \cup \{\mathsf{obj}[0] = 1\} & attr_{\tau_1} = \texttt{(fp \#b1 \#b111 \#b0000)} = [1.\underline{1}11.0000] & [-\infty] & \Rightarrow \textsc{sat} \\
\tau_2 = \tau_1 \cup \{\mathsf{obj}[1] = 1\} & attr_{\tau_2} = \texttt{(fp \#b1 \#b111 \#b0000)} = [1.1\underline{1}1.0000] & [-\infty] & \Rightarrow \textsc{unsat} \\
\tau_3 = \tau_2 \cup \{\mathsf{obj}[2] = 0\} & attr_{\tau_3} = \texttt{(fp \#b1 \#b101 \#b1111)} = [1.10\underline{1}.1111] & [-31/4] & \Rightarrow \textsc{sat} \\
\tau_4 = \tau_3 \cup \{\mathsf{obj}[3] = 1\} & attr_{\tau_4} = \texttt{(fp \#b1 \#b101 \#b1111)} = [1.101.\underline{1}111] & [-31/4] & \Rightarrow \textsc{unsat} \\
\tau_5 = \tau_4 \cup \{\mathsf{obj}[4] = 0\} & attr_{\tau_5} = \texttt{(fp \#b1 \#b101 \#b0111)} = [1.101.0\underline{1}11] & [-23/4] & \Rightarrow \textsc{sat} \\
\tau_6 = \tau_5 \cup \{\mathsf{obj}[5] = 1\} & attr_{\tau_6} = \texttt{(fp \#b1 \#b101 \#b0111)} = [1.101.01\underline{1}1] & [-23/4] & \Rightarrow \textsc{unsat} \\
\tau_7 = \tau_6 \cup \{\mathsf{obj}[6] = 0\} & attr_{\tau_7} = \texttt{(fp \#b1 \#b101 \#b0101)} = [1.101.010\underline{1}] & [-21/4] & \Rightarrow \textsc{sat} \\
\tau_8 = \tau_7 \cup \{\mathsf{obj}[7] = 1\} & attr_{\tau_8} = \texttt{(fp \#b1 \#b101 \#b0101)} = [1.101.0101] & [-21/4] &
\end{array}$$

Fig. 4 An example of $\mathcal{FP}$ optimization using the dynamic attractor. ("$\Longrightarrow$ SAT/UNSAT" denotes the satisfiability of $\varphi_{\mathsf{noNaN}} \wedge \tau_k \wedge A_{\tau_k}[k]$. For ease of illustration, we have underlined the critical bit $attr_{\tau_k}[k]$ in the attractors and each attractor equality of the attractor trajectory $\dashv_\varphi$ inside the vectors of attractor equalities.)

replaced with the new model $\mathcal{M}'$ (rows 20–21). Otherwise, it is not possible to improve the objective function by toggling the value of obj$[i]$, and τ is extended so as to permanently set obj$[i] \neq attr_\tau[i]$ (row 23). At this point, there is a mismatch between the value of the first $i+1$ bits of obj in $\mathcal{M}$, corresponding to the assignment τ, and those of the current dynamic attractor. This mismatch is resolved by calling the function UPDATE_DYNAMIC_ATTRACTOR() with the updated assignment τ and the current loop iteration index i as parameters (row 24). In either case, the execution moves to the next iteration of loop.

After exactly n iterations of the loop, the optimization search terminates with the pair $\langle$SAT, $\mathcal{M}\rangle$, where $\mathcal{M}$ is the optimum model of the given OMT$_{[\mathcal{FP}]}$ instance. The OFP- BS algorithm requires at most $n+2$ incremental calls to an underlying SMT($\mathcal{FP}$) solver. The test in rows $15-16$ allows for saving lots of such SMT calls when the current model already assigns obj$[i]$ to its corresponding value in the attractor.

The function UPDATE_DYNAMIC_ATTRACTOR() takes as input τ, a (partial) assignment over the k most-significant bits of obj, and i, the index of of the current loop iteration in OFP- BS. When obj is minimized (The implementation is dual when obj is maximized), the procedure essentially works as follows. If $\tau = \emptyset$, then nothing is known about the solution of the problem, so $-\infty$ is returned. Otherwise, the procedure must compute the smallest $\mathcal{FP}$ value different from NAN (if any) which extends τ. In this case, the procedure starts by flipping the value of $attr_\tau[i]$, forcing obj$[i] = attr_\tau[i]$ (row 3). This ensures that the value of the first $i+1$ bits of obj in $\mathcal{M}$, corresponding to the assignment τ, is the same as the first $i+1$ bits of the current dynamic attractor. The remaining $n-i-1$ bits of $attr_\tau$ may also need to be updated to reflect this change. Since $\tau \neq \emptyset$ then we know that the sign of the objective function has been permanently decided in τ. If obj$[0] = 0$ in τ, i.e. obj must be positive, the procedure must return the smallest positive $\mathcal{FP}$ value admitted by τ. Hence, we update $attr_\tau$ with $\bigcup_{j=i+1}^{j=n-1} attr_\tau[j] = 0$ and return the corresponding $\mathcal{FP}$ value (rows 4–6). If obj$[0] = 1$ in τ, i.e. obj can be negative values, the procedure must return the largest negative $\mathcal{FP}$ value admitted by τ. When $i \leq ebits$ then at least one bit in the exponent of obj is assigned to 0 in τ (i.e. obj$[i]$). If that is the case, then we update $attr_\tau$ with $\bigcup_{j=i+1}^{j=n-1}$ obj$[j] = 1$ and return the corresponding $\mathcal{FP}$ value (rows 7–10). In practice, we notice that the block of code at rows 4–10 needs to be executed at most once because the decision of tracking the smallest positive value or the largest negative value (different from $-\infty$) is permanent.

Example 8 Let $\langle\varphi_{\mathsf{noNaN}}, \text{obj}\rangle$ be a restricted OMT$_{[\mathcal{FP}]}$ problem where obj is a $\mathcal{FP}$ objective, of sort `(_ FP 3 5)`, to be minimized. We consider the case in which the input formula φ_{noNaN} requires obj to be larger or equal $\frac{-21}{4}$ and it does not impose any other constraint on the value of obj. Given the sequence of (partial) assignments $\tau_0, \ldots, \tau_8$ in Fig. 4, it can be seen that after determining the unsatisfiability of obj[2] $= attr_{\tau_2}[2]$, the dynamic attractor must start tracking the *largest negative value different from* $-\infty$. Hence, the value of the last $n - i - 1$ bits of the dynamic attractor are set to be equal 1. Any subsequent call to UPDATE_DYNAMIC_ATTRACTOR() needs only to flip the value of $attr_\tau[i]$, because the last $n - i - 1$ bits of the dynamic attractor are already set to be equal 1. ◇

We stress the fact that, unlike with the $\mathcal{LA}$ [38,41] and $\mathcal{BV}$ [32] objective domains, OFP-BS does not simply perform binary search over the space of the values of the objective. Rather, after deciding the sign, it first performs binary search of the *exponent* values, which very-rapidly converges to the right order of magnitude, followed by binary search on the *significand* values, which fine-tunes the final result.

Example 9 To understand the range-pruning power of binary search over the exponent, consider the case of a 32-bit $\mathcal{FP}$ obj with 8-bit exponent and 23-bit significand. After assigning, e.g., the sign bit to 0 (positive value) the range of possible values is $[0^+, +\infty]$ ($[0^+, +3.4.10^{38}]$ if we exclude $+\infty$); assigning then the first exponent bit to 0, the range reduces to $[0^+, 2.0]$, reducing the range by more than a 10^{38} factor; by further setting the second exponent bit to 0, $[0^+, 1.1 \cdot 10^{-19}]$, further reducing the range by more than a 10^{19} factor, and so on. ◇

4.3 Search Enhancements

Given a $\mathcal{FP}$ value $attr$ and a $\mathcal{FP}$ goal obj, (a combination of) the following techniques can be used to adjust the behavior of the optimization search, similarly what has been proposed for the case of $\text{OMT}_{[\mathcal{BV}]}$ by Nadel et al. in [32].

- **branching preference**: the bits of the $\mathcal{FP}$ objective obj are marked, inside the OMT solver, as preferred variables for branching starting from the MSB down to the LSB. This ensures that conflicts involving the value of the objective function are handled as early as possible, possibly reducing the amount of work that needs to be redone after each back-jump.
- **polarity initialization**: the phase-saving value of each obj$[i]$ is initialized with the value of $attr[i]$. This encourages the OMT solver to assign the bits of obj so as to reassemble the bits of $attr$, thus possibly speeding-up the convergence towards the optimal value.

In the case of the basic OMT schema described in Sect. 4.1, the effectiveness of either technique depends on the initial choice for $attr$. In the lucky case, the value of $attr$ pulls the optimization search in the right direction and speeds up the search. In the unlucky case, when $attr$ pulls in the wrong direction, there is no visible effect or an overall slow down. For instance, in the case of the *linear-search* optimization schema, enabling both options with an unlucky choice of $attr$ can cause the OMT solver to start the search from the furthest possible point from the optional solution, and thus enumerate an exponential number of intermediate solutions. Naturally, the OMT-based optimization search algorithm is still guaranteed to terminate even in the worst-case scenario, but the unpredictable performance makes using either technique a generally unsuitable option in practice.

In the case of the OFP- BS algorithm described in Sect. 4.2, we use the latest value of the dynamic attractor $attr_\tau$ for both the *branching preference* (lines 11 and 18 of Fig. 2) and the *polarity initialization* (rows 12 and 19 of Fig. 2) techniques. We observe that the value of every bit in the dynamic attractor can change after the sign of the objective function has been decided. Furthermore, the value of all the significand's bits in the dynamic attractor can also change during the process of determining the optimal exponent value of the objective function (see, e.g., Example 5). As a consequence, if the OMT solver applies either enhancement before the correct improving direction is known, this may cause the underlying OMT engine to advance the search starting from a sub-optimal set of initial decisions. Enabling both enhancements at the same time could make things even worse. In order to mitigate this issue, we have designed a variant of our optimization-search approach which does not apply either enhancement on those bits of the objective function for which the best improving direction is not yet known. We have called this variant **safe bits restriction**.

5 Experimental Evaluation

We have implemented the procedures described in the previous sections on top of the OptiMathSAT OMT solver (v. 1.6.2), and assessed its performance on a set of OMT$_{[\mathcal{FP}]}$ formulas that have been automatically generated using the SMT($\mathcal{FP}$) benchmark-set of [4]. The formulas, the results and the scripts necessary to reproduce these results are made publicly available and can be downloaded from [1,2]. The experiments have been performed on an *i7-6500U 2.50GHz Intel Quad-Core* machine with 16GB of ram and running *Ubuntu Linux* 17.10. For each job pair we used a timeout of 600 seconds.

Experiment Setup. The OMT$_{[\mathcal{FP}]}$ instances used in this experiment have been automatically generated starting from the satisfiable formulas included in the SMT($\mathcal{FP}$) benchmark-set of [4]. We did not consider any of the unsatisfiable instances that are present in the remote repository. For each of the original SMT($\mathcal{FP}$) formulas we applied the following transformations. First, we either relaxed or removed some of the constraints in the original problem, so as to broaden the set of feasible solutions. This step is necessary because the majority of the original SMT($\mathcal{FP}$) formulas admits only one solution. Second, for each $\mathcal{FP}$ variable v appearing inside a SMT($\mathcal{FP}$) problem we generated a pair of OMT$_{[\mathcal{FP}]}$ instances, one for the minimization and another for the maximization of v. At the end of this step, we obtained 39536 OMT$_{[\mathcal{FP}]}$ formulas. Third, we randomly selected up to 300 OMT$_{[\mathcal{FP}]}$ instances from each of the five groups of problems in the OMT$_{[\mathcal{FP}]}$ benchmark-set. This filtering step yielded a total of 1120 SMT- LIBv2 formulas.

The first two OMT-based baseline implementations we have considered are OptiMathSAT(omt+lin) and OptiMathSAT(omt+bin), that run the linear- and the binary-search respectively. These configurations have been tested using both the *eager* and the *lazy* $\mathcal{FP}$ approaches. The third baseline implementation we have considered, named OptiMathSAT(eager+obv- bs), is based on a reduction of the OMT$_{[\mathcal{FP}]}$ problem to OMT$_{[\mathcal{BV}]}$ and it uses OptiMathSAT's implementation of the obv- bs engine presented by Nadel et al. [32].[8] For this test, we have generated an OMT$_{[\mathcal{BV}]}$ benchmark-set using a $\mathcal{BV}$ encoding that mimics the essential aspects of the ofp- bs algorithm described Sect. 4.2. We compared these baseline approaches with a configuration using the ofp- bs algorithm and the *eager* $\mathcal{FP}$ approach, namely OptiMathSAT(eager+ofp- bs). We have separately tested the effect of enabling the *branching preference* (bp), the *polarity initialization* (pi) and the *safe bits restriction* (so) enhancements described in Sect. 3.2, whenever these options were supported by the given configuration. We have not included other tools in our experiment because we are not aware of any other OMT$_{[\mathcal{FP}]}$ solver.

Last, in order to assess the significance of the optimization problems used in this experiment, we have collected the run-time statistics of OptiMathSAT on the SMT formulas

[8] Notice that the binaries of the original OMT$_{[\mathcal{BV}]}$ tools presented in [32] are not publicly available.

Table 2 (Top) Comparison among various OptiMathSAT (here simply "OM") configurations on the OMT$_{[\mathcal{FP}]}$ benchmark-set

tool, configuration and encoding	inst.	term.	t.o.	u	bt	st	time (s)
OM (EAGER+OMT+LIN)	1120	1003	117	0	5	73	76,375
OM (EAGER+OMT+LIN+PI)	1120	1003	117	0	5	71	76,785
OM (EAGER+OMT+LIN+BP)	1120	956	164	0	6	105	77,480
OM (EAGER+OMT+LIN+BP+PI)	1120	873	247	0	77	217	54,859
OM (LAZY+OMT+LIN)	1120	868	252	0	93	203	29,832
OM (EAGER+OMT+BIN)	1120	1014	106	0	11	281	67,834
OM (EAGER+OMT+BIN+PI)	1120	970	150	0	8	285	69,765
OM (EAGER+OMT+BIN+BP)	1120	1016	104	0	14	205	68,255
OM (EAGER+OMT+BIN+BP+PI)	1120	991	129	0	65	**321**	56,941
OM (LAZY+OMT+BIN)	1120	900	220	0	90	243	33,260
OM (EAGER+OBVBS) [REDUCTION]	1120	1013	107	0	14	141	65,954
OM (EAGER+OFPBS)	1120	1017	103	0	9	171	70,732
OM (EAGER+OFPBS+PI)	1120	**1019**	101	0	34	280	64,896
OM (EAGER+OFPBS+PI+SO)	1120	1018	102	0	7	179	71,430
OM (EAGER+OFPBS+BP)	1120	975	145	0	2	145	65,543
OM (EAGER+OFPBS+BP+SO)	1120	1000	120	0	3	124	68,390
OM (EAGER+OFPBS+BP+PI)	1120	1001	119	0	77	273	60,365
OM (EAGER+OFPBS+BP+PI+SO)	1120	1006	114	**19**	32	245	59,463
VIRTUAL BEST	1120	**1074**	46	–	559	1074	27,788
OM (EAGER+SMT) [NO OPTIMIZATION]	1120	1048	72	–	–	–	9259

The columns list the total number of instances (inst.), the number of instances solved (term.), the number of timeouts (t.o.), the number of instances uniquely solved by the given configuration (u), the number of instances solved faster than any other configuration (bt), the total number of instances solved with the best time (st) and the total solving time for all solved instances (time)

obtained by stripping the objective function from each OMT instance, so that no optimization is to be performed. We named this configuration OptiMathSAT(EAGER+SMT).

For all problem instances, we verified the correctness of the optimal solution found by each configuration with an SMT solver (MathSAT5). When terminating, all tools returned the same optimum value.

*Experiment Results.*The results of this experiment are listed in Table 2: Fig. 5 depicts the log-scale cactus plot of the same data, for a visual comparison among the different configurations; in addition, Figs. 6, 7 and 8 show a selection of relevant pairwise comparisons among various OptiMathSAT configurations, focusing on variants of the OMT-based linear-search approach, of the OMT-based binary-search approach, and of the OFP- BS approach respectively.

Concerning OMT-based *linear-search* optimization, we observe that OptiMathSAT performs the best when no enhancement is enabled. In particular, the empirical evidence suggests that enabling *branching preference* significantly increases the number of timeouts, generally deteriorating the performance (plot 1A in Fig. 6). Enabling only *polarity initialization* does not result in an appreciable change on the running time of the solver (plot 1B in Fig. 6). In contrast, enabling both enhancements at the same time has a small chance to result in a small improvement of the search time (plot 2A in Fig. 6), but it generally worsens the performance

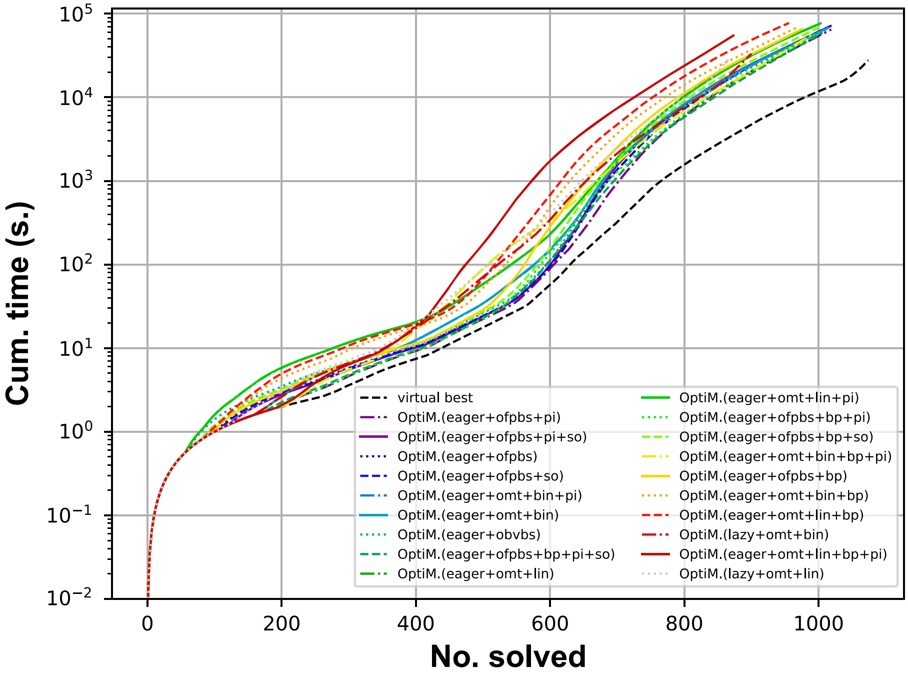

Fig. 5 Cactus plots of the data displayed in Table 2

and results in a drastic increase in the number of timeouts (Table 2). We justify these results as follows. First, when only *polarity initialization* is used, the phase-saving value that is being set by OPTIMATHSAT does not really matter because the optimization search is dominated by the structure of the formula itself rather than by the bits of the $\mathcal{FP}$ objective. Second, when *polarity initialization* is used on top of *branching preference*, there is an even more drastic decrease in performance due to the fact that the initial phase-saving value that is statically assigned by the OMT solver to the bits of the $\mathcal{FP}$ objective cannot be expected to be "good enough" for any situation. In fact, as illustrated in example 5, the initial phase-saving can be misleading and force the OMT solver—when running in *linear-search*—to explore an exponential number of intermediate satisfiable solutions.

In the case of the OMT-based *binary-search* optimization approach, we observe that it solves more formulas than linear-search and it generally appears to be faster (plot 3B in Fig. 6). Overall, *polarity initialization* does not seem to be beneficial, whereas enabling *branching preference* increases the number of formulas solved within the timeout. This behavior is different from the linear-search approach, and we conjecture that it is due to the fact that, with the OMT-based binary-search approach, branching over the bits of the objective function can reveal in advance any (partial) assignment to the bits of the objective function that it is inconsistent wrt. the pivoting cuts learned by the optimization engine.

Using the *lazy* $\mathcal{FP}$ engine results in fewer formulas being solved, although a significant number of these benchmarks is solved faster than with any other configuration (over 90 instances, for both configurations).

The OPTIMATHSAT(EAGER+OBV- BS) configuration is able to solve 1013 formulas within the timeout, showing that OMT$_{[\mathcal{FP}]}$ can be reduced to OMT$_{[\mathcal{BV}]}$ effectively, and that—on

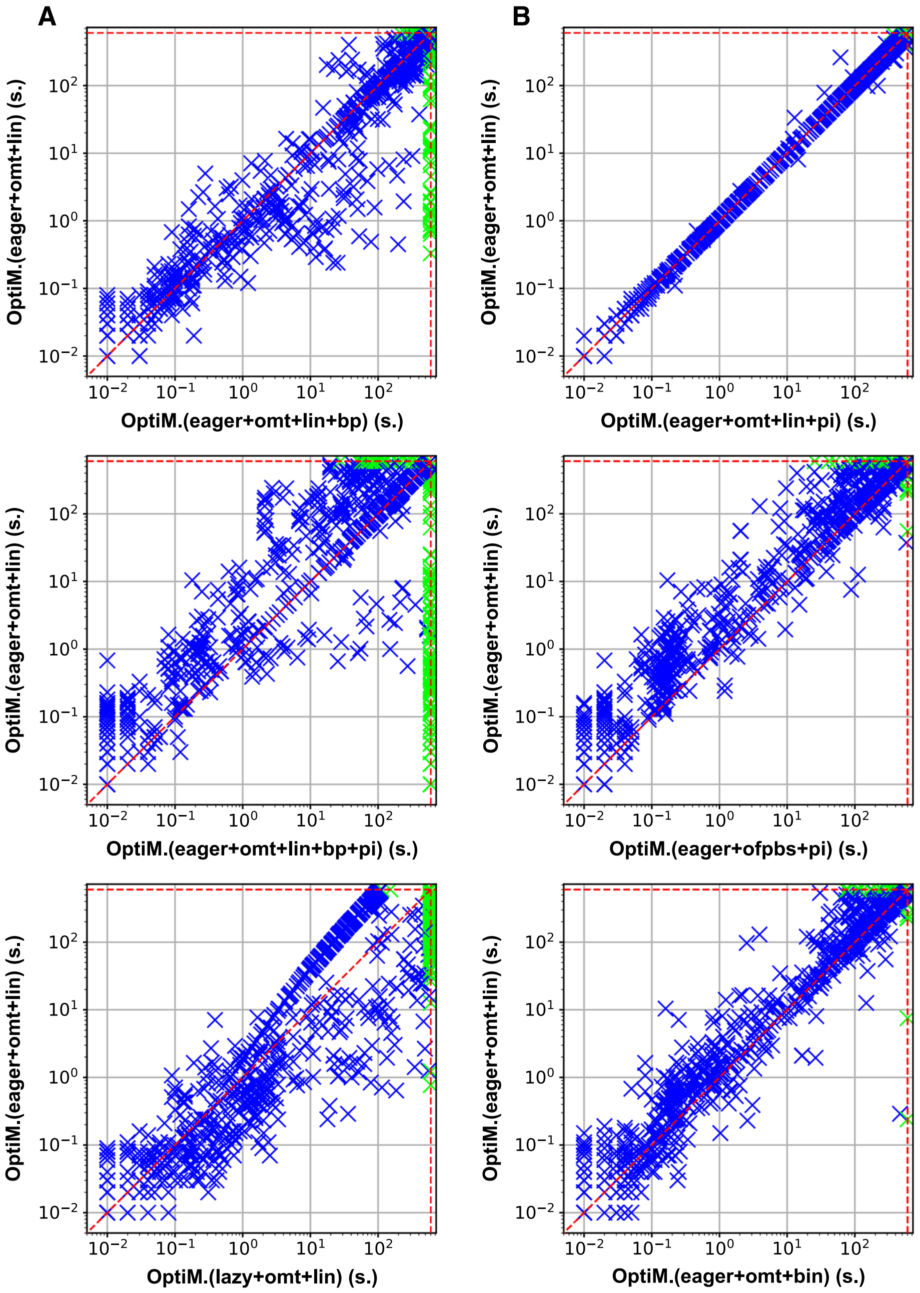

Fig. 6 Pairwise comparisons on $\mathrm{OMT}_{[\mathcal{FP}]}$ formulas using OMT-based linear-search and other configurations. (Blue points denote satisfiable benchmarks, green denotes a timeout.) (Color figure online)

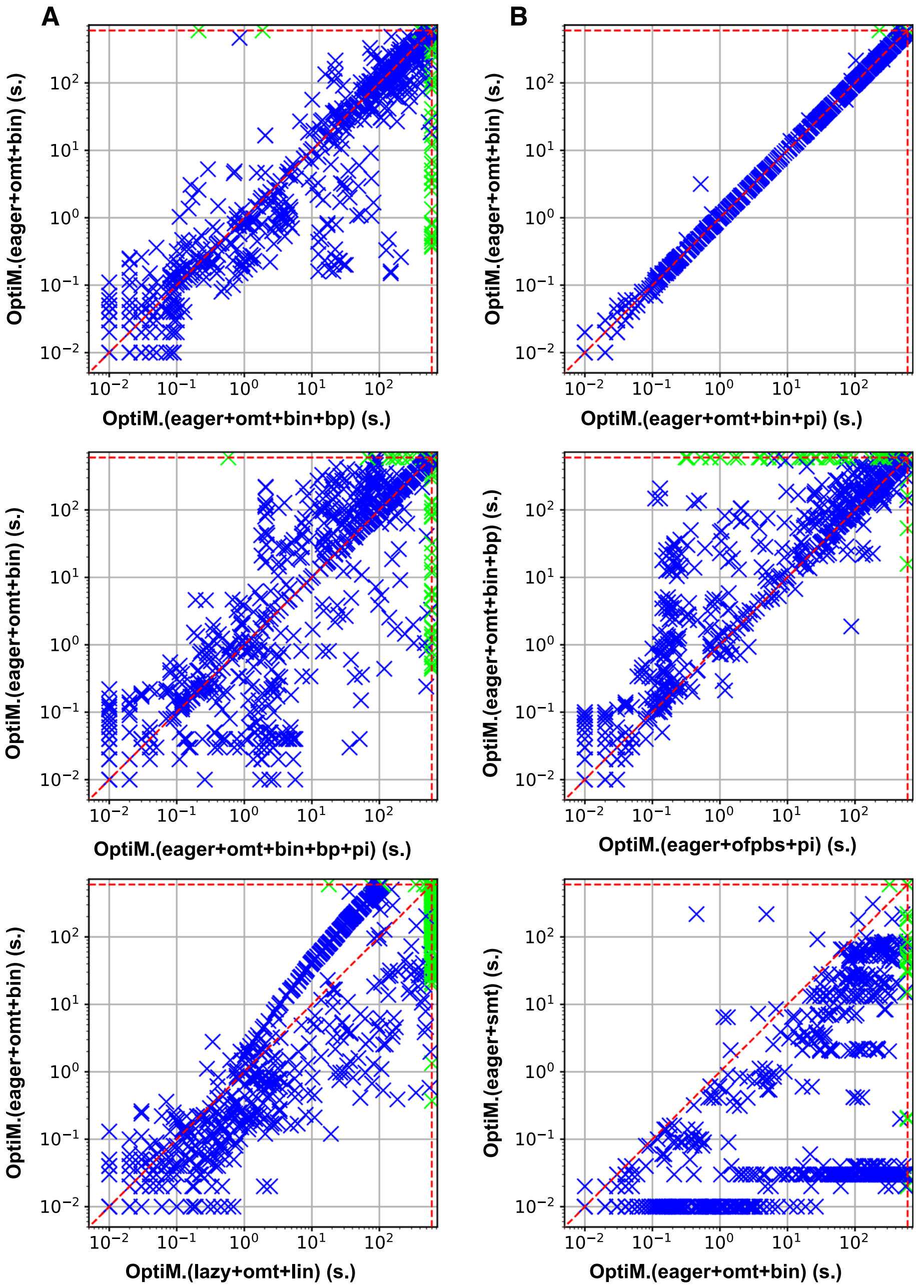

Fig. 7 Pairwise comparisons on OMT$_{[\mathcal{FP}]}$ formulas using OMT-based binary-search and other configurations. (Blue points denote satisfiable benchmarks, green denotes a timeout.)

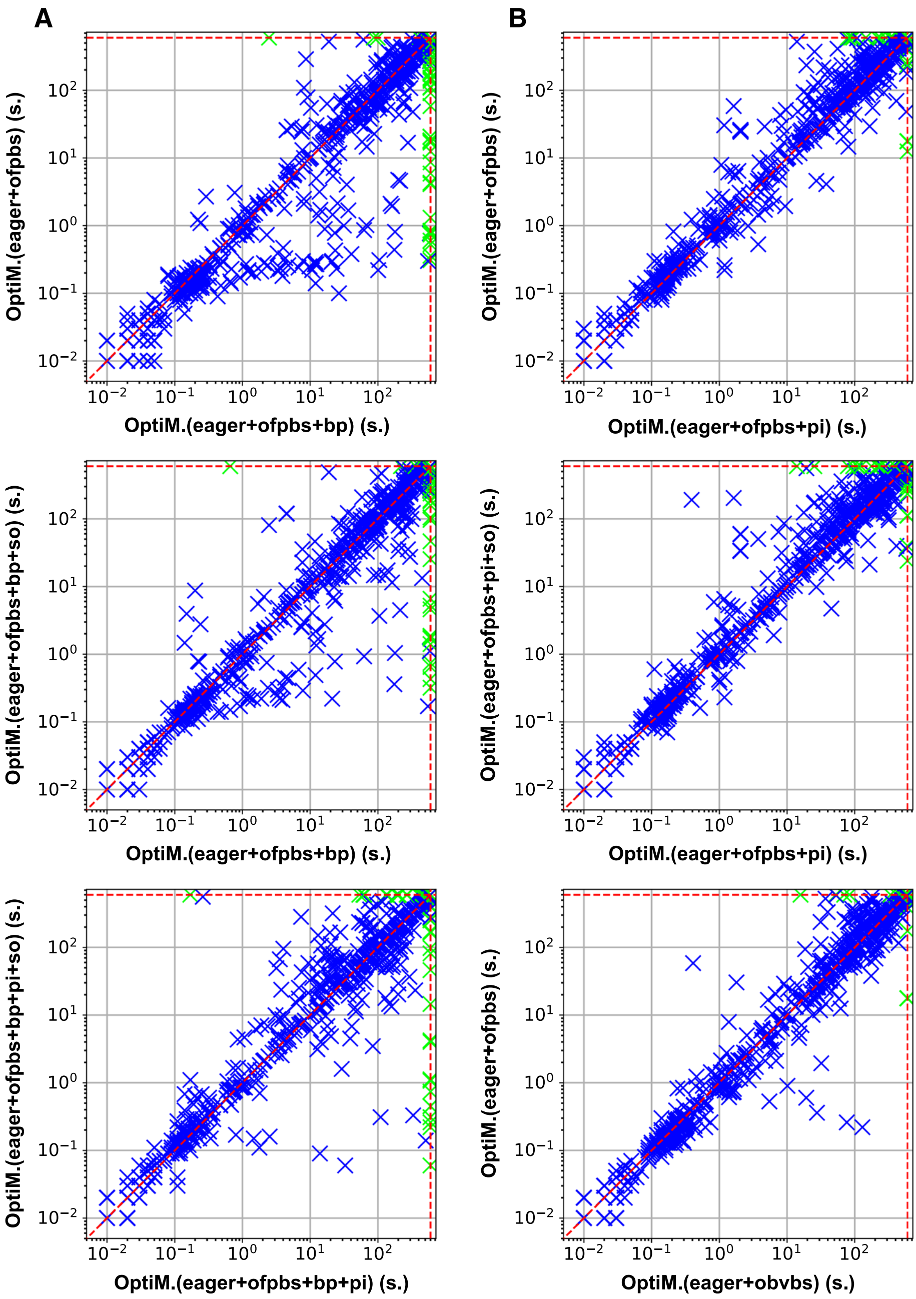

Fig. 8 Pairwise comparisons on OMT$_{[\mathcal{FP}]}$ formulas using the OFP- BS engine and other configurations. (Blue points denote satisfiable benchmarks, green denotes a timeout.) (Color figure online)

the given benchmark-set—the performance of this approach are comparable with the best $\text{OMT}_{[\mathcal{FP}]}$ configurations being tested.

Overall, the best performance is obtained by using the OFP- BS engine, with up to 1019 benchmark-set instances solved in correspondence to the OPTIMATHSAT(EAGER+OFP-BS+PI) configuration. In plot $2B$ of Figs. 6 and 7, we show the pairwise comparison of the best OFP- BS configuration with the best OMT-based run. Similarly to the case of OMT-based optimization with linear-search, we observe that enabling *branching preference* generally makes the performance worse (plot $1A$ in Fig. 8). Instead, when *polarity initialization* is used we observe a general performance improvement that does not only result in an increase in the number of formulas being solved within the timeout, but also a noticeable reduction of the solving time as a whole. This is in contrast with the case of OMT-based optimization, and it can be explained by the fact that OFP- BS uses an internal heuristic function to dynamically determine and update the most appropriate phase-saving value for the bits of the objective function. An equally important role is played by the *safe bits restriction*, that limits the effects of *branching preference* and *polarity initialization* to only certain bits of the *dynamic attractor*. As illustrated by the plots in the second and third rows of Fig. 8 and by the data in Table 2, tThis feature is particularly effective when used in combination with *branching preference*.

The results of OPTIMATHSAT over the SMT-only version of the benchmark-set (no optimization) are reported in the last row of Table 2 and in the scatter-plot $3B$ in Fig. 7, and show that for a large number of instances the OMT problem is considerably harder than its SMT-only version. There are a few exceptions to this rule, that we ascribe to the fact that the removal of the objective function alters the internal stack of formulas, and this can have unpredictable consequences on the behavior of various internal heuristics that depend on it. A solution can be found in a shorter amount of time when the sequence of (heuristic) choices is compatible with its assignment and it requires little back-tracking effort.

6 Conclusions and Future Work

We have presented for the first time OMT procedures for (signed bit-vectors and) floating-point objectives, based on the novel notions of attractor and dynamic attractor, which we have implemented in OPTIMATHSAT and tested on modified problems from SMT-LIB.

Ongoing research involves implementing our OFP- BS procedure on top of the ACDCL SMT($\mathcal{FP}$) procedure—which is not immediate to do efficiently because the latter approach does not allow directly accessing and setting the single bits of the objective (since $\mathcal{BV}$ and $\mathcal{FP}$ are not signature-disjoint). Future research involves experimenting the new OMT procedure directly on problems coming from bit-precise SW and HW verification, produced, e.g., by the NuXmv model checker [3].

Funding Open access funding provided by Università degli Studi di Trento within the CRUI-CARE Agreement.

References

1. http://disi.unitn.it/trentin/resources/floatingpoint_test.tar.gz
2. https://hub.docker.com/repository/docker/patricktrentin88/jar2020_floatingpoint_test
3. NUXMV. https://nuxmv.fbk.eu
4. SmtLibv2. www.smtlib.cs.uiowa.edu/
5. IEEE standard 754, 2008. http://grouper.ieee.org/groups/754/
6. Albuquerque, H. F., Araujo, R. F., de Bessa, I. V., Cordeiro, L. C., de Lima Filho, E. B.: OptCE: A Counterexample-Guided Inductive Optimization Solver. In *SBMF*, volume 10623 of *Lecture Notes in Computer Science*, pages 125–141. Springer, 2017
7. Araujo, R.F., Albuquerque, H.F., de Bessa, I.V., Cordeiro, L.C., Filho, J.E.C.: Counterexample guided inductive optimization based on satisfiability modulo theories. Sci. Comput. Program. **165**, 3–23 (2018)
8. Araújo, R., Bessa, I., Cordeiro, L. C., Filho, J. E. C.: SMT-based Verification Applied to Non-convex Optimization Problems. In *2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC)*, pages 1–8, Nov 2016
9. Bjorner, N., Phan, A.-D.: νZ - Maximal Satisfaction with Z3. In *Proc International Symposium on Symbolic Computation in Software Science*, Gammart, Tunisia, December 2014. EasyChair Proceedings in Computing (EPiC)
10. Bjorner, N., Phan, A.-D, Fleckenstein, L.: νZ - An Optimizing SMT Solver.In *Proc. TACAS*, volume 9035 of *LNCS*. Springer, 2015
11. Bozzano, M., Bruttomesso, R., Cimatti, A., Franzèn, A., Hanna, Z., Khasidashvili, Z., Palti, A. Sebastiani, R.: Encoding RTL Constructs for MathSAT: a Preliminary Report. In *Proc. 3rd Workshop of Pragmatics on Decision Procedure in Automated Reasoning, PDPAR'05*, ENTCS. Elsevier, 2005
12. Brain, M., D'Silva, V., Griggio, A., Haller, L., Kroening, D:. Interpolation-Based Verification of Floating-Point Programs with Abstract CDCL. In *SAS*, pages 412–432, 2013
13. Brain, M., D'Silva, V., Griggio, A., Haller, L., Kroening, D.: Deciding floating-point logic with abstract conflict driven clause learning. Formal Methods in System Design **45**(2), 213–245 (2014)
14. Brain, M., Tinelli, C., Rümmer, P., Wahl, T:. An Automatable Formal Semantics for IEEE-754 Floating-Point Arithmetic. In *ARITH*, pages 160–167. IEEE, 2015
15. Brillout, A., Kroening, D., Wahl, T.: Mixed abstractions for floating-point arithmetic. In *2009 Formal Methods in Computer-Aided Design*, pages 69–76, Nov 2009
16. Brinkmann, R., Drechsler, R.: RTL-datapath verification using integer linear programming. In *Proc. ASP-DAC 2002*, pages 741–746. IEEE, 2002
17. Brummayer, R.: *Efficient SMT Solving for Bit-Vectors and the Extensional Theory of Arrays*. PhD thesis, Informatik, Johannes Kepler University Linz, 2009
18. Brummayer, R., Biere, A.: Boolector: An efficient smt solver for bit-vectors and arrays. In *TACAS*, pages 174–177, Berlin, Heidelberg, 2009. Springer-Verlag
19. Bruttomesso, R., Cimatti, A., Franzén, A., Griggio, A., Hanna, Z., Nadel, A., Palti, A., Sebastiani, R.: A Lazy and Layered SMT($\mathcal{BV}$) Solver for Hard Industrial Verification Problems. In *CAV*, volume 4590 of *LNCS*, pages 547–560. Springer, 2007
20. Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R., Stenico, C.: Satisfiability modulo the theory of costs: Foundations and applications. In *TACAS*, volume 6015 of *LNCS*, pages 99–113. Springer, 2010
21. Cimatti, A., Griggio, A., Schaafsma, B. J., Sebastiani, R.: A Modular Approach to MaxSAT Modulo Theories. In *International Conference on Theory and Applications of Satisfiability Testing, SAT*, volume 7962 of *LNCS*, July 2013
22. Dillig, I., Dillig, T., McMillan, K. L., Aiken, A.: Minimum Satisfying Assignments for SMT. In *CAV*, pages 394–409, 2012
23. Fazekas, K., Bacchus, F., Biere, A.: Implicit Hitting Set Algorithms for Maximum Satisfiability Modulo Theories. In *IJCAR*, volume 10900 of *Lecture Notes in Computer Science*, pages 134–151. Springer, 2018
24. Ganesh, V., Dill, D. L:. A Decision Procedure for Bit-Vectors and Arrays. In *CAV*, 2007
25. Hadarean, L.: *An Efficient and Trustworthy Theory Solver for Bit-vectors in Satisfiability Modulo Theories*. PhD thesis, New York University, 2015
26. Hadarean, L., Bansal, K., Jovanovic, D., Barrett, C., Tinelli, C.: A Tale of Two Solvers: Eager and Lazy Approaches to Bit-Vectors. In *CAV*, volume 8559 of *Lecture Notes in Computer Science*, pages 680–695. Springer, 2014
27. Haller, L., Griggio, A., Brain, M., Kroening, D.: Deciding Floating-Point Logic with Systematic Abstraction. In *Proc. of FMCAD*, 2012
28. Kovásznai, G., Biró, C., Erdélyi, B.: Puli - a problem-specific omt solver. EasyChair Preprint no. 371, EasyChair, 2018

29. Larraz, D., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: Minimal-Model-Guided Approaches to Solving Polynomial Constraints and Extensions. In *SAT*, 2014
30. Li, Y., Albarghouthi, A., Kincad, Z., Gurfinkel, A., Chechik, M.: Symbolic Optimization with SMT Solvers. In *POPL*, 2014
31. Manolios, P., Papavasileiou, V.: Ilp modulo theories. In *CAV*, pages 662–677, 2013
32. Nadel, A., Ryvchin, V.: Bit-Vector Optimization. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2016*, volume 9636 of *LNCS*. Springer, 2016
33. Niemetz, A.: *Bit-Precise Reasoning Beyond Bit-Blasting*. PhD thesis, Informatik, Johannes Kepler University Linz, 2017
34. Niemetz, A., Preiner, M., Fröhlich, A., Biere, A.: Improving Local Search For Bit-Vector Logics in SMT with Path Propagation. In *Proc. 4th Intl. Work. on Design and Implementation of Formal Tools and Systems (DIFTS'15)*, page 10 pages, 2015
35. Nieuwenhuis, R., Oliveras, A.: On SAT Modulo Theories and Optimization Problems. In *Proc. Theory and Applications of Satisfiability Testing - SAT 2006*, volume 4121 of *LNCS*. Springer, 2006
36. Roc, O.: Optimization Modulo Theories. Master's thesis, Polytechnic University of Catalonia, 2011. http://hdl.handle.net/2099.1/14204
37. Ruemmer, P., Wahl, T.: An SMT-LIB Theory of Binary Floating-Point Arithmetic. SMT 2010 Workshop, July 2010. Available at http://www.philipp.ruemmer.org/publications/smt-fpa.pdf
38. Sebastiani, R., Tomasi, S.: Optimization in SMT with LA(Q) Cost Functions. In *IJCAR*, volume 7364 of *LNAI*, pages 484–498. Springer, July 2012
39. Sebastiani, R., Tomasi, S.: Optimization Modulo Theories with Linear Rational Costs. ACM Transactions on Computational Logics **16**(2),(March 2015)
40. Sebastiani, R., Trentin, P.: OptiMathSAT: A Tool for Optimization Modulo Theories. In *Proc. International Conference on Computer-Aided Verification, CAV 2015*, volume 9206 of *LNCS*. Springer, 2015
41. Sebastiani, R., Trentin, P.: Pushing the Envelope of Optimization Modulo Theories with Linear-Arithmetic Cost Functions. In *Proc. Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'15*, volume 9035 of *LNCS*. Springer, 2015
42. Sebastiani, R., Trentin, P.: On Optimization Modulo Theories, MaxSMT and Sorting Networks. In *Proc. Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'17*, volume 10205 of *LNCS*. Springer, 2017
43. Sebastiani, R., Trentin, P.: OptiMathSAT: A Tool for Optimization Modulo Theories. *Journal of Automated Reasoning*, Dec 2018
44. Trentin, P., Sebastiani, R.: Optimization Modulo the Theory of Floating-Point Numbers. In *In proc. 27th International Conference on Automated Deduction - CADE-27.*, LNCS, pages 550–567. Springer, 2019
45. Zeljić, A., Backeman, P., Wintersteiger, C.M., Rümmer, P.: Exploring approximations for floating-point arithmetic using uppsat. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) Automated Reasoning. pp, pp. 246–262. Springer International Publishing, Cham (2018)
46. Zeljić, A., Wintersteiger, C.M., Rümmer, P.: Approximations for model construction. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) Automated Reasoning. pp, pp. 344–359. Springer International Publishing, Cham (2014)
47. Zeljić, A., Wintersteiger, C.M., Rümmer, P.: An approximation framework for solvers and decision procedures. Journal of Automated Reasoning **58**(1), 127–147 (2017)