Optimization Modulo the Theory of Floating-Point Numbers *

Patrick Trentin and Roberto Sebastiani

DISI, University of Trento, Italy

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) $\mathcal{BV}$ has been proposed by Nadel & Ryvchin, unfortunately we are not aware of any existing approach for OMT with $\mathcal{FP}$.
In this paper we fill this gap. We present a novel OMT approach, based on the novel concept of *attractor* and *dynamic attractor*, which extends the work of Nadel & Ryvchin to signed $\mathcal{BV}$ and, most importantly, to $\mathcal{FP}$. We have implemented some OMT($\mathcal{BV}$) and OMT($\mathcal{FP}$) procedures on top of OPTIMATHSAT and tested the latter ones on modified problems from the SMT-LIB repository. The empirical results support the validity and feasibility of the novel approach.

1 Introduction

Optimization Modulo Theories (OMT) [23, 15, 20, 5, 19, 25, 26, 21, 27, 16] 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.

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. (For instance, during the verification process of a piece of software, one may look for the minimum/maximum value of some `int` [resp. `float`] parameter causing an SMT($\mathcal{BV}$) [resp. SMT($\mathcal{FP}$)] call to return SAT—which typically corresponds to the presence of some bug— so that to guarantee a safe range for such parameter.)

OMT for the theory of (unsigned) bit-vectors (OMT($\mathcal{BV}$)) was proposed by Nadel and Ryvchin [21], although a reduction to the problem to MaxSAT was already implemented in the SMT/OMT solver Z3 [6]. The work in [21] was based on the observation that OMT on unsigned $\mathcal{BV}$ can be seen as lexicographic optimization over the bits in

*We would like to thank the anonymous reviewers for their insightful comments and suggestions, and we thank Alberto Griggio for support with MATHSAT5 code.

the bitwise representation of the objective, ordered from the most-significant bit (MSB) to the least-significant bit (LSB).

In this paper we address —for the first time to the best of our knowledge— OMT for the theory of signed Bit-Vectors and, most importantly, for the theory of Floating-Point Arithmetic (OMT($\mathcal{FP}$)), 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 start from introducing the notion of *attractor*, which represent (the bitwise encoding of) the target value for the objective which the optimization process aims at. This allows us for easily leverage the procedure of [21] 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 moves 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 at the opposite 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($\mathcal{FP}$) procedures, which require at most $n+2$, incremental calls to an SMT($\mathcal{FP}$) solver, n being the number of bits in the representation of the objective. Notice that these procedures do not depend on the underlying SMT($\mathcal{FP}$) procedure used, provided the latter allows for accessing and setting the single bits of the objective.

We have implemented these OMT($\mathcal{BV}$) and OMT($\mathcal{FP}$) procedures on top of the OPTIMATHSAT OMT solver [27]. We have run an experimental evaluation of the OMT($\mathcal{FP}$) procedures on modified SMT($\mathcal{FP}$) 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 §2 we provide the necessary background on $\mathcal{BV}$ and $\mathcal{FP}$ theories and reasoning. In §3 we provide the novel theoretical definitions and results. In §4 we describe our novel OMT($\mathcal{FP}$) procedures. In §5 we present the empirical evaluation. In §6 we conclude, hinting some future directions. The proofs of the theoretical results from §3 are in the extended version of this paper [**?**].

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).[1] A $\mathcal{BV}$ constant of width n is an interpreted vector of n values in $\{0,1\}$. We $\overline{overline}$ 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 [3], we may also represent a $\mathcal{BV}$ constant in *binary* (e.g. $\mathbf{28}^{[8]}$ is written $\#b00011100$) or in *hexadecimal* (e.g. $\mathbf{28}^{[8]}$ is written $\#x1C$) form. A $\mathcal{BV}$ term is built from $\mathcal{BV}$ constants, variables and interpreted $\mathcal{BV}$ functions which represent standard 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 like $\geq_n$, $<_n$ (e.g. $\mathbf{0}^{[8]} \geq_8 \mathbf{x}^{[8]}$) and equality. We refer the reader to [3] for further details on the syntax and semantics of Bit-Vector theory.

There are two main techniques for $\mathcal{BV}$ satisfiability, the "*eager*" and the "*lazy*" approach, which are substantially complementary to one another [18]. In the *eager* approach, $\mathcal{BV}$ terms and constraints are encoded into SAT via bit-blasting [17, 13, 22]. 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 sub-portion of the $\mathcal{BV}$ theory [12, 7, 14].

Floating-Point. The theory of *Floating-Point Numbers* ($\mathcal{FP}$), [3, 24, 10], is based on the IEEE standard 754-2008 [4] 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 $n \stackrel{\text{def}}{=} ebits + sbits$ 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. $\mathbf{sign}$ is a $\mathcal{BV}$ of size 1, $\mathbf{exp}$ is a $\mathcal{BV}$ of size $ebits$ and $\mathbf{sig}$ 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)`

[1] 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.

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) [4]. 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, while 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.

Example 1. Let x be the normal $\mathcal{FP}$ constant `(_ FP #b0 #b1100 #b01010000)`, and y be the subnormal $\mathcal{FP}$ constant `(_ FP #b0 #b0000 #b01010000)`, so that their corresponding sort is `(_ FP <4> <8>)`. Then, according to the semantics defined in the IEEE standard 754-2008 [4], the floating-point value of x and y in decimal notation is given by:

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

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

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 [4].

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 [11, 29, 28]. 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 [8, 9]. 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 present our generalization of [21] to the case of signed/unsigned Bit-Vector Optimization, and then move on to deal with Floating-Point Optimization.

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 $\mathsf{obj}[i]$ to denote the i-th bit of obj, where $\mathsf{obj}[0]$ and $\mathsf{obj}[n-1]$ are the Most Significant Bit (MSB) and the Least Significant Bit (LSB) of obj respectively.[1]

Definition 1. *(OMT($\mathcal{BV}$)). Let φ be a SMT($\mathcal{BV}$) formula and obj be a –signed or unsigned– $\mathcal{BV}$ variable occurring in φ. We call an* **Optimization Modulo $\mathcal{BV}$ problem, OMT($\mathcal{BV}$)**, *the problem of finding a model $\mathcal{M}$ for φ (if any) whose value of obj, denoted with $\mathsf{min}_{\mathsf{obj}}(\varphi)$, is 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)*

Hereafter, we generalize the unsigned $\mathcal{BV}$ maximization procedures described in [21] 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. $A[k] \stackrel{def}{=} (\mathsf{obj}[k] = attr[k])$, $k \in [0..n-1]$.*

Example 2. If $\mathsf{obj}^{[8]}$ is an *unsigned $\mathcal{BV}$* objective of width 8, then its corresponding attractor $attr$ is $\mathbf{0}^{[8]}$, i.e. $[00000000]$, when $\mathsf{obj}^{[8]}$ is minimized and it is $\mathbf{255}^{[8]}$, i.e. $[11111111]$, when $\mathsf{obj}^{[8]}$ is maximized. When $\mathsf{obj}^{[8]}$ 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. $\diamond$

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 $\mathcal{M}(\mathsf{obj}) = attr$.

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 all and only 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 $\mathsf{obj}[i]$ by μ_k [resp. τ_k]. Moreover, we use the expression $[\![\mu_k]\!]_i$ where $i \leq k$ to denote the restriction of μ_k to all and only the i most-significant bits of obj, $\mathsf{obj}[0], ..., \mathsf{obj}[i-1]$. 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 $A[i] \stackrel{\text{def}}{=} (\mathsf{obj}[i] = attr[i])$ to denote the single-bit assignment $\mathsf{obj}[i] := attr[i]$ and its negation $\neg A[i]$ to denote the assignment to the complement value $\mathsf{obj}[i] := \overline{attr[i]}$.

Definition 3. *(lexicographic maximization) Consider an OMT instance $\langle \varphi, \mathsf{obj} \rangle$ 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 $(\mathsf{obj}[k] = attr[k])$*. 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, τ_n [resp. $\mathcal{M}$] in Definition 3 assigns to each $\mathsf{obj}[k]$ the value $attr[k]$ unless it is inconsistent wrt. φ and the assignments to the previous $\mathsf{obj}[i]$s, $i \in [0..k-1]$. Notice that this corresponds to minimize [resp. maximize] the value $\sum_{k=0}^{n-1} 2^{n-1-k} \cdot (\mathsf{obj}[k] \mathbf{xor}_1 attr[k])$ [resp. $\sum_{k=0}^{n-1} 2^{n-1-k} \cdot (\mathsf{obj}[k] \ \mathbf{nxor}_1 \ attr[k])$], —where $\mathbf{xor}_n$ is the bitwise-xor operator and $\mathbf{nxor}_n$ is its complement— because $2^{n-1-i} > \sum_{k=i+1}^{n-1} 2^{n-1-k}$.

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

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

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 [21]: *apply the unsigned-$\mathcal{BV}$ maximization [resp. minimization] algorithm of [21] to the objective* $\mathsf{obj}' \stackrel{def}{=} (\mathsf{obj} \ \mathbf{nxor}_n \ attr)$ *[resp.* $\mathsf{obj}' \stackrel{def}{=} (\mathsf{obj} \ \mathbf{xor}_n \ attr)$*] instead than simply to* obj *[resp.* $\overline{\mathsf{obj}}$*].*

Example 3. Let $\mathsf{obj}^{[3]}$ be a signed $\mathcal{BV}$ goal of 3 bits to be minimized and $attr \stackrel{\text{def}}{=} [100]$ be its attractor, so that the corresponding vector of attractor equalities A is equal to $[\mathsf{obj}[0] = 1, \mathsf{obj}[1] = 0, \mathsf{obj}[2] = 0]$.

An assignment $\tau_3 \stackrel{\text{def}}{=} \{A[0], \neg A[1], \neg A[2]\}$ (for which $\mathsf{obj}^{[3]} = -\mathbf{1}^{[3]}$) is lexicographically better than $\tau_3' \stackrel{\text{def}}{=} \{\neg A[0], A[1], A[2]\}$ (for which $\mathsf{obj}^{[3]} = \mathbf{0}^{[3]}$), because the former satisfies the *attractor equality* corresponding to the MSB while the latter does not. Moreover, the assignment τ_3 is lexicographically worse than the assignment $\tau_3'' \stackrel{\text{def}}{=} \{A[0], \neg A[1], A[2]\}$ (for which $\mathsf{obj}^{[3]} = -\mathbf{2}^{[3]}$), because –all the rest being equal– the latter assignment makes the *attractor equality* $(\mathsf{obj}[2] = 0)$ true. $\diamond$

3.2 Floating-Point Optimization

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

Definition 4. *(OMT($\mathcal{FP}$)). Let φ be a SMT($\mathcal{FP}$) formula and* obj *be a $\mathcal{FP}$ variable occurring in φ. We call an* **Optimization Modulo $\mathcal{FP}$ problem***, the problem of finding a model $\mathcal{M}$ for φ (if any) whose value of* obj*, denoted with* $\mathsf{min}_{\mathsf{obj}}(\varphi)$*, 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. $\mathcal{M}'(\mathsf{obj})$ is not* NAN*,*
- *some binary representation of* NAN*, otherwise.*

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

Definition 4 is made necessarily convoluted by the fact that 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. Also, requiring the optimal solution to be always different from NAN makes the resulting OMT($\mathcal{FP}$) problem $\langle \varphi \wedge \neg \mathsf{IsNaN}(\mathsf{obj}), \mathsf{obj} \rangle$ unsatisfiable when φ is satisfied only by models $\mathcal{M}$ s.t. $\mathcal{M}(\mathsf{obj})$ 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. $\mathcal{M}(\mathsf{obj})$ 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. $\mathcal{M}'(\mathsf{obj})$ is not NAN —by invoking an SMT($\mathcal{FP}$) solver on $\varphi \wedge \neg \mathsf{IsNaN}(\mathsf{obj})$ if the model $\mathcal{M}$ returned by the previous SMT call is s.t. $\mathcal{M}(\mathsf{obj})$ is NAN. If the solver returns UNSAT, then we conclude that the minimum is NAN.

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

Definition 5. *(Dynamic Attractor.) Let $\langle \varphi_{\mathsf{noNaN}}, \mathsf{obj} \rangle$ be a restricted OMT($\mathcal{FP}$) problem, where $\varphi_{\mathsf{noNaN}} \stackrel{\text{def}}{=} \varphi \wedge \neg \mathsf{IsNaN}(\mathsf{obj})$ 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. $A_{\tau_k}[i] \stackrel{\text{def}}{=} (\mathsf{obj}[i] = attr_{\tau_k}[i])$, $i \in [0..n-1]$.*

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

Then $\mathcal{M}(\mathsf{obj}) \leq \mathcal{M}'(\mathsf{obj})$ [resp. $\mathcal{M}(\mathsf{obj}) \geq \mathcal{M}'(\mathsf{obj})$].

Lemma 1 states that, given the current assignment τ_k to the k most-significant-bits of obj, $\mathsf{obj}[k] = attr_{\tau_k}[k]$ 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 $\langle \varphi_{\mathsf{noNaN}}, \mathsf{obj} \rangle$ be an 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 4. Let $\langle \varphi_{\mathsf{noNaN}}, \mathsf{obj} \rangle$ be an OMT($\mathcal{FP}$) problem where obj is a $\mathcal{FP}$ objective, of sort `(_ FP 3 5)`, to be minimized. 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* for obj wrt. τ_0 (i.e. $attr_{\tau_0}$) is equal to `(fp #b1 #b111 #b0000)` (i.e. $-\infty$), which gives a preference to any feasible value of obj in the negative domain.

If at some point of the optimization search 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. $+0$).

Furthermore, if later on we also find out that at least one bit in the exponent of obj can be assigned to 0 in a feasible solution of the problem that extends τ_i, for some i, then we can remove $+\infty$ from the optimization search interval. $\diamond$

Definition 6. *(Attractor Trajectory $\mathcal{A}_\varphi$). Consider the restricted OMT($\mathcal{FP}$) problem $\langle \varphi_{\mathsf{noNaN}}, \mathsf{obj} \rangle$ s.t. $\varphi_{\mathsf{noNaN}} \stackrel{def}{=} \varphi \wedge \neg \mathsf{IsNaN}(\mathsf{obj})$ 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** $\mathcal{A}_\varphi$ *as the vector* $[A_{\tau_0}[0], ..., A_{\tau_{n-1}}[n-1]]$.

The attractor trajectory $\mathcal{A}_\varphi$ contains those attractor equalities ($\mathsf{obj}[k] = attr_{\tau_k}[k]$) 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. $\mathsf{obj}[k]$) is still undecided in τ_k.

Example 5. Let $\langle \varphi_{\mathsf{noNaN}}, \mathsf{obj} \rangle$ 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 $29/2$ and it does not impose any other constraint on the value of obj. Given the sequence of (partial) assignments $\tau_0, ..., \tau_8$ in Figure 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 $[\mathsf{obj}[0] = 1, \mathsf{obj}[1] = 0, \mathsf{obj}[2] = 0, \mathsf{obj}[3] = 0, \mathsf{obj}[4] = 0, \mathsf{obj}[5] = 0, \mathsf{obj}[6] = 0, \mathsf{obj}[7] = 0]$. $\diamond$

$\tau_0 = \emptyset$ $\quad attr_{\tau_0} = \texttt{(fp \#b1 \#b111 \#b0000)} = [\underline{1}.111.1111]$ [i.e. $-\infty$] $\Rightarrow$ UNSAT

$\tau_1 = \tau_0 \cup \{\mathsf{obj}[0] = 0\}$ $\quad attr_{\tau_1} = \texttt{(fp \#b0 \#b000 \#b0000)} = [0.\underline{0}00.0000]$ [i.e. +0] $\Rightarrow$ UNSAT

$\tau_2 = \tau_1 \cup \{\mathsf{obj}[1] = 1\}$ $\quad attr_{\tau_2} = \texttt{(fp \#b0 \#b100 \#b0000)} = [0.1\underline{0}0.0000]$ [i.e. +2] $\Rightarrow$ UNSAT

$\tau_3 = \tau_2 \cup \{\mathsf{obj}[2] = 1\}$ $\quad attr_{\tau_3} = \texttt{(fp \#b0 \#b110 \#b0000)} = [0.11\underline{0}.0000]$ [i.e. +8] $\Rightarrow$ SAT

$\tau_4 = \tau_3 \cup \{\mathsf{obj}[3] = 0\}$ $\quad attr_{\tau_4} = \texttt{(fp \#b0 \#b110 \#b0000)} = [0.110.\underline{0}000]$ [$''$ $''$] $\Rightarrow$ UNSAT

$\tau_5 = \tau_4 \cup \{\mathsf{obj}[4] = 1\}$ $\quad attr_{\tau_5} = \texttt{(fp \#b0 \#b110 \#b1000)} = [0.110.1\underline{0}00]$ [i.e. +12] $\Rightarrow$ UNSAT

$\tau_6 = \tau_5 \cup \{\mathsf{obj}[5] = 1\}$ $\quad attr_{\tau_6} = \texttt{(fp \#b0 \#b110 \#b1100)} = [0.110.11\underline{0}0]$ [i.e. +14] $\Rightarrow$ SAT

$\tau_7 = \tau_6 \cup \{\mathsf{obj}[6] = 0\}$ $\quad attr_{\tau_7} = \texttt{(fp \#b0 \#b110 \#b1100)} = [0.110.1100]$ [$''$ $''$] $\Rightarrow$ UNSAT

$\tau_8 = \tau_7 \cup \{\mathsf{obj}[7] = 1\}$ $\quad attr_{\tau_8} = \texttt{(fp \#b0 \#b110 \#b1101)} = [0.110.1101]$ [i.e. $29/2$]

$A_{\tau_0} = [\underline{\mathsf{obj}[0] = 1}, \mathsf{obj}[1] = 1, \mathsf{obj}[2] = 1, \mathsf{obj}[3] = 1, \mathsf{obj}[4] = 0, \mathsf{obj}[5] = 0, \mathsf{obj}[6] = 0, \mathsf{obj}[7] = 0]$

$A_{\tau_1} = [\mathsf{obj}[0] = 0, \underline{\mathsf{obj}[1] = 0}, \mathsf{obj}[2] = 0, \mathsf{obj}[3] = 0, \mathsf{obj}[4] = 0, \mathsf{obj}[5] = 0, \mathsf{obj}[6] = 0, \mathsf{obj}[7] = 0]$

$A_{\tau_2} = [\mathsf{obj}[0] = 0, \mathsf{obj}[1] = 1, \underline{\mathsf{obj}[2] = 0}, \mathsf{obj}[3] = 0, \mathsf{obj}[4] = 0, \mathsf{obj}[5] = 0, \mathsf{obj}[6] = 0, \mathsf{obj}[7] = 0]$

$A_{\tau_3} = [\mathsf{obj}[0] = 0, \mathsf{obj}[1] = 1, \mathsf{obj}[2] = 1, \underline{\mathsf{obj}[3] = 0}, \mathsf{obj}[4] = 0, \mathsf{obj}[5] = 0, \mathsf{obj}[6] = 0, \mathsf{obj}[7] = 0]$

$A_{\tau_4} = [\mathsf{obj}[0] = 0, \mathsf{obj}[1] = 1, \mathsf{obj}[2] = 1, \mathsf{obj}[3] = 0, \underline{\mathsf{obj}[4] = 0}, \mathsf{obj}[5] = 0, \mathsf{obj}[6] = 0, \mathsf{obj}[7] = 0]$

$A_{\tau_5} = [\mathsf{obj}[0] = 0, \mathsf{obj}[1] = 1, \mathsf{obj}[2] = 1, \mathsf{obj}[3] = 0, \mathsf{obj}[4] = 1, \underline{\mathsf{obj}[5] = 0}, \mathsf{obj}[6] = 0, \mathsf{obj}[7] = 0]$

$A_{\tau_6} = [\mathsf{obj}[0] = 0, \mathsf{obj}[1] = 1, \mathsf{obj}[2] = 1, \mathsf{obj}[3] = 0, \mathsf{obj}[4] = 1, \mathsf{obj}[5] = 1, \underline{\mathsf{obj}[6] = 0}, \mathsf{obj}[7] = 0]$

$A_{\tau_7} = [\mathsf{obj}[0] = 0, \mathsf{obj}[1] = 1, \mathsf{obj}[2] = 1, \mathsf{obj}[3] = 0, \mathsf{obj}[4] = 1, \mathsf{obj}[5] = 1, \mathsf{obj}[6] = 0, \underline{\mathsf{obj}[7] = 0}]$

$A_{\tau_8} = [\mathsf{obj}[0] = 0, \mathsf{obj}[1] = 1, \mathsf{obj}[2] = 1, \mathsf{obj}[3] = 0, \mathsf{obj}[4] = 1, \mathsf{obj}[5] = 1, \mathsf{obj}[6] = 0, \mathsf{obj}[7] = 1]$

Fig. 1. An example of $\mathcal{FP}$ optimization using the dynamic attractor. ("$\Rightarrow$ SAT/UNSAT" denotes the satisfiability of $\varphi_{\mathsf{noNaN}} \wedge \tau_k \wedge A_{\tau_k}[k]$, the symbols "$''$ $''$" stand for "the same as above". 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 $\mathcal{A}_\varphi$ inside the vectors of attractor equalities.)

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

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

4 OMT($\mathcal{FP}$) Procedures

In this paper, we consider two approaches for dealing with OMT($\mathcal{FP}$): a basic linear/binary search, based on the inline OMT schema for OMT($\mathcal{LRA} \cup \mathcal{T}$) presented in [25], and *Floating-Point Optimization with Binary Search* (OFP-BS), a brand-new engine inspired by the OBV-BS algorithm for unsigned Bit-Vectors in [21] and by Theorem 2 and relative definitions in §3.2.

4.1 OMT-based Approach

The OMT-based approach for OMT($\mathcal{FP}$) adapts the linear- and binary-search schemata for OMT($\mathcal{LRA} \cup \mathcal{T}$) presented in [25] 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. 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. $\mathsf{obj} = \text{NAN}$, the OMT solver learns a unit-clause of the form $\neg(\text{ISNAN}(\mathsf{obj}))$ 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 [25], 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.

4.2 Floating-Point Optimization with Binary Search

The *Floating-Point Optimization with Binary Search* algorithm is a new engine for OMT($\mathcal{FP}$) which is inspired by the OBV-BS algorithm for OMT($\mathcal{BV}$) [21] and is a direct implementation of Definition 6 and Theorem 2.

The optimization search tries to lexicographically maximize an 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 Figure 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 $n \stackrel{\text{def}}{=} ebits + sbits$ bits overall.

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

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 (lines 11-12). Then, the execution moves to the section of code implementing the core part of the OFP-BS algorithm (lines 15-28), which consists of a loop over the bits of obj, starting from the MSB $\mathsf{obj}[0]$ down to the LSB $\mathsf{obj}[n-1]$.

Inside this loop, OFP-BS first checks whether the value of $\mathsf{obj}[i]$ 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 $\mathsf{obj}[i] = attr_\tau[i]$ (line 16), and the optimization search moves to the next

function OFP-BS (φ, obj)

```
 1: ⟨res, M⟩ := SMT.CHECK_UNDER_ASSUMPTIONS(φ, ∅)
 2: if (res == UNSAT) then
 3:     return ⟨res, ∅⟩                                  // φ is unsatisfiable
 4: if (M(obj) == NaN) then
 5:     ⟨res, M'⟩ := SMT.CHECK_UNDER_ASSUMPTIONS(φ ∧ ¬IsNaN(obj), ∅)
 6:     if (res == UNSAT) then
 7:         return ⟨SAT, M⟩                              // obj can only be NaN
 8:     else
 9:         M := M'
10:         φ := φ ∧ ¬IsNaN(obj)
11: τ := ∅                  // from now on, obj cannot be equal NaN
12: attr_τ := UPDATE_DYNAMIC_ATTRACTOR(τ)
13: SMT.SET_BRANCHING_PREFERENCE(obj)
14: SMT.UPDATE_BITS_POLARITY_TO(obj, attr_τ)
15: for i := 0 up to n − 1 do
16:     eq := (obj[i] = attr_τ[i])                       // attractor equality A_τ[i]
17:     if (M ⊨ eq) then
18:         τ := τ ∪ {eq}
19:     else
20:         SMT.SET_BRANCHING_PREFERENCE(obj)
21:         SMT.UPDATE_BITS_POLARITY_TO(obj, attr_τ)
22:         ⟨res, M'⟩ := SMT.CHECK_UNDER_ASSUMPTIONS(φ, τ ∪ {eq})
23:         if (res == SAT) then
24:             τ := τ ∪ {eq}
25:             M := M'
26:         else
27:             τ := τ ∪ {¬eq}
28:             attr_τ := UPDATE_DYNAMIC_ATTRACTOR(τ)
29: return ⟨SAT, M⟩
```

Fig. 2. OFP-BS Algorithm for Floating-Point optimization.

iteration of the loop. If instead $\mathsf{obj}[i] \neq attr_\tau[i]$ 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 $\tau \cup \{\mathsf{obj}[i] = attr_\tau[i]\}$ (line 22). 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 $\mathsf{obj}[i]$ equal to $attr_\tau[i]$, and $\mathcal{M}$ is replaced with the new model $\mathcal{M}'$ (lines 23-25). Otherwise, it is not possible to improve the objective function by toggling the value of $\mathsf{obj}[i]$, and τ is extended so as to permanently set $\mathsf{obj}[i] \neq attr_\tau[i]$ (line 27). 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 as parameter (line 28). 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 $\langle \text{SAT}, \mathcal{M} \rangle$, where $\mathcal{M}$ is the optimum model of the given OMT($\mathcal{FP} \cup \mathcal{T}$) instance. The OFP-BS algorithm requires at most $n+2$ incremental calls to an underlying SMT($\mathcal{FP}$) solver. The test in rows 17-18 allows for saving lots of such SMT calls when the current model already assigns $\mathsf{obj}[i]$ 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, when obj is minimized [2], and it 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. Since $\tau \neq \emptyset$ then we know that the sign of the objective function has been permanently decided in τ. If $\mathsf{obj}[0] = 0$ in τ, i.e. obj must be positive, the procedure must return the smallest positive $\mathcal{FP}$ value admitted by τ. Hence, we extend τ with $\bigcup_{i=|\tau|}^{i=n-1} \mathsf{obj}[i] = 0$ and return the corresponding $\mathcal{FP}$ value. If $\mathsf{obj}[0] = 1$ in τ, i.e. obj can be negative values, the procedure must return the largest negative $\mathcal{FP}$ value admitted by τ. We first check whether there exists a bit in the exponent of obj which is assigned to 0 in τ. If that is the case, we extend τ with $\bigcup_{i=|\tau|}^{i=n-1} \mathsf{obj}[i] = 1$ and return the corresponding $\mathcal{FP}$ value. Otherwise, the procedure returns the value $-\infty$, which is still a viable extension of τ.

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 OMT($\mathcal{BV}$) by Nadel et al. in [21].

- **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 $\mathsf{obj}[i]$ 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 Section §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.

In the case of the OFP-BS algorithm described in Section §4.2, we use the latest value of the dynamic attractor $attr_\tau$ for both the *branching preference* (lines 11 and 18

[2] The implementation of UPDATE_DYNAMIC_ATTRACTOR() is dual when obj is maximized.

of Figure 2) and the *polarity initialization* (rows 12 and 19 of Figure 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. 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 assess the performance of OPTIMATHSAT (v. 1.6.2) on a set of OMT($\mathcal{FP}$) formulas that have been automatically generated using the SMT($\mathcal{FP}$) benchmark-set of [3]. The formulas, the results and the scripts necessary to reproduce these results are made publicly available and can be downloaded from [1].

Experiment Setup. This experiment has been performed on an *i7-6500U 2.50GHz Intel Quad-Core* machine with $16GB$ of ram and running *Ubuntu Linux* 17.10. For each formula being tested we used a timeout of 600 seconds. 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 [3]. We did not consider any of the unsatisfiable instances that are present in the remote repository. Since the majority of the original SMT($\mathcal{FP}$) formulas admits only one solution, in order to increase the significance of the resulting OMT($\mathcal{FP}$) benchmark set, we relaxed or removed some of the constraints in these formulas so as to broaden the set of feasible solutions.

We consider two OMT-based baseline configurations, 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 approach, 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[3] presented by Nadel et al. in [21]. 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 Section §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 Section §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. For all problem instances, we verified the correctness of

[3] The binaries of the original OMT($\mathcal{BV}$) tools presented in [21] are not publicly available.

tool, configuration & encoding	inst.	term.	t.o.	u	bt	st	time (s.)
OPTIMATHSAT(EAGER+OMT+LIN)	1120	1003	117	0	5	73	76375
OPTIMATHSAT(EAGER+OMT+LIN+PI)	1120	1003	117	0	5	71	76785
OPTIMATHSAT(EAGER+OMT+LIN+BP)	1120	956	164	0	6	105	77480
OPTIMATHSAT(EAGER+OMT+LIN+BP+PI)	1120	873	247	0	77	217	54859
OPTIMATHSAT(EAGER+OMT+BIN)	1120	1014	106	0	11	281	67834
OPTIMATHSAT(EAGER+OMT+BIN+PI)	1120	970	150	0	8	285	69765
OPTIMATHSAT(EAGER+OMT+BIN+BP)	1120	1016	104	0	14	205	68255
OPTIMATHSAT(EAGER+OMT+BIN+BP+PI)	1120	991	129	0	65	**321**	56941
OPTIMATHSAT(LAZY+OMT+LIN)	1120	868	252	0	93	203	29832
OPTIMATHSAT(LAZY+OMT+BIN)	1120	900	220	0	90	243	33260
OPTIMATHSAT(EAGER+OBVBS) [REDUCTION]	1120	1013	107	0	14	141	65954
OPTIMATHSAT(EAGER+OFPBS)	1120	1017	103	0	9	171	70732
OPTIMATHSAT(EAGER+OFPBS+PI)	1120	**1019**	101	0	34	280	64896
OPTIMATHSAT(EAGER+OFPBS+PI+SO)	1120	1018	102	0	7	179	71430
OPTIMATHSAT(EAGER+OFPBS+BP)	1120	975	145	0	2	145	65543
OPTIMATHSAT(EAGER+OFPBS+BP+SO)	1120	1000	120	0	3	124	68390
OPTIMATHSAT(EAGER+OFPBS+BP+PI)	1120	1001	119	0	77	273	60365
OPTIMATHSAT(EAGER+OFPBS+BP+PI+SO)	1120	1006	114	**19**	32	245	59463
VIRTUAL BEST	1120	**1074**	46	-	559	1074	27788

Table 1. Comparison among various OPTIMATHSAT configurations on the OMT($\mathcal{FP}$) benchmark-set. 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 in the shortest amount of time (st) and the total solving time for all solved instances (time).

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 1.

For what concerns 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. Enabling only *polarity initialization* does not result in an appreciable change on the running time of the solver. In contrast, enabling both enhancements at the same time generally worsens the performance and results in a drastic increase in the number of timeouts (Table 1). 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 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. 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.

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 the given benchmark-set– the performance of this approach are comparable with the best 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 being solved in correspondence to the OPTIMATHSAT(EAGER+OFP-BS+PI) configuration. Similarly to the case of OMT-based optimization with linear-search, we observe that enabling *branching preference* generally makes the performance worse. 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*. This feature is particularly effective when used in combination with *branching preference*.

6 Conclusions and Future Work

We have presented for the first time OMT procedures (for signed Bit-Vectors and) Floating-Point numbers, based on the novel notions of attractor, dynamic attractor and attractor trajectory, 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 [2].

References

1. http://disi.unitn.it/trentin/resources/floatingpoint_test.tar.gz.
2. NUXMV. https://nuxmv.fbk.eu.
3. SmtLibv2. www.smtlib.cs.uiowa.edu/.
4. IEEE standard 754, 2008. http://grouper.ieee.org/groups/754/.
5. N. Bjorner and A.-D. Phan. νZ - Maximal Satisfaction with Z3. In *Proc International Symposium on Symbolic Computation in Software Science*, Gammart, Tunisia, December 2014. EasyChair Proceedings in Computing (EPiC).
6. N. Bjorner, A.-D. Phan, and L. Fleckenstein. νZ - An Optimizing SMT Solver. In *Proc. TACAS*, volume 9035 of *LNCS*. Springer, 2015.
7. M. Bozzano, R. Bruttomesso, A. Cimatti, A. Franzèn, Z. Hanna, Z. Khasidashvili, A. Palti, and R. Sebastiani. 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.
8. M. Brain, V. D'Silva, A. Griggio, L. Haller, and D. Kroening. Interpolation-Based Verification of Floating-Point Programs with Abstract CDCL. In *SAS*, pages 412–432, 2013.
9. M. Brain, V. D'Silva, A. Griggio, L. Haller, and D. Kroening. Deciding floating-point logic with abstract conflict driven clause learning. *Formal Methods in System Design*, 45(2):213–245, 2014.
10. M. Brain, C. Tinelli, P. Rümmer, and T. Wahl. An Automatable Formal Semantics for IEEE-754 Floating-Point Arithmetic. In *ARITH*, pages 160–167. IEEE, 2015.
11. A. Brillout, D. Kroening, and T. Wahl. Mixed abstractions for floating-point arithmetic. In *2009 Formal Methods in Computer-Aided Design*, pages 69–76, Nov 2009.
12. R. Brinkmann and R. Drechsler. RTL-datapath verification using integer linear programming. In *Proc. ASP-DAC 2002*, pages 741–746. IEEE, 2002.
13. R. Brummayer and A. Biere. Boolector: An efficient smt solver for bit-vectors and arrays. In *TACAS*, pages 174–177, Berlin, Heidelberg, 2009. Springer-Verlag.
14. R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, Z. Hanna, A. Nadel, A. Palti, and R. Sebastiani. A Lazy and Layered SMT($\mathcal{BV}$) Solver for Hard Industrial Verification Problems. In *CAV*, volume 4590 of *LNCS*, pages 547–560. Springer, 2007.
15. A. Cimatti, A. Franzén, A. Griggio, R. Sebastiani, and C. Stenico. Satisfiability modulo the theory of costs: Foundations and applications. In *TACAS*, volume 6015 of *LNCS*, pages 99–113. Springer, 2010.
16. K. Fazekas, F. Bacchus, and A. Biere. Implicit Hitting Set Algorithms for Maximum Satisfiability Modulo Theories. In *IJCAR*, volume 10900 of *Lecture Notes in Computer Science*, pages 134–151. Springer, 2018.
17. V. Ganesh and D. L. Dill. A Decision Procedure for Bit-Vectors and Arrays. In *CAV*, 2007.
18. L. Hadarean, K. Bansal, D. Jovanovic, C. Barrett, and C. Tinelli. 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.
19. D. Larraz, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. Minimal-Model-Guided Approaches to Solving Polynomial Constraints and Extensions. In *SAT*, 2014.
20. Y. Li, A. Albarghouthi, Z. Kincad, A. Gurfinkel, and M. Chechik. Symbolic Optimization with SMT Solvers. In *POPL*, 2014.
21. A. Nadel and V. Ryvchin. Bit-Vector Optimization. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2016*, volume 9636 of *LNCS*. Springer, 2016.
22. A. Niemetz, M. Preiner, A. Fröhlich, and A. Biere. 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.

23. R. Nieuwenhuis and A. Oliveras. On SAT Modulo Theories and Optimization Problems. In *Proc. Theory and Applications of Satisfiability Testing - SAT 2006*, volume 4121 of *LNCS*. Springer, 2006.
24. P. Ruemmer and T. Wahl. 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`.
25. R. Sebastiani and S. Tomasi. Optimization Modulo Theories with Linear Rational Costs. *ACM Transactions on Computational Logics*, 16(2), March 2015.
26. R. Sebastiani and P. Trentin. 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.
27. R. Sebastiani and P. Trentin. OptiMathSAT: A Tool for Optimization Modulo Theories. *Journal of Automated Reasoning*, Dec 2018.
28. A. Zeljić, P. Backeman, C. M. Wintersteiger, and P. Rümmer. Exploring approximations for floating-point arithmetic using uppsat. In D. Galmiche, S. Schulz, and R. Sebastiani, editors, *Automated Reasoning*, pages 246–262, Cham, 2018. Springer International Publishing.
29. A. Zeljić, C. M. Wintersteiger, and P. Rümmer. Approximations for model construction. In S. Demri, D. Kapur, and C. Weidenbach, editors, *Automated Reasoning*, pages 344–359, Cham, 2014. Springer International Publishing.