Satisfiability Modulo Exponential Integer Arithmetic*

Florian Frohn and Jürgen Giesl(✉)

RWTH Aachen University, Aachen, Germany
{florian.frohn,giesl}@informatik.rwth-aachen.de

Abstract. SMT solvers use sophisticated techniques for polynomial (linear or non-linear) integer arithmetic. In contrast, non-polynomial integer arithmetic has mostly been neglected so far. However, in the context of program verification, polynomials are often insufficient to capture the behavior of the analyzed system without resorting to approximations. In the last years, *incremental linearization* has been applied successfully to satisfiability modulo real arithmetic with transcendental functions. We adapt this approach to an extension of polynomial integer arithmetic with exponential functions. Here, the key challenge is to compute suitable *lemmas* that eliminate the current model from the search space if it violates the semantics of exponentiation. An empirical evaluation of our implementation shows that our approach is highly effective in practice.

1 Introduction

Traditionally, automated reasoning techniques for integers focus on polynomial arithmetic. This is not only true in the context of SMT, but also for program verification techniques, since the latter often search for polynomial invariants that imply the desired properties. As invariants are over-approximations, they are well suited for proving "universal" properties like safety, termination, or upper bounds on the worst-case runtime that refer to all possible program runs. However, proving dual properties like unsafety, non-termination, or lower bounds requires under-approximations, so that invariants are of limited use here.

For lower bounds, an *infinite set* of witnesses is required, as the runtime w.r.t. a finite set of (terminating) program runs is always bounded by a constant. Thus, to prove non-constant lower bounds, *symbolic under-approximations* are required, i.e., formulas that describe an infinite subset of the reachable states. However, polynomial arithmetic is often insufficient to express such approximations. To see this, consider the program

$$x \leftarrow 1;\ y \leftarrow \texttt{nondet}(0, \infty);\ \textbf{while}\ y > 0\ \textbf{do}\ x \leftarrow 3 \cdot x;\ y \leftarrow y - 1\ \textbf{done}$$

where $\texttt{nondet}(0, \infty)$ returns a natural number non-deterministically. Here, the set of reachable states after execution of the loop is characterized by the formula

* funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 235950644 (Project GI 274/6-2)

$$\exists n \in \mathbb{N}.\ x = 3^n \wedge y = 0. \tag{1}$$

In recent work, *acceleration techniques* have successfully been used to deduce lower runtime bounds automatically [17,18]. While they can easily derive a formula like (1) from the code above, this is of limited use, as most[1] SMT solvers cannot handle terms of the form 3^n. Besides lower bounds, acceleration has also successfully been used for proving non-termination [15,18,19] and (un)safety [3,6,7,20,28,29], where its strength is finding long counterexamples that are challenging for other techniques.

Importantly, exponentiation is not just "yet another function" that can result from applying acceleration techniques. There are well-known, important classes of loops where polynomials and exponentiation *always* suffice to represent the values of the program variables after executing a loop [16,26]. Thus, the lack of support for integer exponentiation in SMT solvers is a major obstacle for the further development of acceleration-based verification techniques.

In this work, we first define a novel SMT theory for integer arithmetic with exponentiation. Then we show how to lift standard SMT solvers to this new theory, resulting in our novel tool SwInE (SMT with Integer Exponentiation).

Our technique is inspired by *incremental linearization*, which has been applied successfully to *real* arithmetic with transcendental functions, including the natural exponential function $\mathsf{exp}_e(x) = e^x$, where e is Euler's number [11]. In this setting, incremental linearization considers exp_e as an uninterpreted function. If the resulting SMT problem is unsatisfiable, then so is the original problem. If it is satisfiable and the model that was found for exp_e coincides with the semantics of exponentiation, then the original problem is satisfiable. Otherwise, *lemmas* about exp_e that rule out the current model are added to the SMT problem, and then its satisfiability is checked again. The name "incremental linearization" is due to the fact that these lemmas only contain linear arithmetic.

The main challenge for adapting this approach to integer exponentiation is to generate suitable lemmas, see Sect. 4.2. Except for so-called *monotonicity lemmas*, none of the lemmas from [11] easily carry over to our setting. In contrast to [11], we do not restrict ourselves to linear lemmas, but we also use non-linear, polynomial lemmas. This is due to the fact that we consider a binary version $\lambda x, y.\ x^y$ of exponentiation, whereas [11] fixes the base to e. Thus, in our setting, one obtains *bilinear* lemmas that are linear w.r.t. x as well as y, but may contain multiplication between x and y (i.e., they may contain the subterm $x \cdot y$). More precisely, bilinear lemmas arise from *bilinear interpolation*, which is a crucial ingredient of our approach, as it allows us to eliminate *any* model that violates the semantics of exponentiation (Thm. 23). Therefore, the name "incremental linearization" does not fit to our approach, which is rather an instance of "counterexample-guided abstraction refinement" (CEGAR) [13].

To summarize, our contributions are as follows: We first propose the new SMT theory EIA for integer arithmetic with exponentiation (Sect. 3). Then, based on novel techniques for generating suitable lemmas, we develop a CEGAR approach for EIA (Sect. 4). We implemented our approach in our novel open-source tool

[1] CVC5 uses a dedicated solver for integer exponentiation with base 2.

SwInE [22,23] and evaluated it on a collection of 4627 EIA benchmarks that we synthesized from verification problems. Our experiments show that our approach is highly effective in practice (Sect. 6). All proofs can be found in [21].

2 Preliminaries

We are working in the setting of *SMT-LIB logic* [4], a variant of many-sorted first-order logic with equality. We now introduce a reduced variant of [4], where we only explain those concepts that are relevant for our work.

In SMT-LIB logic, there is a dedicated Boolean sort **Bool**, and hence formulas are just terms of sort **Bool**. Similarly, there is no distinction between predicates and functions, as predicates are simply functions of type **Bool**.

So in SMT-LIB logic, a *signature* $\Sigma = (\Sigma^S, \Sigma^F, \Sigma^R)$ consists of a set Σ^S of *sorts*, a set Σ^F of *function symbols*, and a *ranking function* $\Sigma^R : \Sigma^F \to (\Sigma^S)^+$. The meaning of $\Sigma^R(f) = (s_1, \ldots, s_k)$ is that f is a function which maps arguments of the sorts $s_1, \ldots, s_{k-1}$ to a result of sort s_k. We write $f : s_1 \ \ldots \ s_k$ instead of "$f \in \Sigma^F$ and $\Sigma^R(f) = (s_1, \ldots, s_k)$" if Σ is clear from the context. We always allow to implicitly extend Σ with arbitrarily many constant function symbols (i.e., function symbols x where $|\Sigma^R(x)| = 1$). Note that SMT-LIB logic only considers closed terms, i.e., terms without free variables, and we are only concerned with quantifier-free formulas, so in our setting, all formulas are ground. Therefore, we refer to these constant function symbols as *variables* to avoid confusion with other, predefined constant function symbols like $\mathbf{true}, 0, \ldots$, see below.

Every SMT-LIB signature is an extension of $\Sigma_{\mathbf{Bool}}$ where $\Sigma^S_{\mathbf{Bool}} = \{\mathbf{Bool}\}$ and $\Sigma^F_{\mathbf{Bool}}$ consists of the following function symbols:

$$\mathbf{true}, \mathbf{false} : \mathbf{Bool} \qquad \neg : \mathbf{Bool}\ \mathbf{Bool} \qquad \wedge, \vee, \implies, \iff : \mathbf{Bool}\ \mathbf{Bool}\ \mathbf{Bool}$$

Note that SMT-LIB logic only considers well-sorted terms. A Σ-*structure* $\mathbf{A}$ consists of a *universe* $A = \bigcup_{s \in \Sigma^S} A_s$ and an *interpretation function* that maps each function symbol $f : s_1 \ \ldots \ s_k$ to a function $[\![f]\!]^{\mathbf{A}} : A_{s_1} \times \ldots \times A_{s_{k-1}} \to A_{s_k}$. SMT-LIB logic only considers structures where $A_{\mathbf{Bool}} = \{\mathbf{true}, \mathbf{false}\}$ and all function symbols from $\Sigma_{\mathbf{Bool}}$ are interpreted as usual.

A Σ-*theory* is a class of Σ-structures. For example, consider the extension $\Sigma_{\mathbf{Int}}$ of $\Sigma_{\mathbf{Bool}}$ with the additional sort **Int** and the following function symbols:

$$0, 1, \ldots : \mathbf{Int} \qquad +, -, \cdot, \mathsf{div}, \mathsf{mod} : \mathbf{Int}\ \mathbf{Int}\ \mathbf{Int} \qquad <, \leq, >, \geq, =, \neq : \mathbf{Int}\ \mathbf{Int}\ \mathbf{Bool}$$

Then the $\Sigma_{\mathbf{Int}}$-theory *non-linear integer arithmetic* (NIA)[2] contains all $\Sigma_{\mathbf{Int}}$-structures where $A_{\mathbf{Int}} = \mathbb{Z}$ and all symbols from $\Sigma_{\mathbf{Int}}$ are interpreted as usual.

If $\mathbf{A}$ is a Σ-structure and Σ' is a subsignature of Σ, then the *reduct* of $\mathbf{A}$ to Σ' is the unique Σ'-structure that interprets its function symbols like $\mathbf{A}$. So the

[2] As we only consider quantifier-free formulas, we omit the prefix "QF_" in theory names and write, e.g., NIA instead of QF_NIA. In [4], QF_NIA is called an *SMT-LIB logic*, which restricts the (first-order) *theory* of integer arithmetic to the quantifier-free fragment. For simplicity, we do not distinguish between SMT-LIB logics and theories.

theory *linear integer arithmetic* (LIA) consists of the reducts of all elements of NIA to $\Sigma_{\mathbf{Int}} \setminus \{\cdot, \mathsf{div}, \mathsf{mod}\}$.

Given a Σ-structure $\mathbf{A}$ and a Σ-term t, the *meaning* $[\![t]\!]^{\mathbf{A}}$ of t results from interpreting all function symbols according to $\mathbf{A}$. For function symbols f whose interpretation is fixed by a Σ-theory $\mathcal{T}$, we denote f's interpretation by $[\![f]\!]^{\mathcal{T}}$. Given a Σ-theory $\mathcal{T}$, a Σ-formula φ (i.e., a Σ-term of type **Bool**) is *satisfiable in* $\mathcal{T}$ if there is an $\mathbf{A} \in \mathcal{T}$ such that $[\![\varphi]\!]^{\mathbf{A}} = \mathbf{true}$. Then $\mathbf{A}$ is called a *model* of φ, written $\mathbf{A} \models \varphi$. If *every* $\mathbf{A} \in \mathcal{T}$ is a model of φ, then φ is $\mathcal{T}$*-valid*, written $\models_{\mathcal{T}} \varphi$. We write $\psi \equiv_{\mathcal{T}} \varphi$ for $\models_{\mathcal{T}} \psi \iff \varphi$.

We sometimes also consider *uninterpreted functions*. Then the signature may not only contain the function symbols of the theory under consideration and variables, but also additional non-constant function symbols.

We write "term", "structure", "theory", ... instead of "Σ-term", "Σ-structure", "Σ-theory", ... if Σ is irrelevant or clear from the context. Similarly, we just write "$\equiv$" and "valid" instead of "$\equiv_{\mathcal{T}}$" and "$\mathcal{T}$-valid" if $\mathcal{T}$ is clear from the context. Moreover, we use unary minus and t^c (where t is a term of sort **Int** and $c \in \mathbb{N}$) as syntactic sugar, and we use infix notation for binary function symbols.

In the sequel, we use $x, y, z, \ldots$ for variables, $s, t, p, q, \ldots$ for terms of sort **Int**, $\varphi, \psi, \ldots$ for formulas, and $a, b, c, d, \ldots$ for integers.

3 The SMT Theory EIA

We now introduce our novel SMT theory for *exponential integer arithmetic*. To this end, we define the signature $\Sigma^{\mathsf{exp}}_{\mathbf{Int}}$, which extends $\Sigma_{\mathbf{Int}}$ with

$$\mathsf{exp} : \mathbf{Int}\ \mathbf{Int}\ \mathbf{Int}.$$

If the 2^{nd} argument of exp is non-negative, then its semantics is as expected, i.e., we are interested in structures $\mathbf{A}$ such that $[\![\mathsf{exp}]\!]^{\mathbf{A}}(c, d) = c^d$ for all $d \geq 0$. However, if the 2^{nd} argument is negative, then we have to use different semantics. The reason is that we may have $c^d \notin \mathbb{Z}$ if $d < 0$. Intuitively, exp should be a partial function, but all functions are total in SMT-LIB logic. We solve this problem by interpreting $\mathsf{exp}(c, d)$ as $c^{|d|}$. This semantics has previously been used in the literature, and the resulting logic admits a known decidable fragment [5].

Definition 1 (EIA). *The theory* exponential integer arithmetic (EIA) *contains all* $\Sigma^{\mathsf{exp}}_{\mathbf{Int}}$-structures $\mathbf{A}$ with $[\![\mathsf{exp}]\!]^{\mathbf{A}}(c, d) = c^{|d|}$ whose reduct to $\Sigma_{\mathbf{Int}}$ is in NIA.

Alternatively, one could treat $\mathsf{exp}(c, d)$ like an uninterpreted function if d is negative. Doing so would be analogous to the treatment of division by zero in SMT-LIB logic. Then, e.g., $\mathsf{exp}(0, -1) \neq \mathsf{exp}(0, -2)$ would be satisfied by a structure $\mathbf{A}$ with $[\![\mathsf{exp}]\!]^{\mathbf{A}}(c, d) = c^d$ if $d \geq 0$ and $[\![\mathsf{exp}]\!]^{\mathbf{A}}(c, d) = d$, otherwise. However, the drawback of this approach is that important laws of exponentiation like

$$\mathsf{exp}(\mathsf{exp}(x, y), z) = \mathsf{exp}(x, y \cdot z)$$

would not be valid. Thus, we focus on the semantics from Def. 1.

Algorithm 1: CEGAR for EIA

Input: a $\Sigma_{\mathbf{Int}}^{\mathsf{exp}}$-formula φ
// Preprocessing
1 **do**
2 $\quad \varphi' \leftarrow \varphi$
3 $\quad \varphi \leftarrow \textsc{FoldConstants}(\varphi)$
4 $\quad \varphi \leftarrow \textsc{Rewrite}(\varphi)$
5 **while** $\varphi \neq \varphi'$
// Refinement Loop
6 **while** *there is a NIA-model* $\mathbf{A}$ *of* φ **do**
7 $\quad$ **if** $\mathbf{A}$ *is a counterexample* **then**
8 $\quad\quad \mathcal{L} \leftarrow \varnothing$
9 $\quad\quad$ **for** kind $\in \{$*Symmetry*, *Monotonicity*, *Bounding*, *Interpolation*$\}$ **do**
10 $\quad\quad\quad \mathcal{L} \leftarrow \mathcal{L} \cup \textsc{ComputeLemmas}(\varphi, \text{kind})$
11 $\quad\quad \varphi \leftarrow \varphi \wedge \bigwedge\{\psi \in \mathcal{L} \mid \mathbf{A} \not\models \psi\}$
12 $\quad$ **else return sat**
13 **return unsat**

4 Solving EIA Problems via CEGAR

We now explain our technique for solving EIA problems, see Alg. 1. Our goal is to (dis)prove satisfiability of φ in EIA. The loop in Line 6 is a CEGAR loop which lifts an SMT solver for NIA (which is called in Line 6) to EIA. So the *abstraction* consists of using NIA- instead of EIA-models. Hence, exp is considered to be an uninterpreted function in Line 6, i.e., the SMT solver also searches for an interpretation of exp. If the model found by the SMT solver is a *counterexample* (i.e., if $[\![\mathsf{exp}]\!]^{\mathbf{A}}$ conflicts with $[\![\mathsf{exp}]\!]^{\mathrm{EIA}}$), then the formula under consideration is refined by adding suitable lemmas in Lines 9 – 11 and the loop is iterated again.

Definition 2 (Counterexample). *We call a NIA-model* $\mathbf{A}$ *of* φ *a* counterexample *if there is a subterm* $\mathsf{exp}(s,t)$ of φ such that $[\![\mathsf{exp}(s,t)]\!]^{\mathbf{A}} \neq ([\![s]\!]^{\mathbf{A}})^{|[\![t]\!]^{\mathbf{A}}|}$.

In the sequel, we first discuss our preprocessings (first loop in Alg. 1) in Sect. 4.1. Then we explain our refinement (Lines 9 – 11) in Sect. 4.2. Here, we first introduce the different kinds of lemmas that are used by our implementation in Sect. 4.2.1 – 4.2.4. If implemented naively, the number of lemmas can get quite large, so we explain how to generate lemmas *lazily* in Sect. 4.2.5. Finally, we conclude this section by stating important properties of Alg. 1.

Example 3 (Leading Example). To illustrate our approach, we show how to prove

$$\forall x, y.\ |x| > 2 \wedge |y| > 2 \implies \mathsf{exp}(\mathsf{exp}(x,y),y) \neq \mathsf{exp}(x,\mathsf{exp}(y,y))$$

by encoding absolute values suitably[3] and proving unsatisfiability of its negation:

$$x^2 > 4 \wedge y^2 > 4 \wedge \mathsf{exp}(\mathsf{exp}(x,y),y) = \mathsf{exp}(x,\mathsf{exp}(y,y))$$

[3] We tested several encodings, but surprisingly, this non-linear encoding worked best.

4.1 Preprocessings In the first loop of Alg. 1, we preprocess φ by alternating *constant folding* (Line 3) and *rewriting* (Line 4) until a fixpoint is reached. Constant folding evaluates subexpressions without variables, where subexpressions $\mathsf{exp}(c, d)$ are evaluated to $c^{|d|}$, i.e., according to the semantics of EIA. Rewriting reduces the number of occurrences of exp via the following (terminating) rewrite rules:

$$\begin{aligned} \mathsf{exp}(x, c) &\to x^{|c|} && \text{if } c \in \mathbb{Z} \\ \mathsf{exp}(\mathsf{exp}(x, y), z) &\to \mathsf{exp}(x, y \cdot z) \\ \mathsf{exp}(x, y) \cdot \mathsf{exp}(z, y) &\to \mathsf{exp}(x \cdot z, y) \end{aligned}$$

In particular, the 1^{st} rule allows us to rewrite[4] $\mathsf{exp}(s, 0)$ to $s^0 = 1$ and $\mathsf{exp}(s, 1)$ to $s^1 = s$. Note that the rule

$$\mathsf{exp}(x, y) \cdot \mathsf{exp}(x, z) \to \mathsf{exp}(x, y + z)$$

would be unsound, as the right-hand side would need to be $\mathsf{exp}(x, |y| + |z|)$ instead. We leave the question whether such a rule is beneficial to future work.

Example 4 (Preprocessing). For our leading example, applying the 2^{nd} rewrite rule at the underlined position yields:

$$\begin{aligned} & x^2 > 4 \wedge y^2 > 4 \wedge \underline{\mathsf{exp}(\mathsf{exp}(x, y), y)} = \mathsf{exp}(x, \mathsf{exp}(y, y)) \\ \to\ & x^2 > 4 \wedge y^2 > 4 \wedge \mathsf{exp}(x, y^2) = \mathsf{exp}(x, \mathsf{exp}(y, y)) \end{aligned} \tag{2}$$

Lemma 5. *We have* $\varphi \equiv_{\text{EIA}} \textsc{FoldConstants}(\varphi)$ *and* $\varphi \equiv_{\text{EIA}} \textsc{Rewrite}(\varphi)$.

4.2 Refinement Our refinement (Lines 9 – 11 of Alg. 1) is based on the four kinds of lemmas named in Line 9: *symmetry lemmas*, *monotonicity lemmas*, *bounding lemmas*, and *interpolation lemmas*. In the sequel, we explain how we compute a set $\mathcal{L}$ of such lemmas. Then our refinement conjoins

$$\{\psi \in \mathcal{L} \mid \mathbf{A} \not\models \psi\}$$

to φ in Line 11. As our lemmas allow us to eliminate *any* counterexample, this set is never empty, see Thm. 23. To compute $\mathcal{L}$, we consider all terms that are *relevant* for the formula φ.

Definition 6 (Relevant Terms). *A term* $\mathsf{exp}(s, t)$ *is* relevant *if* φ *has a subterm of the form* $\mathsf{exp}(\pm s, \pm t)$.

Example 7 (Relevant Terms). For our leading example (2), the relevant terms are all terms of the form $\mathsf{exp}(\pm x, \pm y^2)$, $\mathsf{exp}(\pm y, \pm y)$, or $\mathsf{exp}(\pm x, \pm \mathsf{exp}(y, y))$.

While the formula φ is changed in Line 11 of Alg. 1, we only conjoin new lemmas to φ, and thus, relevant terms can never become irrelevant. Moreover, by construction our lemmas only contain exp-terms that were already relevant before. Thus, the set of relevant terms is not changed by our CEGAR loop.

As mentioned in Sect. 1, our approach may also compute lemmas with non-linear polynomial arithmetic. However, our lemmas are linear if s is an integer

[4] Note that we have $[\![\mathsf{exp}(0, 0)]\!]^{\text{EIA}} = 0^0 = 1$.

constant and t is linear for all subterms $\mathsf{exp}(s,t)$ of φ. Here, despite the fact that the function "mod" is not contained in the signature of LIA, we also consider literals of the form $s \bmod c = 0$ where $c \in \mathbb{N}_+ = \mathbb{N} \setminus \{0\}$ as linear. The reason is that, according to the SMT-LIB standard, LIA contains a function[5] "$\mathsf{divisible}_c$ **Int Bool**" for each $c \in \mathbb{N}_+$, which yields **true** iff its argument is divisible by c, and hence we have $s \bmod c = 0$ iff $\mathsf{divisible}_c(s)$.

In the sequel, $[\![\ldots]\!]$ means $[\![\ldots]\!]^{\mathbf{A}}$, where $\mathbf{A}$ is the model from Line 6 of Alg. 1.

4.2.1 Symmetry Lemmas *Symmetry lemmas* encode the relation between terms of the form $\mathsf{exp}(\pm s, \pm t)$. For each relevant term $\mathsf{exp}(s,t)$, the set $\mathcal{L}$ contains the following symmetry lemmas:

$$t \bmod 2 = 0 \implies \mathsf{exp}(s,t) = \mathsf{exp}(-s,t) \quad (\text{SYM}_1)$$
$$t \bmod 2 = 1 \implies \mathsf{exp}(s,t) = -\mathsf{exp}(-s,t) \quad (\text{SYM}_2)$$
$$\mathsf{exp}(s,t) = \mathsf{exp}(s,-t) \quad (\text{SYM}_3)$$

Note that SYM_1 and SYM_2 are just implications, not equivalences, as, for example, $c^{|d|} = (-c)^{|d|}$ does not imply $d \bmod 2 = 0$ if $c = 0$.

Example 8 (Symmetry Lemmas). For our leading example (2), the following symmetry lemmas would be considered, among others:

$$\text{SYM}_1: \quad -y \bmod 2 = 0 \implies \mathsf{exp}(-y,-y) = \mathsf{exp}(y,-y) \quad (3)$$
$$\text{SYM}_2: \quad -y \bmod 2 = 1 \implies \mathsf{exp}(-y,-y) = -\mathsf{exp}(y,-y) \quad (4)$$
$$\text{SYM}_3: \quad \mathsf{exp}(x,\mathsf{exp}(y,y)) = \mathsf{exp}(x,-\mathsf{exp}(y,y)) \quad (5)$$
$$\text{SYM}_3: \quad \mathsf{exp}(y,y) = \mathsf{exp}(y,-y) \quad (6)$$

Note that, e.g., (3) results from the term $\mathsf{exp}(-y,-y)$, which is relevant (see Def. 6) even though it does not occur in φ.

To show soundness of our refinement, we have to show that our lemmas are EIA-valid.

Lemma 9. *Let s,t be terms of sort* **Int**. Then SYM_1 – SYM_3 are EIA-valid.

4.2.2 Monotonicity Lemmas *Monotonicity lemmas* are of the form

$$s_2 \geq s_1 > 1 \wedge t_2 \geq t_1 > 0 \wedge (s_2 > s_1 \vee t_2 > t_1) \implies \mathsf{exp}(s_2,t_2) > \mathsf{exp}(s_1,t_1), \quad (\text{MON})$$

i.e., they prohibit violations of monotonicity of exp.

Example 10 (Monotonicity Lemmas). For our leading example (2), we obtain, e.g., the following lemmas:

$$x > 1 \wedge \mathsf{exp}(y,y) > y^2 > 0 \implies \mathsf{exp}(x,\mathsf{exp}(y,y)) > \mathsf{exp}(x,y^2) \quad (7)$$
$$x > 1 \wedge -\mathsf{exp}(y,y) > y^2 > 0 \implies \mathsf{exp}(x,-\mathsf{exp}(y,y)) > \mathsf{exp}(x,y^2) \quad (8)$$

So for each pair of two different relevant terms $\mathsf{exp}(s_1,t_1), \mathsf{exp}(s_2,t_2)$ where $[\![s_2]\!] \geq [\![s_1]\!] > 1$ and $[\![t_2]\!] \geq [\![t_1]\!] > 0$, the set $\mathcal{L}$ contains MON.

Lemma 11. *Let s_1, s_2, t_1, t_2 be terms of sort* **Int**. Then MON is EIA-valid.

[5] We excluded these functions from $\Sigma_{\mathbf{Int}}$, as they can be simulated with mod.

4.2.3 Bounding Lemmas *Bounding lemmas* provide bounds on relevant terms $\mathsf{exp}(s,t)$ where $[\![s]\!]$ and $[\![t]\!]$ are non-negative. Together with symmetry lemmas, they also give rise to bounds for the cases where s or t are negative.

For each relevant term $\mathsf{exp}(s,t)$ where $[\![s]\!]$ and $[\![t]\!]$ are non-negative, the following lemmas are contained in $\mathcal{L}$:

$$t = 0 \implies \mathsf{exp}(s,t) = 1 \qquad (\text{BND}_1)$$
$$t = 1 \implies \mathsf{exp}(s,t) = s \qquad (\text{BND}_2)$$
$$s = 0 \wedge t \neq 0 \iff \mathsf{exp}(s,t) = 0 \qquad (\text{BND}_3)$$
$$s = 1 \implies \mathsf{exp}(s,t) = 1 \qquad (\text{BND}_4)$$
$$s + t > 4 \wedge s > 1 \wedge t > 1 \implies \mathsf{exp}(s,t) > s \cdot t + 1 \qquad (\text{BND}_5)$$

The cases $t \in \{0,1\}$ are also addressed by our first rewrite rule (see Sect. 4.1). However, this rewrite rule only applies if t is an integer constant. In contrast, the first two lemmas above apply if t evaluates to 0 or 1 in the current model.

Example 12 (Bounding Lemmas). For our leading example (2), the following bounding lemmas would be considered, among others:

$$\text{BND}_1: \quad \mathsf{exp}(y,y) = 0 \implies \mathsf{exp}(x,\mathsf{exp}(y,y)) = 1$$
$$\text{BND}_2: \quad \mathsf{exp}(y,y) = 1 \implies \mathsf{exp}(x,\mathsf{exp}(y,y)) = x$$
$$\text{BND}_3: \quad x = 0 \wedge \mathsf{exp}(y,y) \neq 0 \iff \mathsf{exp}(x,\mathsf{exp}(y,y)) = 0$$
$$\text{BND}_4: \quad x = 1 \implies \mathsf{exp}(x,\mathsf{exp}(y,y)) = 1$$
$$\text{BND}_5: \quad y > 2 \implies \mathsf{exp}(y,y) > y^2 + 1 \qquad (9)$$
$$\text{BND}_5: \quad -y > 2 \implies \mathsf{exp}(-y,-y) > y^2 + 1 \qquad (10)$$

Lemma 13. *Let s,t be terms of sort* **Int**. Then BND_1 – BND_5 are EIA-valid.

The bounding lemmas are defined in such a way that they provide lower bounds for $\mathsf{exp}(s,t)$ for almost all non-negative values of s and t. The reason why we focus on lower bounds is that polynomials can only bound $\mathsf{exp}(s,t)$ from above for finitely many values of s and t. The missing (lower and upper) bounds are provided by *interpolation lemmas*.

4.2.4 Interpolation Lemmas In addition to bounding lemmas, we use *interpolation lemmas* that are constructed via *bilinear interpolation* to provide bounds. Here, we assume that the arguments of exp are positive, as negative arguments are handled by symmetry lemmas, and bounding lemmas yield tight bounds if at least one argument of exp is 0. The correctness of interpolation lemmas relies on the following observation.

Lemma 14. *Let $f : \mathbb{R}_+ \to \mathbb{R}_+$ be convex, $w_1, w_2 \in \mathbb{R}_+$, and $w_1 < w_2$. Then*

$$\forall x \in [w_1, w_2].\ f(x) \leq f(w_1) + \frac{f(w_2) - f(w_1)}{w_2 - w_1} \cdot (x - w_1) \qquad \textit{and}$$
$$\forall x \in \mathbb{R}_+ \setminus (w_1, w_2).\ f(x) \geq f(w_1) + \frac{f(w_2) - f(w_1)}{w_2 - w_1} \cdot (x - w_1).$$

Here, $[w_1, w_2]$ and (w_1, w_2) denote closed and open real intervals. Note that the right-hand side of the inequations above is the linear interpolant of f between

w_1 and w_2. Intuitively, it corresponds to the secant of f between the points $(w_1, f(w_1))$ and $(w_2, f(w_2))$, and thus the lemma follows from convexity of f.

Let $\mathsf{exp}(s,t)$ be relevant, $[\![s]\!] = c > 0$, $[\![t]\!] = d > 0$, and $[\![\mathsf{exp}]\!](c,d) \neq c^d$, i.e., we want to prohibit the current interpretation of $\mathsf{exp}(s,t)$.

Interpolation Lemmas for Upper Bounds First assume $[\![\mathsf{exp}]\!](c,d) > c^d$, i.e., to rule out this counterexample, we need a lemma that provides a suitable upper bound for $\mathsf{exp}(c,d)$. Let $c', d' \in \mathbb{N}_+$ and:

$$c^- := \min(c,c') \qquad c^+ := \max(c,c') \qquad d^- := \min(d,d') \qquad d^+ := \max(d,d')$$
$$[c^\pm] := [c^- \mathinner{..} c^+] \qquad [d^\pm] := [d^- \mathinner{..} d^+]$$

Here, $[a \mathinner{..} b]$ denotes a closed integer interval. Then we first use d^-, d^+ for linear interpolation w.r.t. the 2^{nd} argument of $\lambda x, y.\ x^y$. To this end, let

$$\mathrm{ip}_2^{[d^\pm]}(x,y) := x^{d^-} + \frac{x^{d^+} - x^{d^-}}{d^+ - d^-} \cdot (y - d^-),$$

where we define $\frac{a}{b} := \frac{a}{b}$ if $b \neq 0$ and $\frac{a}{0} := 0$. So if $d^- < d^+$, then $\mathrm{ip}_2^{[d^\pm]}(x,y)$ corresponds to the linear interpolant of x^y w.r.t. y between d^- and d^+. Then $\mathrm{ip}_2^{[d^\pm]}(x,y)$ is a suitable upper bound, as

$$\forall x \in \mathbb{N}_+, y \in [d^\pm].\ x^y \leq \mathrm{ip}_2^{[d^\pm]}(x,y) \tag{11}$$

follows from Lemma 14. Hence, we could derive the following EIA-valid lemma:[6]

$$s > 0 \wedge t \in [d^\pm] \implies \mathsf{exp}(s,t) \leq \mathrm{ip}_2^{[d^\pm]}(s,t) \tag{IP$_1$}$$

Example 15 (Linear Interpolation w.r.t. y). Let $[\![\mathsf{exp}(s,t)]\!] = [\![\mathsf{exp}]\!](3,9) > 3^9$, i.e., we have $c = 3$ and $d = 9$. Moreover, assume $c' = d' = 1$, i.e., we get $c^- = 1$, $c^+ = 3$, $d^- = 1$, and $d^+ = 9$. Then

$$\mathrm{ip}_2^{[d^\pm]}(x,y) = \mathrm{ip}_2^{[1..9]}(x,y) = x^1 + \frac{x^9 - x^1}{9-1} \cdot (y-1) = x + \frac{x^9 - x}{8} \cdot (y-1).$$

Hence, IP$_1$ corresponds to

$$s > 0 \wedge t \in [1,9] \implies \mathsf{exp}(s,t) \leq s + \frac{s^9 - s}{8} \cdot (t-1).$$

This lemma would be violated by our counterexample, as we have

$$\left[\!\!\left[s + \frac{s^9 - s}{8} \cdot (t-1) \right]\!\!\right] = 3 + \frac{3^9 - 3}{8} \cdot 8 = 3^9 < [\![\mathsf{exp}]\!](3,9) = [\![\mathsf{exp}(s,t)]\!].$$

However, the degree of $\mathrm{ip}_2^{[d^\pm]}(s,t)$ depends on d^+, which in turn depends on the model that was found by the underlying SMT solver. Thus, the degree of $\mathrm{ip}_2^{[d^\pm]}(s,t)$ can get very large, which is challenging for the underlying solver.

So we next use c^-, c^+ for linear interpolation w.r.t. the 1^{st} argument of $\lambda x, y.\ x^y$, resulting in

[6] Strictly speaking, this lemma is not a $\Sigma^{\mathsf{exp}}_{\mathbf{Int}}$-term if $d^+ > d^-$, as the right-hand side makes use of division in this case. However, an equivalent $\Sigma^{\mathsf{exp}}_{\mathbf{Int}}$-term can clearly be obtained by multiplying with the divisor.

$$\mathrm{ip}_1^{[c^\pm]}(x,y) := (c^-)^y + \frac{(c^+)^y - (c^-)^y}{c^+ - c^-} \cdot (x - c^-).$$

Then due to Lemma 14, $\mathrm{ip}_1^{[c^\pm]}(x,y)$ is also an upper bound on the exponentiation function, i.e., we have

$$\forall y \in \mathbb{N}_+, x \in [c^\pm].\ x^y \leq \mathrm{ip}_1^{[c^\pm]}(x,y). \tag{12}$$

Note that we have $\frac{y-d^-}{d^+-d^-} \in [0,1]$ for all $y \in [d^\pm]$, and thus

$$\mathrm{ip}_2^{[d^\pm]}(x,y) = x^{d^-} \cdot \left(1 - \frac{y-d^-}{d^+-d^-}\right) + x^{d^+} \cdot \frac{y-d^-}{d^+-d^-}$$

is monotonically increasing in both x^{d^-} and x^{d^+}. Hence, in the definition of $\mathrm{ip}_2^{[d^\pm]}$, we can approximate x^{d^-} and x^{d^+} with their upper bounds $\mathrm{ip}_1^{[c^\pm]}(x,d^-)$ and $\mathrm{ip}_1^{[c^\pm]}(x,d^+)$ that can be derived from (12). Then (11) yields

$$\forall x \in [c^\pm], y \in [d^\pm].\ x^y \leq \mathrm{ip}^{[c^\pm][d^\pm]}(x,y) \tag{13}$$

where

$$\mathrm{ip}^{[c^\pm][d^\pm]}(x,y) := \mathrm{ip}_1^{[c^\pm]}(x,d^-) + \frac{\mathrm{ip}_1^{[c^\pm]}(x,d^+) - \mathrm{ip}_1^{[c^\pm]}(x,d^-)}{d^+ - d^-} \cdot (y - d^-).$$

So the set $\mathcal{L}$ contains the lemma

$$s \in [c^\pm] \wedge t \in [d^\pm] \implies \mathsf{exp}(s,t) \leq \mathrm{ip}^{[c^\pm][d^\pm]}(s,t), \tag{IP$_2$}$$

which is valid due to (13), and rules out any counterexample with $[\![\mathsf{exp}]\!](c,d) > c^d$, as $\mathrm{ip}^{[c^\pm][d^\pm]}(c,d) = c^d$.

Example 16 (Bilinear Interpolation, Ex. 15 continued). In our example, we have:

$$\mathrm{ip}_1^{[c^\pm]}(x,y) = \mathrm{ip}_1^{[1..3]}(x,y) = 1^y + \frac{3^y - 1^y}{3-1} \cdot (x-1) = 1 + \frac{3^y - 1}{2} \cdot (x-1)$$
$$\mathrm{ip}_1^{[c^\pm]}(s,d^-) = \mathrm{ip}_1^{[1..3]}(s,1) = 1 + \frac{3-1}{2} \cdot (s-1) = s$$
$$\mathrm{ip}_1^{[c^\pm]}(s,d^+) = \mathrm{ip}_1^{[1..3]}(s,9) = 1 + \frac{3^9 - 1}{2} \cdot (s-1) = 1 + 9841 \cdot (s-1)$$

Hence, we obtain the lemma

$$s \in [1,3] \wedge t \in [1,9] \implies \mathsf{exp}(s,t) \leq s + \frac{1 + 9841 \cdot (s-1) - s}{8} \cdot (t-1).$$

This lemma is violated by our counterexample, as we have

$$\left[\!\!\left[s + \frac{1 + 9841 \cdot (s-1) - s}{8} \cdot (t-1) \right]\!\!\right] = 3^9 < [\![\mathsf{exp}]\!](3,9) = [\![\mathsf{exp}(s,t)]\!].$$

IP$_2$ relates $\mathsf{exp}(s,t)$ with the *bilinear* function $\mathrm{ip}^{[c^\pm][d^\pm]}(s,t)$, i.e., this function is linear w.r.t. both s and t, but it multiplies s and t. Thus, if s is an integer constant and t is linear, then the resulting lemma is linear, too.

To compute interpolation lemmas, a second point (c',d') is needed. In our implementation, we store all points (c,d) where interpolation has previously been applied and use the one which is closest to the current one. The same heuristic is

used to compute *secant lemmas* in [11]. For the 1^{st} interpolation step, we use $(c', d') = (c, d)$. In this case, IP$_2$ simplifies to $s = c \wedge t = d \implies \mathsf{exp}(s,t) \leq c^d$.

Lemma 17. *Let $c^+ \geq c^- > 0$ and $d^+ \geq d^- > 0$. Then* IP$_2$ *is EIA-valid.*

Interpolation Lemmas for Lower Bounds While bounding lemmas already yield lower bounds, the bounds provided by BND$_5$ are not exact, in general. Hence, if $[\![\mathsf{exp}]\!](c,d) < c^d$, then we also use bilinear interpolation to obtain a precise lower bound for $\mathsf{exp}(c,d)$. Dually to (11) and (12), Lemma 14 implies:

$$\forall x, y \in \mathbb{N}_+.\ x^y \geq \mathrm{ip}_2^{[d..d+1]}(x,y) \quad (14) \qquad \forall x, y \in \mathbb{N}_+.\ x^y \geq \mathrm{ip}_1^{[c..c+1]}(x,y) \quad (15)$$

Additionally, we also obtain

$$\forall x, y \in \mathbb{N}_+.\ x^{y+1} - x^y \geq \mathrm{ip}_1^{[c..c+1]}(x, y+1) - \mathrm{ip}_1^{[c..c+1]}(x,y) \quad (16)$$

from Lemma 14. The reason is that for $f(x) := x^{y+1} - x^y$, the right-hand side of (16) is equal to the linear interpolant of f between c and $c+1$. Moreover, f is convex, as $f(x) = x^y \cdot (x-1)$ where for any fixed $y \in \mathbb{N}_+$, both x^y and $x-1$ are non-negative, monotonically increasing, and convex on $\mathbb{R}_+$.

If $y \geq d$, then $\mathrm{ip}_2^{[d..d+1]}(x,y) = x^d + (x^{d+1} - x^d) \cdot (y-d)$ is monotonically increasing in the first occurrence of x^d, and in $x^{d+1} - x^d$. Thus, by approximating x^d and $x^{d+1} - x^d$ with their lower bounds from (15) and (16), (14) yields

$$\begin{aligned}\forall x \in \mathbb{N}_+, y \geq d.\ x^y &\geq \mathrm{ip}_1^{[c..c+1]}(x,d) + (\mathrm{ip}_1^{[c..c+1]}(x,d+1) - \mathrm{ip}_1^{[c..c+1]}(x,d)) \cdot (y-d) \\ &= \mathrm{ip}^{[c..c+1][d..d+1]}(x,y). \quad (17)\end{aligned}$$

So dually to IP$_2$, the set $\mathcal{L}$ contains the lemma

$$s \geq 1 \wedge t \geq d \implies \mathsf{exp}(s,t) \geq \mathrm{ip}^{[c..c+1][d..d+1]}(s,t) \quad (\mathrm{IP}_3)$$

which is valid due to (17) and rules out any counterexample with $[\![\mathsf{exp}]\!](c,d) < c^d$, as $\mathrm{ip}^{[c..c+1][d..d+1]}(c,d) = c^d$.

Example 18 (Interpolation, Lower Bounds). Let $[\![\mathsf{exp}(s,t)]\!] = [\![\mathsf{exp}]\!](3,9) < 3^9$, i.e., we have $c = 3$, and $d = 9$. Then

$$\begin{aligned}\mathrm{ip}_1^{[3..4]}(x,9) &= 3^9 + (4^9 - 3^9) \cdot (x-3) = 19683 + 242461 \cdot (x-3) \\ \mathrm{ip}_1^{[3..4]}(x,10) &= 3^{10} + (4^{10} - 3^{10}) \cdot (x-3) = 59049 + 989527 \cdot (x-3) \\ \mathrm{ip}^{[3..4][9..10]}(x,y) &= \mathrm{ip}_1^{[3..4]}(x,9) + (\mathrm{ip}_1^{[3..4]}(x,10) - \mathrm{ip}_1^{[3..4]}(x,9)) \cdot (y-9)\end{aligned}$$

and thus we obtain the lemma

$$s \geq 1 \wedge t \geq 9 \implies \mathsf{exp}(s,t) \geq 747066 \cdot s \cdot t - 6481133 \cdot s - 2201832 \cdot t + 19108788.$$

It is violated by our counterexample, as we have

$$[\![747066 \cdot s \cdot t - 6481133 \cdot s - 2201832 \cdot t + 19108788]\!] = 3^9 > [\![\mathsf{exp}]\!](3,9).$$

Lemma 19. *Let $c, d \in \mathbb{N}_+$. Then* IP$_3$ *is EIA-valid.*

4.2.5 Lazy Lemma Generation In practice, it is not necessary to compute the entire set of lemmas $\mathcal{L}$. Instead, we can stop as soon as $\mathcal{L}$ contains a single lemma which is violated by the current counterexample. However, such a strategy

would result in a quite fragile implementation, as its behavior would heavily depend on the order in which lemmas are computed, which in turn depends on low-level details like the order of iteration over sets, etc. So instead, we improve Lines 9 – 11 of Alg. 1 and use the following precedence on our four kinds of lemmas:

$$\text{symmetry} \succ \text{monotonicity} \succ \text{bounding} \succ \text{interpolation}$$

Then we compute all lemmas of the same kind, starting with symmetry lemmas, and we only proceed with the next kind if none of the lemmas computed so far is violated by the current counterexample. The motivation for the order above is as follows: Symmetry lemmas obtain the highest precedence, as other kinds of lemmas depend on them for restricting $\mathsf{exp}(s,t)$ in the case that s or t is negative. As the coefficients in interpolation lemmas for $\mathsf{exp}(s,t)$ grow exponentially w.r.t. $[\![t]\!]$ (see, e.g., Ex. 18), interpolation lemmas get the lowest precedence. Finally, we prefer monotonicity lemmas over bounding lemmas, as monotonicity lemmas are linear (if the arguments of exp are linear), whereas BND_5 may be non-linear.

Example 20 (Leading Example Finished). We now finish our leading example which, after preprocessing, looks as follows (see Ex. 4):

$$x^2 > 4 \wedge y^2 > 4 \wedge \mathsf{exp}(x, y^2) = \mathsf{exp}(x, \mathsf{exp}(y, y)) \tag{2}$$

Then our implementation generates 12 symmetry lemmas, 4 monotonicity lemmas, and 8 bounding lemmas before proving unsatisfiability, including

$$(3), (4), (5), (6), (7), (8), (9), \text{ and } (10).$$

These lemmas suffice to prove unsatisfiability for the case $x > 2$ (the cases $x \in [-2\,..\,2]$ or $y \in [-2\,..\,2]$ are trivial). For example, if $y < -2$ and $-y \bmod 2 = 0$, we get

$$y < -2 \overset{(10)}{\curvearrowright} \mathsf{exp}(-y,-y) > y^2 + 1 \overset{(3)}{\curvearrowright} \mathsf{exp}(y,-y) > y^2 + 1$$
$$\overset{(6)}{\curvearrowright} \mathsf{exp}(y,y) > y^2 + 1 \overset{(7)}{\curvearrowright} \mathsf{exp}(x, \mathsf{exp}(y,y)) > \mathsf{exp}(x, y^2) \overset{(2)}{\curvearrowright} \textbf{false}$$

and for the cases $y > 2$ and $y < -2 \wedge -y \bmod 2 = 1$, unsatisfiability can be shown similarly. For the case $x < -2$, 5 more symmetry lemmas, 2 more monotonicity lemmas, and 3 more bounding lemmas are used. The remaining 3 symmetry lemmas and 3 bounding lemmas are not used in the final proof of unsatisfiability.

While our leading example can be solved without interpolation lemmas, in general, interpolation lemmas are a crucial ingredient of our approach.

Example 21. Consider the formula

$$1 < x < y \wedge 0 < z \wedge \mathsf{exp}(x,z) < \mathsf{exp}(y,z).$$

Our implementation first rules out 33 counterexamples using 7 bounding lemmas and 42 interpolation lemmas in $\sim$0.1 seconds, before finding the model $[\![x]\!] = 21$, $[\![y]\!] = 721$, and $[\![z]\!] = 4$. Recall that interpolation lemmas are only used if a counterexample cannot be ruled out by any other kinds of lemmas. So without interpolation lemmas, our implementation could not solve this example.

Our main soundness theorem follows from soundness of our preprocessings (Lemma 5) and the fact that all of our lemmas are EIA-valid (Lemmas 9, 11, 13, 17, and 19).

Theorem 22 (Soundness of Alg. 1). *If Alg. 1 returns* **sat**, *then φ is satisfiable in EIA. If Alg. 1 returns* **unsat**, *then φ is unsatisfiable in EIA.*

Another important property of Alg. 1 is that it can eliminate *any* counterexample, and hence it makes progress in every iteration.

Theorem 23 (Progress Theorem). *If* $\mathbf{A}$ *is a counterexample and* $\mathcal{L}$ *is computed as in Alg. 1, then*

$$\mathbf{A} \not\models \bigwedge \mathcal{L}.$$

Despite Theorems 22 and 23, EIA is of course undecidable, and hence Alg. 1 is incomplete. For example, it does not terminate for the input formula

$$y \neq 0 \wedge \mathsf{exp}(2, x) = \mathsf{exp}(3, y). \tag{18}$$

Here, to prove unsatisfiability, one needs to know that $2^{|x|}$ is 1 or even, but $3^{|y|}$ is odd and greater than 1 (unless $y = 0$). This cannot be derived from the lemmas used by our approach. Thus, Alg. 1 would refine the formula (18) infinitely often.

Note that monotonicity lemmas are important, even though they are not required to prove Thm. 23. The reason is that *all* (usually infinitely many) counterexamples must be eliminated to prove **unsat**. For instance, reconsider Ex. 20, where the monotonicity lemma (7) eliminates infinitely many counterexamples with $[\![\mathsf{exp}(x, \mathsf{exp}(y, y))]\!] \leq [\![\mathsf{exp}(x, y^2)]\!]$. In contrast, Thm. 23 only guarantees that every single counterexample can be eliminated. Consequently, our implementation does not terminate on our leading example if monotonicity lemmas are disabled.

5 Related Work

The most closely related work applies *incremental linearization* to NIA, or to non-linear real arithmetic with transcendental functions (NRAT). Like our approach, incremental linearization is an instance of the CEGAR paradigm: An initial abstraction (where certain predefined functions are considered as uninterpreted functions) is refined via linear lemmas that rule out the current counterexample.

Our approach is inspired by, but differs significantly from the approach for linearization of NRAT from [11]. There, non-linear polynomials are linearized as well, whereas we leave the handling of polynomials to the backend solver. Moreover, [11] uses linear lemmas only, whereas we also use bilinear lemmas. Furthermore, [11] fixes the base to Euler's number e, whereas we consider a binary version of exponentiation.

The only lemmas that easily carry over from [11] are monotonicity lemmas. While [11] also uses symmetry lemmas, they express properties of the sine function, i.e., they are fundamentally different from ours. Our bounding lemmas are related to the "lower bound" and "zero" lemmas from [11], but there, $\lambda x.\ e^x$ is trivially bounded by 0. Interpolation lemmas are related to the "tangent" and

"secant lemmas" from [11]. However, tangent lemmas make use of first derivatives, so they are not expressible with integer arithmetic in our setting, as we have $\frac{\partial}{\partial y} x^y = x^y \cdot \ln x$. Secant lemmas are essentially obtained by linear interpolation, so our interpolation lemmas can be seen as a generalization of secant lemmas to binary functions. A preprocessing by rewriting is not considered in [11].

In [10], incremental linearization is applied to NIA. The lemmas that are used in [10] are similar to those from [11], so they differ fundamentally from ours, too.

Further existing approaches for NRAT are based on interval propagation [14,24]. As observed in [11], interval propagation effectively computes a piecewise *constant* approximation, which is less expressive than our bilinear approximations.

Recently, a novel approach for NRAT based on the *topological degree test* has been proposed [12,30]. Its strength is finding irrational solutions more often than other approaches for NRAT. Hence, this line of work is orthogonal to ours.

EIA could also be tackled by combining NRAT techniques with branch-and-bound, but the following example shows that doing so is not promising.

Example 24. Consider the formula $x = \mathsf{exp}(3, y) \wedge y > 0$. To tackle it with existing solvers, we have to encode it using the natural exponential function:

$$e^z = 3 \wedge x = e^{y \cdot z} \wedge y > 0 \tag{19}$$

Here x and y range over the integers and z ranges over the reals. Any model of (19) satisfies $z = \ln 3$, where $\ln 3$ is irrational. As finding such models is challenging, the leading tools MathSat [9] and CVC5 [2] fail for $e^z = 3$.

MetiTarski [1] integrates decision procedures for real closed fields and approximations for transcendental functions into the theorem prover Metis [27] to prove theorems about the reals. In a related line of work, iSAT3 [14] has been coupled with SPASS [35]. Clearly, these approaches differ fundamentally from ours.

Recently, the complexity of a decidable extension of linear integer arithmetic with exponentiation has been investigated [5]. It is equivalent to EIA without the functions "$\cdot$", "div", and "mod", and where the first argument of all occurrences of exp must be the same constant. Integrating decision procedures for fragments like this one into our approach is an interesting direction for future work.

6 Implementation and Evaluation

Implementation We implemented our approach in our novel tool SwInE. It is based on SMT-Switch [31], a library that offers a unified interface for various SMT solvers. SwInE uses the backend solvers Z3 4.12.2 [32] and CVC5 1.0.8 [2]. It supports incrementality and can compute models for variables, but not yet for uninterpreted functions, due to limitations inherited from SMT-Switch.

The backend solver (which defaults to Z3) can be selected via command-line flags. For more information on SwInE and a precompiled release, we refer to [22,23].

Benchmarks To evaluate our approach, we synthesized a large collection of EIA problems from verification benchmarks for safety, termination, and complexity

analysis. More precisely, we ran our verification tool LoAT [18] on the benchmarks for *linear Constrained Horn Clauses (CHCs)* with *linear integer arithmetic* from the *CHC Competitions* 2022 and 2023 [8] as well as on the benchmarks for *Termination* and *Complexity of Integer Transition Systems* from the *Termination Problems Database (TPDB)* [34], the benchmark set of the *Termination and Complexity Competition* [25], and extracted all SMT problems with exponentiation that LoAT created while analyzing these benchmarks. Afterwards, we removed duplicates.

The resulting benchmark set consists of 4627 SMT problems, which are available at [22]:

- 669 problems that resulted from the benchmarks of the CHC Competition '22 (called *CHC Comp '22 Problems* below)
- 158 problems that resulted from the benchmarks of the CHC Competition '23 (*CHC Comp '23 Problems*)
- 3146 problems that resulted from the complexity benchmarks of the TPDB (*Complexity Problems*)
- 654 problems that resulted from the termination benchmarks of the TPDB (*Termination Problems*)

Evaluation We ran SwInE with both supported backend solvers (Z3 and CVC5). To evaluate the impact of the different components of our approach, we also tested with configurations where we disabled rewriting, symmetry lemmas, bounding lemmas, interpolation lemmas, or monotonicity lemmas. All experiments were performed on StarExec [33] with a wall clock timeout of 10s and a memory limit of 128GB per example. We chose a small timeout, as LoAT usually has to discharge many SMT problems to solve a single verification task. So in our setting, each individual SMT problem should be solved quickly.

The results can be seen in Tables 1 to 4, where VB means "virtual best". All but 48 of the 4627 benchmarks can be solved, and all unsolved benchmarks are Complexity Problems. All CHC Comp Problems can be solved with both backend solvers. Considering Complexity and Termination Problems, Z3 and CVC5 perform almost equally well on unsatisfiable instances, but Z3 solves more satisfiable instances.

Regarding the different components of our approach, our evaluation shows that the impact of rewriting is quite significant. For example, it enables Z3 to solve 81 additional Complexity Problems. Symmetry lemmas enable Z3 to solve more Complexity Problems, but they are less helpful for CVC5. In fact, symmetry lemmas are needed for most of the examples where Z3 succeeds but CVC5 fails, so they seem to be challenging for CVC5, presumably due to the use of "mod". Bounding and interpolation lemmas are crucial for proving satisfiability. In particular, disabling interpolation lemmas harms more than disabling any other feature, which shows their importance. For example, Z3 can only prove satisfiability of 3 CHC Comp Problems without interpolation lemmas.

Interestingly, only CVC5 benefits from monotonicity lemmas, which enable it to solve more Complexity Problems. From our experience, CVC5 explores the search space in a more systematic way than Z3, so that subsequent candidate

Table 1: CHC Comp '22 – Results

backend	configuration	**sat**	**unsat**	**unknown**
Z3	default	296	373	0
CVC5		296	373	0
VB		296	373	0
Z3	no rewriting	291	373	5
	no symmetry	296	373	0
	no bounding	110	373	186
	no interpolation	3	372	294
	no monotonicity	296	373	0
	no rewriting, no lemmas	1	364	304
CVC5	no rewriting	296	372	1
	no symmetry	296	373	0
	no bounding	186	373	110
	no interpolation	28	372	269
	no monotonicity	296	373	0
	no rewriting, no lemmas	1	364	304

Fig. 1: CHC Comp '22 – Runtime

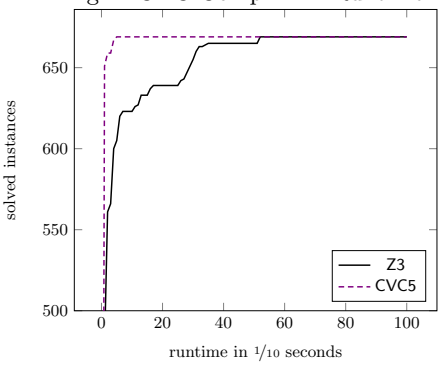


Table 2: CHC Comp '23 – Results

backend	configuration	**sat**	**unsat**	**unknown**
Z3	default	87	71	0
CVC5		87	71	0
VB		87	71	0
Z3	no rewriting	86	71	1
	no symmetry	87	71	0
	no bounding	79	71	8
	no interpolation	0	71	87
	no monotonicity	87	71	0
	no rewriting, no lemmas	0	61	97
CVC5	no rewriting	87	71	0
	no symmetry	87	71	0
	no bounding	79	71	8
	no interpolation	36	71	51
	no monotonicity	87	71	0
	no rewriting, no lemmas	0	61	97

Fig. 2: CHC Comp '23 – Runtime

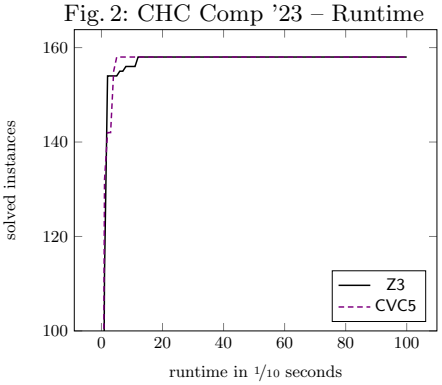


Table 3: Complexity – Results

backend	configuration	**sat**	**unsat**	**unknown**
Z3	default	1282	1789	75
CVC5		990	1784	372
VB		1309	1789	48
Z3	no rewriting	1201	1789	156
	no symmetry	975	1789	382
	no bounding	674	1788	684
	no interpolation	586	1787	773
	no monotonicity	1284	1789	73
	no rewriting, no lemmas	30	1733	1383
CVC5	no rewriting	900	1784	462
	no symmetry	954	1784	408
	no bounding	181	1784	1181
	no interpolation	405	1782	959
	no monotonicity	795	1784	567
	no rewriting, no lemmas	30	1728	1388

Fig. 3: Complexity – Runtime

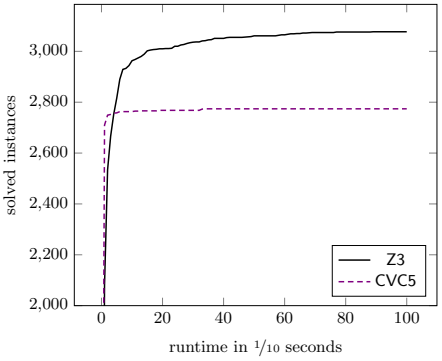
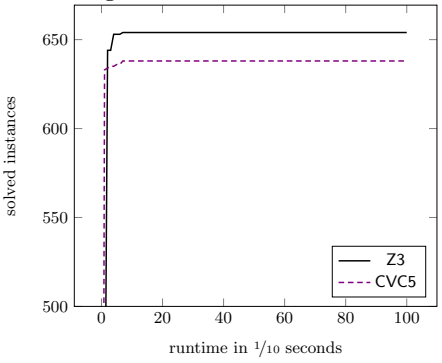


Table 4: Termination – Results

backend	configuration	**sat**	**unsat**	**unknown**
Z3	default	223	431	0
CVC5		208	430	16
VB		223	431	0
Z3	no rewriting	223	431	0
	no symmetry	223	431	0
	no bounding	177	429	48
	no interpolation	15	429	210
	no monotonicity	223	431	0
	no rewriting, no lemmas	7	428	219
CVC5	no rewriting	208	430	16
	no symmetry	208	430	16
	no bounding	171	428	55
	no interpolation	10	428	216
	no monotonicity	208	430	16
	no rewriting, no lemmas	7	428	219

Fig. 4: Termination – Runtime



models often have a similar structure. Then monotonicity lemmas can help CVC5 to find structurally different candidate models.

Remarkably, disabling a single component does not reduce the number of **unsat**'s significantly. Thus, we also evaluated configurations where *all* components were disabled, so that exp is just an uninterpreted function. This reduces the number of **sat** results dramatically, but most **unsat** instances can still be solved. Hence, most of them do not require reasoning about exponentials, so it would be interesting to obtain instances where proving **unsat** is more challenging.

The runtime of SwInE can be seen in Figures 1 to 4. Most instances can be solved in a fraction of a second, as desired for our use case. Moreover, CVC5 can solve more instances in the first half second, but Z3 can solve more instances later on. We refer to [22] for more details on our evaluation.

Validation We implemented sanity checks for both **sat** and **unsat** results. For **sat**, we evaluate the input problem using EIA semantics for exp, and the current model for all variables. For **unsat**, assume that the input problem φ contains the subterms $\mathsf{exp}(s_0, t_0), \ldots, \mathsf{exp}(s_n, t_n)$. Then we enumerate all SMT problems

$$\varphi \wedge \bigwedge_{i=0}^{n} t_i = c_i \wedge \mathsf{exp}(s_i, t_i) = s_i^{c_i} \qquad \text{where } c_1, \ldots, c_n \in [0\,..\,k] \text{ for some } k \in \mathbb{N}$$

(we used $k = 10$). If any of them is satisfiable in NIA, then φ is satisfiable in EIA. None of these checks revealed any problems.

7 Conclusion

We presented the novel SMT theory EIA, which extends the theory *non-linear integer arithmetic* with integer exponentiation. Moreover, inspired by *incremental linearization* for similar extensions of *non-linear real arithmetic*, we developed a CEGAR approach to solve EIA problems. The core idea of our approach is to regard exponentiation as an uninterpreted function and to eliminate counterexamples, i.e., models that violate the semantics of exponentiation, by generating suitable *lemmas*. Here, the use of *bilinear interpolation* turned out to be crucial, both in practice (see our evaluation in Sect. 6) and in theory, as interpolation lemmas are essential for being able to eliminate *any* counterexample (see Thm. 23). Finally, we evaluated the implementation of our approach in our novel tool SwInE on thousands of EIA problems that were synthesized from verification tasks using our verification tool LoAT. Our evaluation shows that SwInE is highly effective for our use case, i.e., as backend for LoAT. Hence, we will couple SwInE and LoAT in future work.

With SwInE, we provide an SMT-LIB compliant open-source solver for EIA [23]. In this way, we hope to attract users with applications that give rise to challenging benchmarks, as our evaluation suggests that our benchmarks are relatively easy to solve. Moreover, we hope that other solvers with support for integer exponentiation will follow, with the ultimate goal of standardizing EIA.

References

1. Akbarpour, B., Paulson, L.C.: MetiTarski: An automatic theorem prover for real-valued special functions. J. Autom. Reason. **44**(3), 175–205 (2010). https://doi.org/10.1007/S10817-009-9149-2
2. Barbosa, H., Barrett, C.W., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y.: CVC5: A versatile and industrial-strength SMT solver. In: TACAS '22. pp. 415–442. LNCS 13243 (2022). https://doi.org/10.1007/978-3-030-99524-9_24
3. Bardin, S., Finkel, A., Leroux, J., Petrucci, L.: FAST: Acceleration from theory to practice. Int. J. Softw. Tools Technol. Transf. **10**(5), 401–424 (2008). https://doi.org/10.1007/s10009-008-0064-3
4. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org` (2016)
5. Benedikt, M., Chistikov, D., Mansutti, A.: The complexity of Presburger arithmetic with power or powers. In: ICALP '23. pp. 112:1–112:18. LIPIcs 261 (2023). https://doi.org/10.4230/LIPIcs.ICALP.2023.112
6. Bozga, M., Iosif, R., Konečný, F.: Fast acceleration of ultimately periodic relations. In: CAV '10. pp. 227–242. LNCS 6174 (2010). https://doi.org/10.1007/978-3-642-14295-6_23
7. Bozga, M., Iosif, R., Konečný, F.: Relational analysis of integer programs. Tech. Rep. TR-2012-10, VERIMAG (2012), https://www-verimag.imag.fr/TR/TR-2012-10.pdf
8. CHC Competition, https://chc-comp.github.io
9. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The MathSAT5 SMT solver. In: TACAS '13. pp. 93–107. LNCS 7795 (2013). https://doi.org/10.1007/978-3-642-36742-7_7
10. Cimatti, A., Griggio, A., Irfan, A., Roveri, M., Sebastiani, R.: Experimenting on solving nonlinear integer arithmetic with incremental linearization. In: SAT '18. pp. 383–398. LNCS 10929 (2018). https://doi.org/10.1007/978-3-319-94144-8_23
11. Cimatti, A., Griggio, A., Irfan, A., Roveri, M., Sebastiani, R.: Incremental linearization for satisfiability and verification modulo nonlinear arithmetic and transcendental functions. ACM Trans. Comput. Log. **19**(3), 19:1–19:52 (2018). https://doi.org/10.1145/3230639
12. Cimatti, A., Griggio, A., Lipparini, E., Sebastiani, R.: Handling polynomial and transcendental functions in SMT via unconstrained optimisation and topological degree test. In: ATVA '22. pp. 137–153. LNCS 13505 (2022). https://doi.org/10.1007/978-3-031-19992-9_9
13. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: CAV '00. pp. 154–169. LNCS 1855, Springer (2000). https://doi.org/10.1007/10722167_15
14. Fränzle, M., Herde, C., Teige, T., Ratschan, S., Schubert, T.: Efficient solving of large non-linear arithmetic constraint systems with complex Boolean structure. J. Satisf. Boolean Model. Comput. **1**(3-4), 209–236 (2007). https://doi.org/10.3233/sat190012
15. Frohn, F., Giesl, J.: Proving non-termination via loop acceleration. In: FMCAD '19. pp. 221–230 (2019). https://doi.org/10.23919/FMCAD.2019.8894271
16. Frohn, F., Giesl, J.: Termination of triangular integer loops is decidable. In: CAV '19. pp. 426–444. LNCS 11562 (2019). https://doi.org/10.1007/978-3-030-25543-5_24

17. Frohn, F., Naaf, M., Brockschmidt, M., Giesl, J.: Inferring lower runtime bounds for integer programs. ACM Trans. Program. Lang. Syst. **42**(3), 13:1–13:50 (2020). https://doi.org/10.1145/3410331
18. Frohn, F., Giesl, J.: Proving non-termination and lower runtime bounds with LoAT. In: IJCAR '22. pp. 712–722. LNCS 13385 (2022). https://doi.org/10.1007/978-3-031-10769-6_41
19. Frohn, F., Giesl, J.: Proving non-termination by Acceleration Driven Clause Learning. In: CADE '23. LNCS 14132 (2023). https://doi.org/10.1007/978-3-031-38499-8_13
20. Frohn, F., Giesl, J.: ADCL: Acceleration Driven Clause Learning for constrained Horn clauses. In: SAS '23. pp. 259–285. LNCS 14284 (2023). https://doi.org/10.1007/978-3-031-44245-2_13
21. Frohn, F., Giesl, J.: Satisfiability modulo exponential integer arithmetic. CoRR **abs/2402.01501** (2024). https://doi.org/10.48550/arXiv.2402.01501
22. Frohn, F., Giesl, J.: Evaluation of "Satisfiability modulo exponential integer arithmetic" (2024), https://aprove-developers.github.io/swine-eval/
23. Frohn, F., Giesl, J.: SwInE (2024), https://ffrohn.github.io/swine/
24. Gao, S., Avigad, J., Clarke, E.M.: δ-complete decision procedures for satisfiability over the reals. In: IJCAR '12. pp. 286–300. LNCS 7364 (2012). https://doi.org/10.1007/978-3-642-31365-3_23
25. Giesl, J., Rubio, A., Sternagel, C., Waldmann, J., Yamada, A.: The termination and complexity competition. In: TACAS '19. pp. 156–166. LNCS 11429 (2019). https://doi.org/10.1007/978-3-030-17502-3_10
26. Hark, M., Frohn, F., Giesl, J.: Termination of triangular polynomial loops. Form. Methods Syst. Des. (2023). https://doi.org/10.1007/s10703-023-00440-z
27. Hurd, J.: First-order proof tactics in higher-order logic theorem provers. In: STRATA '03. pp. 56–68. Report NASA/CP-2003-212448 (2003), https://apps.dtic.mil/sti/pdfs/ADA418902.pdf
28. Jeannet, B., Schrammel, P., Sankaranarayanan, S.: Abstract acceleration of general linear loops. In: POPL '14. pp. 529–540 (2014). https://doi.org/10.1145/2535838.2535843
29. Kroening, D., Lewis, M., Weissenbacher, G.: Under-approximating loops in C programs for fast counterexample detection. Formal Methods Syst. Des. **47**(1), 75–92 (2015). https://doi.org/10.1007/s10703-015-0228-1
30. Lipparini, E., Ratschan, S.: Satisfiability of non-linear transcendental arithmetic as a certificate search problem. In: NFM '23. pp. 472–488. LNCS 13903 (2023). https://doi.org/10.1007/978-3-031-33170-1_29
31. Mann, M., Wilson, A., Zohar, Y., Stuntz, L., Irfan, A., Brown, K., Donovick, C., Guman, A., Tinelli, C., Barrett, C.W.: SMT-Switch: A solver-agnostic C++ API for SMT solving. In: SAT '21. pp. 377–386. LNCS 12831 (2021). https://doi.org/10.1007/978-3-030-80223-3_26
32. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: TACAS '08. pp. 337–340. LNCS 4963 (2008). https://doi.org/10.1007/978-3-540-78800-3_24
33. Stump, A., Sutcliffe, G., Tinelli, C.: StarExec: A cross-community infrastructure for logic solving. In: IJCAR '14. pp. 367–373. LNCS 8562 (2014). https://doi.org/10.1007/978-3-319-08587-6_28
34. Termination Problems Data Base (TPDB), http://termination-portal.org/wiki/TPDB
35. Weidenbach, C., Dimova, D., Fietzke, A., Kumar, R., Suda, M., Wischnewski, P.: SPASS version 3.5. In: CADE '22. pp. 140–145. LNCS 5663 (2009). https://doi.org/10.1007/978-3-642-02959-2_10