Journal of Automated Reasoning manuscript No.
(will be inserted by the editor)

Satisfiability Modulo Exponential Integer Arithmetic

Florian Frohn · Jürgen Giesl

the date of receipt and acceptance should be inserted later

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.

We implemented our approach and evaluate it empirically on several sets of benchmarks from different domains: Most of them stem from program verification, and one set stems from recurrence solving. On all sets, 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 needed, 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,

RWTH Aachen University, Aachen, Germany
Programming Languages and Verification
E-mail: florian.frohn@cs.rwth-aachen.de, giesl@informatik.rwth-aachen.de

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

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

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

$$\exists n \in \mathbb{N}.\ x = 3^n \land y = 0. \tag{1.1}$$

In recent work, *acceleration techniques* have successfully been used to deduce lower runtime bounds automatically [19, 20]. While they can easily derive a formula like (1.1) 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 [17, 20, 22] and (un)safety [4, 8, 9, 21, 33, 35], 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 [18, 31]. 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 by Cimatti et al. [13], including the natural exponential function $\mathsf{exp}_e(x) = e^x$, where e is Euler's number. 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 *monotonicity lemmas*, none of the lemmas of Cimatti et al. easily carry over to our setting. In contrast to Cimatti et al., 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 Cimatti et al. fix 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

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

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. 4.26). Therefore, the name "incremental linearization" does not fit to our approach, which is rather an instance of "counterexample-guided abstraction refinement" (CEGAR, [15]).

A preliminary shorter version of this paper appeared at IJCAR 2024 [24]. The current paper contains the following, yet unpublished contributions:

– We present two new kinds of lemmas, called *prime lemmas* and *induction lemmas*.
– We present a new technique called *phasing*, which alternates between a **sat**-*phase* where the solver searches for a model within a restricted part of the search space, and an **unsat**-*phase* where the solver aims for proving unsatisfiability.
– While we did not give any proofs in [24], we now present proofs for all lemmas and theorems.
– We present a reimplementation of SwInE, called SwInE-Z3 in the sequel, which directly builds on top of the API of the SMT solver Z3 [39]. In contrast, the original implementation of our approach from [24] was built on top of SMT-Switch [37], a library that offers a unified interface for various SMT solvers. However, our evaluation in [24] showed that Z3 performs best in our setting, so that using its API directly allows for improving robustness.[2]
– We present a novel collection of challenging unsatisfiable benchmarks, which were obtained from verifying solutions for recurrence relations.
– We improve and extend our evaluation by including our novel benchmarks and the tools SwInE-Z3 and Z3. In order to use Z3 for integer exponentiation, we axiomatize the semantics of exponentiation via universally quantified assertions.

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). Next, we show how to improve our approach via phasing (Sect. 5). After discussing related work (Sect. 6), we describe two open-source implementations of our approach, one on top of SMT-Switch [27], and one on top of Z3 [28], and we present an experimental evaluation on several collections of EIA benchmarks that we synthesized from verification problems and from recurrence solving. Our experiments show that our approach is highly effective in practice (Sect. 7). We refer to App. A for those (parts of the) proofs that were omitted from the main part of the paper.

2 Preliminaries

We are working in the setting of *SMT-LIB logic* [5], a variant of many-sorted first-order logic with equality. We now introduce a reduced variant of SMT-LIB logic, 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 with results of sort **Bool**.

[2] For example, SwInE uses the parser of SMT-Switch, which strictly adheres to the SMT-LIB standard. In contrast, Z3 (and thus SwInE-Z3) can often also parse inputs that violate the SMT-LIB standard in a way that does not introduce ambiguity.

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 $\mathbf{true}, 0, 1, \ldots$, 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 $\mathbf{Int}$ and the following function symbols:

$$\begin{aligned} c &: \mathbf{Int} && \text{for all } c \in \mathbb{N} \\ +, -, \cdot, \mathsf{div}, \mathsf{mod} &: \mathbf{Int}\ \mathbf{Int}\ \mathbf{Int} \\ <, \leq, >, \geq, =, \neq &: \mathbf{Int}\ \mathbf{Int}\ \mathbf{Bool} \\ \mathsf{divisible}_c &: \mathbf{Int}\ \mathbf{Bool} && \text{for all } c \in \mathbb{N}_{>0} = \{n \in \mathbb{N} \mid n > 0\} \end{aligned}$$

Then the $\Sigma_{\mathbf{Int}}$-theory *non-linear integer arithmetic* (NIA)[3] contains all $\Sigma_{\mathbf{Int}}$-structures $\mathbf{A}$ where $A_{\mathbf{Int}} = \mathbb{Z}$,

$$[\![\mathsf{divisible}_c]\!]^{\mathbf{A}}(x) = \begin{cases} \mathbf{true} & \text{if } x \mathbin{\mathsf{mod}} c = 0 \\ \mathbf{false} & \text{otherwise,} \end{cases}$$

and all other symbols from $\Sigma_{\mathbf{Int}}$ are interpreted as usual.[4]

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 theory *linear integer arithmetic* (LIA) consists of the reducts of all elements of NIA to $\Sigma_{\mathrm{LIA}} = \Sigma_{\mathbf{Int}} \setminus \{\cdot, \mathsf{div}, \mathsf{mod}\}$.[5]

[3] As we only consider quantifier-free formulas, we omit the prefix "QF_" in theory names and write, e.g., NIA instead of QF_NIA. Barrett et al. [5] call QF_NIA 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.

[4] Note that all functions are total in SMT-LIB logic. So div and mod are interpreted as integer division and modulo, respectively, if their second argument is non-zero. If the second argument is zero, then the interpretation of div and mod is arbitrary, i.e., $\lambda x.\ x \mathbin{\mathsf{div}} 0$ and $\lambda x.\ x \mathbin{\mathsf{mod}} 0$ are treated like uninterpreted functions.

[5] Note that Σ_{LIA} indeed contains the predicates $\mathsf{divisible}_c$, as LIA would not admit quantifier elimination without them. To see this, consider the formula $\varphi := \exists y.\ x = y + y$. It is equivalent to $\mathsf{divisible}_2(x)$, but there is no quantifier-free LIA formula without divisibility predicates which is equivalent to φ.

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 sort **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. As in SMT-LIB, we prefix theory names with "UF" to indicate that the signature may contain uninterpreted functions. So for example, UFNIA denotes the theory of non-linear integer arithmetic with uninterpreted functions.

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 write binary function symbols in infix notation.

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 $\Sigma_{\mathbf{Int}}^{\mathsf{exp}}$, 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 [6].

Definition 3.1 (EIA) The theory *exponential integer arithmetic (EIA)* contains all $\Sigma_{\mathbf{Int}}^{\mathsf{exp}}$-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 Thm. 3.1.

Algorithm 1: CEGAR for EIA

```
   Input: a $\Sigma^{\mathsf{exp}}_{\mathbf{Int}}$-formula $\varphi$
   // Preprocessing
1  do
2      $\varphi' \leftarrow \varphi$
3      $\varphi \leftarrow$ FoldConstants($\varphi$)
4      $\varphi \leftarrow$ Rewrite($\varphi$)
5  while $\varphi \neq \varphi'$
   // Refinement Loop
6  $Kinds \leftarrow \{Symmetry, Monotonicity, Bounding, Prime, Induction, Interpolation\}$
7  while there is an UFNIA-model $\mathbf{A}$ of $\varphi$
8      if $\mathbf{A}$ is a counterexample then
9          $\mathcal{L} \leftarrow \varnothing$
10         for $k \in Kinds$
11             $\mathcal{L} \leftarrow \mathcal{L} \cup$ ComputeLemmas($\varphi, k$)
12         $\varphi \leftarrow \varphi \wedge \bigwedge\{\psi \in \mathcal{L} \mid \mathbf{A} \not\models \psi\}$
13     else return sat
14 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 7 is a CEGAR loop which lifts an SMT solver for UFNIA (which is called in Line 7) to EIA. So the *abstraction* consists of using UFNIA- instead of EIA-models. Hence, exp is considered to be an uninterpreted function in Line 7, 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 10 – 12 and the loop is iterated again.

Definition 4.1 (Counterexample) We call an UFNIA-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 – 12) in Sect. 4.2. Here, we first introduce the different kinds of lemmas that are used by our implementation in Sect. 4.2.1 – 4.2.6. If implemented naively, the number of lemmas can get quite large, so we explain how to generate lemmas *lazily* in Sect. 4.3. Finally, we conclude this section by stating important properties of Alg. 1.

Example 4.2 (Leading Example) To illustrate our approach, we show how to prove

$$\forall x, y.\ |x| > 2 \wedge |y| > 2 \implies \mathsf{exp}(\mathsf{exp}(x,y),y) \neq \mathsf{exp}(x,\mathsf{exp}(y,y)). \qquad (4.1)$$

Here, we also consider negative values for x and y (i.e., we use the premise $|x| > 2 \wedge |y| > 2$ instead of $x > 2 \wedge y > 2$) to illustrate how our techniques works when exp is used with negative arguments. To prove (4.1), we encode absolute values suitably[6] and prove unsatisfiability of its negation:

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

[6] 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 1st rule allows us to rewrite[7] $\mathsf{exp}(s, 0)$ 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.

Example 4.3 (Preprocessing) For our leading example, applying the 2nd rewrite rule at the underlined position yields:

$$\begin{aligned}
& x^2 > 4 \land y^2 > 4 \land \underline{\mathsf{exp}(\mathsf{exp}(x, y), y)} = \mathsf{exp}(x, \mathsf{exp}(y, y)) \\
\to\ & x^2 > 4 \land y^2 > 4 \land \mathsf{exp}(x, y^2) = \mathsf{exp}(x, \mathsf{exp}(y, y))
\end{aligned} \tag{4.2}$$

Lemma 4.4 *We have*

$$\varphi \equiv_{\text{EIA}} \textsc{FoldConstants}(\varphi) \qquad \textit{and} \qquad \varphi \equiv_{\text{EIA}} \textsc{Rewrite}(\varphi).$$

Proof For constant folding, the claim is trivial, so we only have to show $\varphi \equiv_{\text{EIA}} \textsc{Rewrite}(\varphi)$. To this end, it suffices to show $[\![\ell]\!]^{\mathbf{A}} = [\![r]\!]^{\mathbf{A}}$ for all rewrite rules $\ell \to r$ and all $\mathbf{A} \in \text{EIA}$. Let $\mathbf{A} \in \text{EIA}$ be arbitrary but fixed and let $[\![\ldots]\!]$ denote $[\![\ldots]\!]^{\mathbf{A}}$.

For the first rewrite rule $\mathsf{exp}(x, c) \to x^{|c|}$, recall that $x^{|c|}$ is syntactic sugar for $\overbrace{x \cdot \ldots \cdot x}^{|c| \text{ times}}$, i.e., we have to show $[\![\mathsf{exp}(x, c)]\!] = [\![\overbrace{x \cdot \ldots \cdot x}^{|c| \text{ times}}]\!]$. We obtain

$$[\![\mathsf{exp}(x, c)]\!] = [\![x]\!]^{|[\![c]\!]|} = [\![x]\!]^{|c|} = \overbrace{[\![x]\!] \cdot \ldots \cdot [\![x]\!]}^{|c| \text{ times}} = [\![\overbrace{x \cdot \ldots \cdot x}^{|c| \text{ times}}]\!].$$

The proofs for the remaining rewrite rules are analogous. See App. A for the complete proofs.

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

4.2 Refinement

Our refinement (Lines 9 – 12 of Alg. 1) is based on the six kinds of lemmas named in Line 6: *symmetry lemmas*, *monotonicity lemmas*, *bounding lemmas*, *prime lemmas*, *induction 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 12. As our lemmas allow us to eliminate *any* counterexample, this set is never empty, see Thm. 4.26. To compute $\mathcal{L}$, we consider all terms that are *relevant* for the formula φ.

Definition 4.5 (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 4.6 (Relevant Terms) For our leading example (4.2), the relevant terms are all terms of the form

$$\mathsf{exp}(\pm x, \pm y^2), \qquad \mathsf{exp}(\pm y, \pm y), \quad \text{or} \quad \mathsf{exp}(\pm x, \pm \mathsf{exp}(y,y)).$$

While the formula φ is changed in Line 12 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 constant and t is linear for all subterms $\mathsf{exp}(s,t)$ of φ.

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:

$$\mathsf{divisible}_2(t) \implies \mathsf{exp}(s,t) = \mathsf{exp}(-s,t) \tag{SYM$_1$}$$
$$\neg\mathsf{divisible}_2(t) \implies \mathsf{exp}(s,t) = -\mathsf{exp}(-s,t) \tag{SYM$_2$}$$
$$\mathsf{exp}(s,t) = \mathsf{exp}(s,-t) \tag{SYM$_3$}$$

Note that SYM$_1$ and SYM$_2$ are just implications, not equivalences, as, for example, $c^{|d|} = (-c)^{|d|}$ does not imply $\mathsf{divisible}_2(d)$ if $c = 0$.

Example 4.7 (Symmetry Lemmas) For our leading example (4.2), the following symmetry lemmas would be considered, among others:

$$\text{SYM}_1: \quad \mathsf{divisible}_2(-y) \implies \mathsf{exp}(-y,-y) = \mathsf{exp}(y,-y) \tag{4.3}$$
$$\text{SYM}_2: \quad \neg\mathsf{divisible}_2(-y) \implies \mathsf{exp}(-y,-y) = -\mathsf{exp}(y,-y) \tag{4.4}$$
$$\text{SYM}_3: \quad \mathsf{exp}(x,\mathsf{exp}(y,y)) = \mathsf{exp}(x,-\mathsf{exp}(y,y)) \tag{4.5}$$
$$\text{SYM}_3: \quad \mathsf{exp}(y,y) = \mathsf{exp}(y,-y) \tag{4.6}$$

Note that, e.g., (4.3) results from the term $\mathsf{exp}(-y,-y)$, which is relevant (see Thm. 4.5) even though it does not occur in φ.

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

Lemma 4.8 *Let s, t be terms of sort* **Int**. Then $\textsc{sym}_1 - \textsc{sym}_3$ are EIA-valid.

Proof Let $\mathbf{A} \in$ EIA again be arbitrary but fixed and let $[\![\ldots]\!]$ denote $[\![\ldots]\!]^{\mathbf{A}}$. For $\textsc{sym}_1$, assume $[\![\mathsf{divisible}_2(t)]\!] = \mathbf{true}$, i.e., assume that $[\![t]\!]$ is even. Then it remains to show

$$[\![\mathsf{exp}(s,t) = \mathsf{exp}(-s,t)]\!] = \mathbf{true}, \quad \text{i.e.,} \quad [\![\mathsf{exp}(s,t)]\!] = [\![\mathsf{exp}(-s,t)]\!]\,.$$

We have:

$$\begin{aligned} [\![\mathsf{exp}(s,t)]\!] &= [\![s]\!]^{|[\![t]\!]|} \\ &= [\![-s]\!]^{|[\![t]\!]|} && \text{(as } [\![t]\!]\text{, and thus also } |\,[\![t]\!]\,|\text{, is even)} \\ &= [\![\mathsf{exp}(-s,t)]\!] \end{aligned}$$

The proofs for $\textsc{sym}_2$ and $\textsc{sym}_3$ are analogous, see App. A.

4.2.2 Monotonicity Lemmas

Monotonicity lemmas are of the form

$$s_2 \geq s_1 > 1 \land t_2 \geq t_1 > 0 \land (s_2 > s_1 \lor t_2 > t_1) \implies \mathsf{exp}(s_2,t_2) > \mathsf{exp}(s_1,t_1), \qquad (\textsc{mon})$$

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

Example 4.9 (Monotonicity Lemmas) For our leading example (4.2), we obtain, e.g., the following lemmas:

$$x > 1 \land \mathsf{exp}(y,y) > y^2 > 0 \implies \mathsf{exp}(x,\mathsf{exp}(y,y)) > \mathsf{exp}(x,y^2) \qquad (4.7)$$

$$x > 1 \land -\mathsf{exp}(y,y) > y^2 > 0 \implies \mathsf{exp}(x,-\mathsf{exp}(y,y)) > \mathsf{exp}(x,y^2) \qquad (4.8)$$

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 $\textsc{mon}$. Here and in the sequel, unless mentioned otherwise, $[\![\ldots]\!]$ means $[\![\ldots]\!]^{\mathbf{A}}$, where $\mathbf{A}$ is the model from Line 7 of Alg. 1.

Lemma 4.10 *Let s_1, s_2, t_1, t_2 be terms of sort* **Int**. Then $\textsc{mon}$ is EIA-valid.

Proof Let $\mathbf{A} \in$ EIA be arbitrary but fixed and let $[\![\ldots]\!]$ denote $[\![\ldots]\!]^{\mathbf{A}}$. Assume

$$[\![s_2 \geq s_1 > 1 \land t_2 \geq t_1 > 0 \land (s_2 > s_1 \lor t_2 > t_1)]\!] = \mathbf{true},$$

i.e.,

$$[\![s_2]\!] \geq [\![s_1]\!] > 1, [\![t_2]\!] \geq [\![t_1]\!] > 0, \text{ and } ([\![s_2]\!] > [\![s_1]\!] \text{ or } [\![t_2]\!] > [\![t_1]\!]).$$

Then it remains to prove

$$[\![\mathsf{exp}(s_2,t_2) > \mathsf{exp}(s_1,t_1)]\!] = \mathbf{true}, \quad \text{i.e.,} \quad [\![\mathsf{exp}(s_2,t_2)]\!] > [\![\mathsf{exp}(s_1,t_1)]\!]\,.$$

First note that $\lambda x.\ x^{[\![t_1]\!]}$ is strictly monotonically increasing on $\mathbb{N}_{>1}$ as $[\![t_1]\!] > 0$, and $\lambda y.\ [\![s_1]\!]^y$ is strictly monotonically increasing on $\mathbb{N}_{>0}$ as $[\![s_1]\!] > 1$. Since we have $[\![s_2]\!] \in \mathbb{N}_{>1}$ and $[\![t_2]\!] \in \mathbb{N}_{>0}$, we get

$$[\![\mathsf{exp}(s_2,t_2)]\!] = [\![s_2]\!]^{|[\![t_2]\!]|} > [\![s_1]\!]^{|[\![t_1]\!]|} = [\![\mathsf{exp}(s_1,t_1)]\!]$$

due to monotonicity, as we have $[\![s_2]\!] \geq [\![s_1]\!]$ and $[\![t_2]\!] \geq [\![t_1]\!]$, where at least one of both inequations is strict.

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 $\mathcal{L}$:

$$t = 0 \implies \mathsf{exp}(s,t) = 1 \qquad (\text{BND}_1)$$

$$t = 1 \implies \mathsf{exp}(s,t) = s \qquad (\text{BND}_2)$$

$$s = 0 \land t \neq 0 \iff \mathsf{exp}(s,t) = 0 \qquad (\text{BND}_3)$$

$$s = 1 \implies \mathsf{exp}(s,t) = 1 \qquad (\text{BND}_4)$$

$$s + t > 4 \land s > 1 \land t > 1 \implies \mathsf{exp}(s,t) > s \cdot t + 1 \qquad (\text{BND}_5)$$

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 4.11 (Bounding Lemmas) For our leading example (4.2), the following bounding lemmas would be considered, among others:

$$\text{BND}_1: \quad \mathsf{exp}(y,y) = 0 \implies \mathsf{exp}(x,\mathsf{exp}(y,y)) = 1$$

$$\text{BND}_2: \quad \mathsf{exp}(y,y) = 1 \implies \mathsf{exp}(x,\mathsf{exp}(y,y)) = x$$

$$\text{BND}_3: \quad x = 0 \land \mathsf{exp}(y,y) \neq 0 \iff \mathsf{exp}(x,\mathsf{exp}(y,y)) = 0$$

$$\text{BND}_4: \quad x = 1 \implies \mathsf{exp}(x,\mathsf{exp}(y,y)) = 1$$

$$\text{BND}_5: \quad y > 2 \implies \mathsf{exp}(y,y) > y^2 + 1 \qquad (4.9)$$

$$\text{BND}_5: \quad -y > 2 \implies \mathsf{exp}(-y,-y) > y^2 + 1 \qquad (4.10)$$

Lemma 4.12 *Let s, t be terms of sort* **Int**. Then $\text{BND}_1 - \text{BND}_5$ are EIA-valid.

Proof We only prove that BND_5 is EIA-valid and refer to App. A for the (simpler) proofs of $\text{BND}_1 - \text{BND}_4$.

Let $\mathbf{A} \in \text{EIA}$ again be arbitrary but fixed and let $[\![\ldots]\!]$ denote $[\![\ldots]\!]^{\mathbf{A}}$. To show validity of BND_5, assume

$$[\![s + t > 4 \land s > 1 \land t > 1]\!] = \mathbf{true}, \quad \text{i.e,} \quad [\![s]\!] + [\![t]\!] > 4,\ [\![s]\!] > 1, \text{ and } [\![t]\!] > 1.$$

Then it remains to show

$$[\![\mathsf{exp}(s,t) > s \cdot t + 1]\!] = \mathbf{true}, \quad \text{i.e.,} \quad [\![\mathsf{exp}(s,t)]\!] > [\![s]\!] \cdot [\![t]\!] + 1.$$

We use induction on $[\![s]\!] + [\![t]\!]$. In the base case, we have either $[\![s]\!] = 2$ and $[\![t]\!] = 3$, or $[\![s]\!] = 3$ and $[\![t]\!] = 2$. In the former case, we have:

$$[\![\mathsf{exp}(s,t)]\!] = [\![s]\!]^{|[\![t]\!]|} = 2^{|3|} = 8 > 7 = 2 \cdot 3 + 1 = [\![s]\!] \cdot [\![t]\!] + 1$$

In the latter case, we have:

$$[\![\mathsf{exp}(s,t)]\!] = [\![s]\!]^{|[\![t]\!]|} = 3^{|2|} = 9 > 7 = 3 \cdot 2 + 1 = [\![s]\!] \cdot [\![t]\!] + 1$$

For the induction step, assume $[\![s]\!] + [\![t]\!] > 5$. If $[\![t]\!] > 2$, then:

$$\begin{aligned}
[\![\mathsf{exp}(s,t)]\!] &= [\![s]\!]^{|[\![t]\!]|} \\
&= [\![s]\!]^{[\![t]\!]} \\
&= [\![s]\!]^{[\![t]\!]-1} \cdot [\![s]\!] \\
&> ([\![s]\!] \cdot ([\![t]\!] - 1) + 1) \cdot [\![s]\!] && \text{(by the induction hypothesis)} \\
&\geq ([\![s]\!] \cdot ([\![t]\!] - 1) + 1) \cdot 2 && \text{(as } [\![s]\!] \geq 2\text{)} \\
&= ([\![s]\!] \cdot ([\![t]\!] - 1) + 1) + ([\![s]\!] \cdot ([\![t]\!] - 1) + 1) \\
&\geq ([\![s]\!] \cdot ([\![t]\!] - 1) + 1) + ([\![s]\!] \cdot 2 + 1) && \text{(as } [\![t]\!] > 2\text{)} \\
&> ([\![s]\!] \cdot ([\![t]\!] - 1) + 1) + [\![s]\!] \\
&= [\![s]\!] \cdot [\![t]\!] + 1
\end{aligned}$$

If $[\![t]\!] = 2$, then $[\![s]\!] > 3$. Now we have

$$\begin{aligned}
[\![\mathsf{exp}(s,t)]\!] &= [\![s]\!]^{|[\![t]\!]|} \\
&= [\![s]\!]^2 \\
&= ([\![s-1]\!] + 1)^2 \\
&= [\![s-1]\!]^2 + 2 \cdot [\![s-1]\!] + 1 \\
&> [\![s-1]\!] \cdot 2 + 1 + 2 \cdot [\![s-1]\!] + 1 && \text{(by the induction hypothesis)} \\
&\geq [\![s-1]\!] \cdot 2 + 8 && \text{(as } [\![s]\!] > 3 \text{ and thus, } [\![s-1]\!] \geq 3\text{)} \\
&= [\![s]\!] \cdot 2 + 6 \\
&= [\![s]\!] \cdot [\![t]\!] + 6 \\
&> [\![s]\!] \cdot [\![t]\!] + 1
\end{aligned}$$

The reason why the bounding lemmas 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 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: BND1–BND4 provide tight bounds for all cases where at least one argument of exp is 0 or 1, and BND5 provides lower bounds for all other cases except for $\mathsf{exp}(2,2)$. To cover *all* cases, we could replace BND5 by

$$s > 1 \wedge t \geq 0 \implies \mathsf{exp}(s,t) \geq s \cdot t.$$

However, in contrast to the alternative lemma above, BND5 expresses that $\mathsf{exp}(s,t)$ is *strictly* greater than $s \cdot t$. The missing (lower and upper) bounds are provided by *interpolation lemmas* (see Sect. 4.2.6).

4.2.4 Prime Lemmas

Prime lemmas are of the form

$$\mathsf{divisible}_d(\mathsf{exp}(s,t)) \iff \mathsf{divisible}_d(s) \wedge t \neq 0 \qquad \text{where } d \text{ is prime.} \qquad \text{(PRIME)}$$

So prime lemmas express that $\mathsf{exp}(s,t)$ and s have the same prime factors for $t \neq 0$.

Example 4.13 (Prime Lemmas) Consider the formula

$$y \neq 0 \land \mathsf{exp}(2, x) = \mathsf{exp}(3, y), \tag{4.11}$$

which we used in [24] (where prime lemmas were not yet integrated) to illustrate incompleteness of our approach. Unsatisfiability of this formula can easily be proven with the following two prime lemmas, which rule out structures $\mathbf{A}$ where $\mathbf{A} \models x \neq 0$:

$$\begin{aligned} &\mathsf{divisible}_2(\mathsf{exp}(2, x)) \iff \mathsf{divisible}_2(2) \land x \neq 0 \\ \equiv\ &\mathsf{divisible}_2(\mathsf{exp}(2, x)) \iff x \neq 0 \end{aligned}$$

$$\begin{aligned} &\mathsf{divisible}_2(\mathsf{exp}(3, y)) \iff \mathsf{divisible}_2(3) \land y \neq 0 \\ \equiv\ &\neg\mathsf{divisible}_2(\mathsf{exp}(3, y)) \end{aligned}$$

The reason is that $x \neq 0$ implies $\mathsf{divisible}_2(\mathsf{exp}(2, x))$ by the first prime lemma, which, together with the second prime lemma, implies $\mathsf{exp}(2, x) \neq \mathsf{exp}(3, y)$. For the case $x = 0$, the following additional lemmas are needed:

$$\begin{aligned} x = 0 &\implies \mathsf{exp}(2, x) = 1 && (\text{BND}_1) \\ y = 1 &\implies \mathsf{exp}(3, y) = 3 && (\text{BND}_2) \\ y > 1 &\implies \mathsf{exp}(3, y) > 3 \cdot y + 1 && (\text{BND}_5) \\ & \mathsf{exp}(3, y) = \mathsf{exp}(3, -y) && (\text{SYM}_3) \end{aligned}$$

Then we have $\mathsf{exp}(2, x) = 1$ by the first bounding lemma and $\mathsf{exp}(3, y) > 1$ for all $y > 0$ by the second and third bounding lemma. Together with the symmetry lemma, the latter also implies $\mathsf{exp}(3, y) > 1$ for all $y < 0$. As (4.11) implies $y \neq 0$, this suffices to prove unsatisfiability.

So for each relevant term $\mathsf{exp}(s, t)$ where $[\![s]\!] \geq 2$, $[\![\mathsf{exp}(s, t)]\!] \geq 2$, and where the sets of prime factors of $[\![s]\!]$ and $[\![\mathsf{exp}(s, t)]\!]$ differ, $\mathcal{L}$ contains PRIME, where d is the smallest prime which divides $[\![s]\!]$ or $[\![\mathsf{exp}(s, t)]\!]$, but not both of them. To find d, we use wheel factorization [41].

Lemma 4.14 *Let s, t be terms of sort* **Int** and let $d \in \mathbb{N}$ be prime. Then PRIME is EIA-valid.

Proof Again, let $\mathbf{A} \in$ EIA be arbitrary but fixed and let $[\![\ldots]\!]$ denote $[\![\ldots]\!]^{\mathbf{A}}$. We need to prove

$$[\![\mathsf{divisible}_d(\mathsf{exp}(s, t))]\!] = [\![\mathsf{divisible}_d(s) \ \land\ t \neq 0]\!],$$

i.e.,

$$[\![\mathsf{divisible}_d(\mathsf{exp}(s, t))]\!] \quad \text{iff} \quad [\![\mathsf{divisible}_d(s)]\!] \ \text{and} \ [\![t]\!] \neq 0.$$

If $[\![t]\!] = 0$, then the claim holds since then $[\![s]\!]^{|[\![t]\!]|} = [\![s]\!]^0 = 1$ is not divisible by the prime number d. Otherwise, if $[\![t]\!] \neq 0$, then we have to prove that $[\![s]\!]$ and $[\![s]\!]^{|[\![t]\!]|}$ have the same prime factors. If P is the multiset of $[\![s]\!]$'s prime factors, i.e., $\prod P = [\![s]\!]$, then $\prod_{i=1}^{|[\![t]\!]|} \prod P = [\![s]\!]^{|[\![t]\!]|}$, so the claim follows.

4.2.5 Induction Lemmas

Induction lemmas are of the form

$$s_1 = s_2 \wedge t_2 - d = t_1 \geq 0 \implies \mathsf{exp}(s_2, t_2) = \mathsf{exp}(s_1, t_1) \cdot s_1^d \qquad \text{where } d \in \mathbb{N}_{>0}. \quad (\textsc{ind})$$

So intuitively, IND results from $\mathsf{exp}(s_2, t_2)$ by d unrollings of the recursive definition

$$\mathsf{exp}(s, x) = \begin{cases} \mathsf{exp}(s, x-1) \cdot s & \text{if } x > 0 \\ 1 & \text{if } x = 0 \end{cases}$$

as follows:

$$\begin{aligned} \mathsf{exp}(s_2, t_2) &= \mathsf{exp}(s_2, t_2 - 1) \cdot s_2 \\ &= \mathsf{exp}(s_2, t_2 - 2) \cdot s_2^2 \\ &= \ldots \\ &= \mathsf{exp}(s_2, t_2 - d) \cdot s_2^d \\ &= \mathsf{exp}(s_1, t_1) \cdot s_1^d \end{aligned}$$

Example 4.15 (Induction Lemmas) We demonstrate the usefulness of induction lemmas by an example where SMT solving is used to verify a solution for a recurrence relation. Consider the following recurrence relation (where n ranges over $\mathbb{N}_{>0}$):

$$f(n) = 2 \cdot f(n-1) + 2^n \qquad (4.12)$$

A possible solution (i.e., a function f that satisfies (4.12)) is

$$f(n) = (f(0) + n) \cdot 2^n. \qquad (4.13)$$

To verify that this function is indeed a solution, it suffices to replace f with the definition from (4.13) and show that the resulting equation holds for all $n \in \mathbb{N}_{>0}$ and all $f(0) \in \mathbb{Z}$, i.e., to prove

$$\forall n \in \mathbb{N}_{>0}, f_0 \in \mathbb{Z}.\ (f_0 + n) \cdot 2^n = 2 \cdot (f_0 + n - 1) \cdot 2^{n-1} + 2^n.$$

To do so, it suffices to prove unsatisfiability of

$$n \geq 1 \wedge (f_0 + n) \cdot \mathsf{exp}(2, n) \neq 2 \cdot (f_0 + n - 1) \cdot \mathsf{exp}(2, n-1) + \mathsf{exp}(2, n). \qquad (4.14)$$

To this end, our implementation deduces the following induction lemma:

$$n \geq 1 \implies \mathsf{exp}(2, n) = 2 \cdot \mathsf{exp}(2, n-1)$$

Then unsatisfiability can easily be proven without reasoning about exp: Assume $n \geq 1$. Then by the induction lemma, we may substitute $\mathsf{exp}(2, n)$ with $2 \cdot \mathsf{exp}(2, n-1)$, resulting in

$$\begin{aligned} &(f_0 + n) \cdot 2 \cdot \mathsf{exp}(2, n-1) \neq 2 \cdot (f_0 + n - 1) \cdot \mathsf{exp}(2, n-1) + 2 \cdot \mathsf{exp}(2, n-1) \\ \equiv\ &(f_0 + n) \cdot 2 \cdot \mathsf{exp}(2, n-1) \neq 2 \cdot (f_0 + n) \cdot \mathsf{exp}(2, n-1) \end{aligned}$$

which is trivially unsatisfiable. Hence, (4.13) is a valid solution for (4.12).

So for each pair of relevant terms $\mathsf{exp}(s_1, t_1), \mathsf{exp}(s_2, t_2)$ where $[\![s_1]\!] = [\![s_2]\!]$ and $[\![t_2]\!] - d = [\![t_1]\!] \geq 0$ for some $d \in \mathbb{N}_{>0}$, $\mathcal{L}$ contains IND.

Lemma 4.16 *Let s_1, s_2, t_1, t_2 be terms of sort* **Int** and let $d \in \mathbb{N}_{>0}$. Then IND is EIA-valid.

Proof Let $\mathbf{A} \in$ EIA be an arbitrary but fixed model of

$$s_1 = s_2 \wedge t_2 - d = t_1 \geq 0$$

and let $[\![\ldots]\!]$ denote $[\![\ldots]\!]^{\mathbf{A}}$. Then we have:

$$\begin{aligned}
\left[\!\left[\mathsf{exp}(s_1, t_1) \cdot s_1^d\right]\!\right] &= [\![\mathsf{exp}(s_1, t_1)]\!] \cdot [\![s_1]\!]^d \\
&= [\![s_1]\!]^{|[\![t_1]\!]|} \cdot [\![s_1]\!]^d \\
&= [\![s_1]\!]^{|[\![t_1]\!]+d|} && \text{(as } \mathbf{A} \models t_1 \geq 0 \text{ and } d \in \mathbb{N}_{>0}) \\
&= [\![s_1]\!]^{|[\![t_2]\!]|} && \text{(as } \mathbf{A} \models t_2 - d = t_1) \\
&= [\![s_2]\!]^{|[\![t_2]\!]|} && \text{(as } \mathbf{A} \models s_1 = s_2) \\
&= [\![\mathsf{exp}(s_2, t_2)]\!]
\end{aligned}$$

4.2.6 Interpolation Lemmas

To provide bounds in cases where no bounding lemmas are violated, we use *interpolation lemmas* that are constructed via *bilinear interpolation*. 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. As usual, $[w_1, w_2]$ and (w_1, w_2) denote closed and open real intervals.

Lemma 4.17 *Let $c \geq 0$, $w_1, w_2 \in \mathbb{R}_{>c}$,*[8] *$w_1 < w_2$, and let $f : \mathbb{R}_{>c} \to \mathbb{R}_{>0}$ be convex. Then*

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

$$\forall x \in \mathbb{R}_{>c} \setminus (w_1, w_2).\ f(x) \geq f(w_1) + \frac{f(w_2) - f(w_1)}{w_2 - w_1} \cdot (x - w_1).$$

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.

Proof (Proof of Thm. 4.17) For the first inequation, recall that a function $f : \mathbb{R}_{>c} \to \mathbb{R}_{>0}$ is convex if for all $w_1, w_2 \in \mathbb{R}_{>c}$ and all $v \in [0, 1]$, we have

$$f(v \cdot w_2 + (1 - v) \cdot w_1) \leq v \cdot f(w_2) + (1 - v) \cdot f(w_1). \qquad (4.15)$$

[8] The lemma also holds for $\mathbb{R}_{\geq c}$ (and the proof is analogous).

Let $x \in [w_1, w_2]$ and $v = \frac{x-w_1}{w_2-w_1}$. Then we have $v \in [0,1]$ and

$$\begin{aligned} f(x) &= f(v \cdot w_2 + (1-v) \cdot w_1) \\ &\leq v \cdot f(w_2) + (1-v) \cdot f(w_1) && \text{(by convexity of } f \text{ and } v \in [0,1]) \\ &= f(w_1) + \frac{f(w_2) - f(w_1)}{w_2 - w_1} \cdot (x - w_1), \end{aligned}$$

as desired.

For the second inequation, note that dually to (4.15), convex functions $f : \mathbb{R}_{>c} \to \mathbb{R}_{>0}$ satisfy

$$f(v \cdot w_2 + (1-v) \cdot w_1) \geq v \cdot f(w_2) + (1-v) \cdot f(w_1)$$

for all $w_1, w_2 \in \mathbb{R}_{>c}$ and $v \notin (0,1)$ such that $v \cdot w_2 + (1-v) \cdot w_1 \in \mathbb{R}_{>c}$ [40, 46]. Let $x \in \mathbb{R}_{>c} \setminus (w_1, w_2)$ and $v = \frac{x-w_1}{w_2-w_1}$. Then we have $v \notin (0,1)$ and

$$\begin{aligned} f(x) &= f(v \cdot w_2 + (1-v) \cdot w_1) \\ &\geq v \cdot f(w_2) + (1-v) \cdot f(w_1) && \text{(by convexity of } f \text{ and } v \notin (0,1)) \\ &= f(w_1) + \frac{f(w_2) - f(w_1)}{w_2 - w_1} \cdot (x - w_1), \end{aligned}$$

as desired.

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 $c', d' \in \mathbb{N}_{>0}$ and:

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

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

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

where we define $\frac{a}{\overline{\overline{b}}} := \frac{a}{b}$ if $b \neq 0$ and $\frac{a}{\overline{\overline{0}}} := 0$. 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

$$\forall x \in \mathbb{N}_{>0}, y \in [d^\pm].\ x^y \leq \mathrm{ip}_2^{[d^\pm]}(x,y) \tag{4.16}$$

follows from Thm. 4.17: If $d^- = d^+$, then (4.16) is trivial. Otherwise, let $x \in \mathbb{N}_{>0}$ be arbitrary but fixed. Then (4.16) follows by applying Thm. 4.17 to $f(y) := x^y$, which is clearly convex on $\mathbb{R}_{>0}$.

Hence, we could derive the following EIA-valid lemma:[9]

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

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

Example 4.18 (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

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

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 1st argument of $\lambda x, y.\ x^y$, resulting in

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

Then due to Thm. 4.17, $\mathrm{ip}_1^{[c^\pm]}(x,y)$ is also an upper bound on the exponentiation function, i.e., we have

$$\forall y \in \mathbb{N}_{>0}, x \in [c^\pm].\ x^y \leq \mathrm{ip}_1^{[c^\pm]}(x,y). \tag{4.17}$$

To see this, note that (4.17) is trivial if $c^- = c^+$. Otherwise, let $y \in \mathbb{N}_{>0}$ be arbitrary but fixed. Then (4.17) follows by applying Thm. 4.17 to $f(x) := x^y$, which is clearly convex on $\mathbb{R}_{>0}$.

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 (4.17). Then (4.16) yields

$$\forall x \in [c^\pm], y \in [d^\pm].\ x^y \leq \mathrm{ip}^{[c^\pm][d^\pm]}(x,y) \tag{4.18}$$

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

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

which is valid due to (4.18), and rules out any counterexample with $[\![\mathsf{exp}]\!](c,d) > c^d$, as $\mathrm{ip}^{[c^\pm][d^\pm]}(c,d) = c^d$.

Lemma 4.19 *Let $c^+ \geq c^- > 0$ and $d^+ \geq d^- > 0$. Then* IP$_2$ *is EIA-valid.*

Example 4.20 (Bilinear Interpolation, Thm. 4.18 continued) In our example, we have:

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

Hence, we obtain the lemma

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

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 by Cimatti et al. [13] to compute *secant lemmas*. For the 1^{st} interpolation step, we choose $(c',d') = (c,d)$. In this case, IP$_2$ simplifies to $s = c \land t = d \implies \mathsf{exp}(s,t) \leq c^d$.

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 (4.16) and (4.17), Thm. 4.17 implies:

$$\forall x,y \in \mathbb{N}_{>0}.\ x^y \geq \mathrm{ip}_2^{[d..d+1]}(x,y) \tag{4.19}$$

$$\forall x,y \in \mathbb{N}_{>0}.\ x^y \geq \mathrm{ip}_1^{[c..c+1]}(x,y) \tag{4.20}$$

To see this for (4.19), let $x \in \mathbb{N}_{>0}$ be arbitrary but fixed. Then (4.19) follows by applying Thm. 4.17 to $f(y) := x^y$, which is clearly convex on $\mathbb{R}_{>0}$. While Thm. 4.17 implies the claim for $y \in \mathbb{R}_{>0} \setminus (d, d+1)$, note that $\mathbb{N}_{>0} \subseteq \mathbb{R}_{>0} \setminus (d,d+1)$, since $(d,d+1)$ does not contain any integers. The proof of (4.20) works analogously by considering $f(x) := x^y$ for fixed $y \in \mathbb{N}_{>0}$.

Additionally, we also obtain

$$\forall x,y \in \mathbb{N}_{>0}.\ x^{y+1} - x^y \geq \mathrm{ip}_1^{[c..c+1]}(x,y+1) - \mathrm{ip}_1^{[c..c+1]}(x,y) \tag{4.21}$$

from Thm. 4.17. The reason is that for $f(x) := x^{y+1} - x^y$, the right-hand side of (4.21) 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 $y \in \mathbb{N}_{>0}$, both x^y and $x-1$ are non-negative, monotonically increasing, and convex on $\mathbb{R}_{\geq 1}$. Thus, (4.21) follows by applying Thm. 4.17 to $f(x) := x^{y+1} - x^y$, since $\mathbb{N}_{>0} \subseteq \mathbb{R}_{\geq 1} \setminus (c, c+1)$.

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 (4.20) and (4.21), (4.19) yields

$$\begin{aligned}
&\forall x \in \mathbb{N}_{>0}, y \geq d. \\
&\qquad x^y \geq \mathrm{ip}_1^{[c..c+1]}(x,d) + (\mathrm{ip}_1^{[c..c+1]}(x,d+1) - \mathrm{ip}_1^{[c..c+1]}(x,d)) \cdot (y-d) \\
&\qquad\; = \mathrm{ip}^{[c..c+1][d..d+1]}(x,y). \qquad (4.22)
\end{aligned}$$

So dually to IP_2, the set $\mathcal{L}$ contains the lemma

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

which is valid due to (4.22) and rules out any counterexample with $[\![\mathsf{exp}]\!](c,d) < c^d$, as $\mathrm{ip}^{[c..c+1][d..d+1]}(c,d) = c^d$.

Lemma 4.21 *Let $c, d \in \mathbb{N}_{>0}$. Then IP_3 is EIA-valid.*

Example 4.22 (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

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

4.3 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 use the following precedence on our six kinds of lemmas:

$$\text{symmetry} \succ \text{monotonicity} \succ \text{bounding} \succ \text{prime} \approx \text{induction} \approx \text{interpolation}$$

Then we compute all lemmas of the same kind, starting with symmetry lemmas, and we only proceed with kinds of lower precedence if none of the lemmas computed so far is violated by the current counterexample. The motivation for the order above is as follows: First, we prefer symmetry, monotonicity, and bounding lemmas, as there are only finitely many of them for a finite set of relevant terms. In contrast, in principle there could be infinitely many prime, induction, and interpolation lemmas, so they all get the same, lowest precedence. The reason is that the natural numbers

Algorithm 2: CEGAR for EIA with Lazy Lemma Generation

```
Input: a $\Sigma_{\mathbf{Int}}^{\mathsf{exp}}$-formula $\varphi$
// Preprocessing
1  do
2      $\varphi' \leftarrow \varphi$
3      $\varphi \leftarrow$ FoldConstants($\varphi$)
4      $\varphi \leftarrow$ Rewrite($\varphi$)
5  while $\varphi \neq \varphi'$
// Refinement Loop
6  while there is an UFNIA-model $\mathbf{A}$ of $\varphi$
7      if $\mathbf{A}$ is a counterexample then
8          $\mathcal{L} \leftarrow \varnothing$
9          for $k \in \{\mathit{Symmetry}, \mathit{Monotonicity}, \mathit{Bounding}\}$
10             $\mathcal{L} \leftarrow \{\psi \in \text{ComputeLemmas}(\varphi, k) \mid \mathbf{A} \not\models \psi\}$
11             if $\mathcal{L} \neq \varnothing$ then break
12         if $\mathcal{L} = \varnothing$ then
13             for $k \in \{\mathit{Prime}, \mathit{Induction}, \mathit{Interpolation}\}$
14                 $\mathcal{L} \leftarrow \mathcal{L} \cup \{\psi \in \text{ComputeLemmas}(\varphi, k) \mid \mathbf{A} \not\models \psi\}$
15         $\varphi \leftarrow \varphi \wedge \bigwedge \mathcal{L}$
16     else return sat
17 return unsat
```

c, d, c^-, c^+, d^-, d^+ that occur in these lemmas depend on the current UFNIA-model **A**. 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. Moreover, 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.

The improved version of Alg. 1 is shown in Alg. 2. Note that the first inner loop in Line 9 breaks as soon as a lemma that is violated by the current model **A** has been found, i.e., as soon as $\mathcal{L}$ is non-empty. Then due to the check in Line 12, the second inner loop in Line 13 is only executed if the first inner loop failed to eliminate the current counterexample. In contrast to the first inner loop, the second one in Line 13 does not break, as prime, induction, and interpolation lemmas have the same precedence.

Example 4.23 (Leading Example Finished) We now finish our leading example which, after preprocessing, looks as follows (see Thm. 4.3):

$$x^2 > 4 \wedge y^2 > 4 \wedge \mathsf{exp}(x, y^2) = \mathsf{exp}(x, \mathsf{exp}(y, y)) \tag{4.2}$$

Then our implementation generates 12 symmetry lemmas, 4 monotonicity lemmas, and 17 bounding lemmas before proving unsatisfiability, including

$$(4.3), (4.4), (4.5), (4.6), (4.7), (4.8), (4.9), \text{ and } (4.10).$$

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 $\mathsf{divisible}_2(-y)$, we get

$$\begin{aligned} y < -2 &\overset{(4.10)}{\curvearrowright} \mathsf{exp}(-y, -y) > y^2 + 1 \overset{(4.3)}{\curvearrowright} \mathsf{exp}(y, -y) > y^2 + 1 \\ &\overset{(4.6)}{\curvearrowright} \mathsf{exp}(y, y) > y^2 + 1 \overset{(4.7)}{\curvearrowright} \mathsf{exp}(x, \mathsf{exp}(y, y)) > \mathsf{exp}(x, y^2) \overset{(4.2)}{\curvearrowright} \mathbf{false} \end{aligned}$$

and for the cases $y > 2$ and $y < -2 \land \neg\mathsf{divisible}_2(-y)$, unsatisfiability can be shown similarly. For the overall proof, 5 more symmetry lemmas, 2 more monotonicity lemmas, and 4 more bounding lemmas are used. The remaining lemmas that are deduced by our implementation 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 4.24 Consider the formula

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

Our implementation first rules out 7 counterexamples using 7 bounding lemmas, 15 prime lemmas, and 9 interpolation lemmas, before finding the model $[\![x]\!] = 5$, $[\![y]\!] = 7$, and $[\![z]\!] = 3$. Without interpolation lemmas, our implementation keeps computing prime lemmas indefinitely, and without prime and interpolation lemmas, our implementation returns **unknown**.[10]

Our main soundness theorem follows from soundness of our preprocessings (Lemma 4.4) and the fact that all of our lemmas are EIA-valid (Lemmas 4.8, 4.10, 4.12, 4.14, 4.16, 4.19, and 4.21).

Theorem 4.25 (Soundness of Alg. 2) *If Alg. 2 returns* **sat***, then φ is satisfiable in EIA. If Alg. 2 returns* **unsat***, then φ is unsatisfiable in EIA.*

Another important property of Alg. 2 is that it can eliminate *any* counterexample, and hence it makes progress in every iteration.

Theorem 4.26 (Progress Theorem) *If $\mathbf{A}$ is a counterexample and $\mathcal{L}$ is computed as in Alg. 2, then*

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

Proof Let $\mathbf{A} \in \mathrm{EIA}$ be arbitrary but fixed and let $[\![\ldots]\!]$ denote $[\![\ldots]\!]^{\mathbf{A}}$. Since $\mathbf{A}$ is a counterexample, there is a subexpression $\mathsf{exp}(s, t)$ of φ such that $[\![\mathsf{exp}(s, t)]\!] \neq [\![\mathsf{exp}(s, t)]\!]^{\mathrm{EIA}}$. We now show that $\mathbf{A}$ violates at least one of our lemmas.

First assume $[\![s]\!], [\![t]\!] \in \mathbb{N}$. Then

- $t = 0$ implies $[\![\mathrm{BND}_1]\!] = \mathbf{false}$ and
- $s = 0 \land t \neq 0$ implies $[\![\mathrm{BND}_3]\!] = \mathbf{false}$.

Hence, assume $[\![s]\!], [\![t]\!] \in \mathbb{N}_{>0}$. Then

- $[\![\mathsf{exp}(s, t)]\!] > [\![\mathsf{exp}(s, t)]\!]^{\mathrm{EIA}}$ implies $[\![\mathrm{IP}_2]\!] = \mathbf{false}$ (as remarked after IP_2), and
- $[\![\mathsf{exp}(s, t)]\!] < [\![\mathsf{exp}(s, t)]\!]^{\mathrm{EIA}}$ implies $[\![\mathrm{IP}_3]\!] = \mathbf{false}$ (as remarked after IP_3).

Now assume $[\![s]\!] < 0$ and $[\![t]\!] \geq 0$. Since $\mathsf{exp}(-s, t)$ is relevant, the set $\mathcal{L}$ contains bounding and interpolation lemmas for $\mathsf{exp}(-s, t)$. Hence, if $[\![\mathsf{exp}(-s, t)]\!] \neq [\![\mathsf{exp}(-s, t)]\!]^{\mathrm{EIA}}$, then one of these lemmas is violated by $\mathbf{A}$, as argued above. Thus, assume $[\![\mathsf{exp}(-s, t)]\!] = [\![\mathsf{exp}(-s, t)]\!]^{\mathrm{EIA}}$. Then

- if $[\![t]\!]$ is even, then $[\![\mathrm{SYM}_1]\!] = \mathbf{false}$, and

[10] Note that without interpolation lemmas, Alg. 2 may fail to eliminate a counterexample, as Thm. 4.26 does not hold without interpolation lemmas. In such cases, our implementation returns **unknown**.

– if $[\![t]\!]$ is odd, then $[\![\text{SYM}_2]\!] = \mathbf{false}$.

Next assume $[\![s]\!] \geq 0$ and $[\![t]\!] < 0$. Since $\mathsf{exp}(s,-t)$ is relevant, the set $\mathcal{L}$ contains bounding and interpolation lemmas for $\mathsf{exp}(s,-t)$. Hence, if $[\![\mathsf{exp}(s,-t)]\!] \neq [\![\mathsf{exp}(s,-t)]\!]^{\text{EIA}}$, then one of these lemmas is violated by $\mathbf{A}$, as argued above. Thus, assume $[\![\mathsf{exp}(s,-t)]\!] = [\![\mathsf{exp}(s,-t)]\!]^{\text{EIA}}$. Then $[\![\text{SYM}_3]\!] = \mathbf{false}$.

Finally, assume $[\![s_1]\!] < 0$ and $[\![t_1]\!] < 0$. Since $\mathsf{exp}(-s,-t)$ is relevant, the set $\mathcal{L}$ contains bounding and interpolation lemmas for $\mathsf{exp}(-s,-t)$. Hence, if $[\![\mathsf{exp}(-s,-t)]\!] \neq [\![\mathsf{exp}(-s,-t)]\!]^{\text{EIA}}$, then one of these lemmas is violated by $\mathbf{A}$, as argued above. Thus, assume $[\![\mathsf{exp}(-s,-t)]\!] = [\![\mathsf{exp}(-s,-t)]\!]^{\text{EIA}}$.

We only consider the case that $[\![t]\!]$ is even (the case that $[\![t]\!]$ is odd works analogously). Assume $[\![\text{SYM}_1]\!] = \mathbf{true}$ and $[\![\text{SYM}_3]\!] = \mathbf{true}$. Then we get

$$\begin{aligned}
[\![\mathsf{exp}(s,t)]\!]^{\text{EIA}} &= [\![\mathsf{exp}(-s,t)]\!]^{\text{EIA}} && \text{(as } t \text{ is even)}\\
&= [\![\mathsf{exp}(-s,-t)]\!]^{\text{EIA}} && \text{(by definition of } [\![\mathsf{exp}]\!]^{\text{EIA}})\\
&= [\![\mathsf{exp}(-s,-t)]\!] &&\\
&= [\![\mathsf{exp}(s,-t)]\!] && (\text{SYM}_1)\\
&= [\![\mathsf{exp}(s,t)]\!]\,, && (\text{SYM}_3)
\end{aligned}$$

which contradicts $[\![\mathsf{exp}(s,t)]\!] \neq [\![\mathsf{exp}(s,t)]\!]^{\text{EIA}}$. Thus, we have $[\![\text{SYM}_1]\!] = \mathbf{false}$ or $[\![\text{SYM}_3]\!] = \mathbf{false}$.

Despite Theorems 4.25 and 4.26, EIA is of course undecidable, and hence Alg. 2 is incomplete. For example, it does not terminate for the input formula

$$x \geq y \geq 0 \land \mathsf{exp}(2,x) \neq \mathsf{exp}(2,x-y) \cdot \mathsf{exp}(2,y). \tag{4.23}$$

Here, to prove unsatisfiability,[11] one would need a rewrite rule like

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

but such a rule would be unsound in our setting, as the right-hand side would need to be $\mathsf{exp}(x,|y|+|z|)$ instead. However, to avoid introducing non-linear terms like $|y|$ and $|z|$, we did not include such a rewrite rule. Thus, Alg. 2 would refine the formula (4.23) indefinitely.

Note that monotonicity, prime, and induction lemmas are important, even though they are not required to prove Thm. 4.26. The reason is that *all* (usually infinitely many) counterexamples must be eliminated to prove **unsat**. For instance, reconsider Thm. 4.23, where the monotonicity lemma (4.7) eliminates infinitely many counterexamples with $[\![\mathsf{exp}(x,\mathsf{exp}(y,y))]\!] \leq [\![\mathsf{exp}(x,y^2)]\!]$. In contrast, Thm. 4.26 only guarantees that every single counterexample can be eliminated. Consequently, our implementation does not terminate on our leading example if

[11] Note that induction lemmas are not sufficient for proving unsatisfiability of (4.23), as the difference between the exponents in induction lemmas is always a constant, whereas it is non-constant in (4.23).

Algorithm 3: CEGAR for EIA with Phasing

```
   Input: a $\Sigma_{\mathbf{Int}}^{\mathsf{exp}}$-formula $\varphi$
   // Preprocessing
1  do
2      $\varphi' \leftarrow \varphi$
3      $\varphi \leftarrow \textsc{FoldConstants}(\varphi)$
4      $\varphi \leftarrow \textsc{Rewrite}(\varphi)$
5  while $\varphi \neq \varphi'$
6  $phase \leftarrow$ sat
7  $b \leftarrow 1$
   // Refinement Loop
8  while $\top$
9      if $phase =$ sat then $\xi \leftarrow \varphi \wedge \{-2^b \leq t \leq 2^b \mid \mathsf{exp}(s,t)$ is relevant$\}$
10     else $\xi \leftarrow \varphi$
11     if there is an UFNIA-model $\mathbf{A}$ of $\xi$ then
12         if $\mathbf{A}$ is a counterexample then
13             if $phase =$ unsat then
14                 $phase \leftarrow$ sat
15                 $b$++
16             else
17                 for $k \in \{Symmetry, Monotonicity, Bounding, Prime\}$
18                     $\mathcal{L} \leftarrow \{\psi \in \textsc{ComputeLemmas}(\varphi, k) \mid \mathbf{A} \not\models \psi\}$
19                     if $\mathcal{L} \neq \varnothing$ then break
20                 if $\mathcal{L} = \varnothing$ then
21                     for $k \in \{Induction, Interpolation\}$
22                         $\mathcal{L} \leftarrow \mathcal{L} \cup \{\psi \in \textsc{ComputeLemmas}(\varphi, k) \mid \mathbf{A} \not\models \psi\}$
23                 $\varphi \leftarrow \varphi \wedge \bigwedge \mathcal{L}$
24         else return sat
25     else if $phase =$ unsat then return unsat
26     else $phase \leftarrow$ unsat
```

monotonicity lemmas are disabled. Similarly, our implementation fails to solve Thm. 4.13 and Thm. 4.15 without prime and induction lemmas, respectively.

5 Phasing

When investigating the behavior of our implementation for satisfiable examples where it failed to find a solution, we frequently observed the following problem: Even though a "small" solution exists where the exponents (i.e., the second arguments of the occurring exp-terms) have small absolute values, the underlying SMT solver returned candidate models with exponents whose absolute values were quite large. Then interpolation lemmas with huge coefficients were generated, as these coefficients grow exponentially w.r.t. the second argument of exp. Afterwards, the underlying solver took too long when searching for a new candidate model.

To counter this phenomenon, we developed *phasing*, a technique to prefer "small" over "large" solutions, see Alg. 3. The idea of phasing is to alternate between a **sat**-*phase* and an **unsat**-*phase*. In the **sat**-phase, the goal is to find small candidate models. To this end, we impose a bound on the absolute values of the exponents to block large models (Line 9). If we fail to find a candidate model in the **sat**-phase, then we switch to the **unsat**-phase (Line 26). In the **unsat**-phase, our goal is prove unsatisfiability, so we do not impose any bounds on the exponents in Line 10. If we fail to prove unsatisfiability, we discard the found model (unless it can be lifted to an EIA-model) in Line 13 and switch back to the **sat**-phase (Line 14), but with a higher bound (Line 15).

Note that the bound grows exponentially w.r.t. the number of phase changes (Lines 9 and 15). Thus, if there are no small models, then the size of the considered models grows rapidly, so that we can still find a solution quickly.

6 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 by Cimatti et al. [13]. There, non-linear polynomials are linearized as well, whereas we leave the handling of polynomials to the backend solver. Moreover, Cimatti et al. use linear lemmas only, whereas we also use bilinear lemmas. Furthermore, Cimatti et al. fix the base to Euler's number e, whereas we consider a binary version of exponentiation.

The only lemmas of Cimatti et al. [13] that easily carry over to our approach are monotonicity lemmas. While Cimatti et al. also use 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 of Cimatti et al., but there, $\lambda x.\ e^x$ is trivially bounded by 0. Interpolation lemmas are related to the "tangent" and "secant lemmas" of Cimatti et al.. 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, prime lemmas, or induction lemmas are not considered by Cimatti et al..

Cimatti et al. [12] also applied incremental linearization to NIA. Here, they used similar lemmas as for NRAT [13], so they differ again fundamentally from ours.

Further existing approaches for NRAT are based on interval propagation [16, 29]. As observed by Cimatti et al. [13], interval propagation effectively computes a piecewise *constant* approximation, which is less expressive than our bilinear approximations.

In recent years, a novel approach for NRAT based on the *topological degree test* has been proposed [14, 36]. 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 6.1 Consider the formula $x = \mathsf{exp}(3, y) \land y > 0$. To handle it with existing solvers, we have to encode it using the natural exponential function:

$$e^z = 3 \land x = e^{y \cdot z} \land y > 0 \tag{6.1}$$

Here x and y range over the integers and z ranges over the reals. Any model of (6.1) satisfies $z = \ln 3$, where $\ln 3$ is irrational. As finding such models is challenging, the leading tools MathSat [11] and CVC5 [3] fail for $e^z = 3$.

Recently, a variant of our original approach from [24] has been integrated into CVC5 [3], where it is used to solve SMT problems with parametric bitvectors [7]. In their setting, the base of exp is fixed to 2, i.e., they consider a special case of our work. Moreover, they do not use interpolation, prime, or induction lemmas, i.e., their solver for exponentials is significantly simpler than ours. Note, however, that their primary goal is to solve problems with parametric bitvectors, and their solver for exponentials is just one component in a sophisticated toolchain for that purpose.

MetiTarski [1] integrates decision procedures for real closed fields and approximations for transcendental functions into the theorem prover Metis [32] to prove theorems about the reals. In a related line of work, iSAT3 [16] has been coupled with SPASS [47]. Clearly, these approaches differ fundamentally from ours.

Recently, the complexity of decidable extensions of linear integer arithmetic with exponentiation has been investigated [6, 10, 34]. Benedikt et al. [6] consider *Semenov arithmetic*, which extends LIA with a function $\lambda x.c^{|x|}$ for some fixed natural number $c > 1$. Thus, it is equivalent to EIA with quantifiers, but without the functions "$\cdot$", "div", and "mod", and where the first argument of all occurrences of exp must be the same constant. In contrast, Chistikov et al. [10] consider only quantifier-free conjunctions of literals from Semenov arithmetic. Integrating such decision procedures into our approach is an interesting direction for future work. The technique of Chistikov et al. is more efficient than the one of Benedikt et al., but as it can only handle conjunctions of literals, it needs to be implemented as a theory solver, i.e., *within* an SMT solver. In contrast, our approach can be implemented *on top of* an SMT solver, which is easier, as it does not require detailed knowledge of the internals of the SMT solver. Thus, the approach of Benedikt et al. seems to be a better fit for our framework.

Karimov et al. [34] showed that decidability of the existential fragment of an extension of Semenov arithmetic with a second function $\lambda x.d^{|x|}$ (where c and d are multiplicatively independent) would lead to mathematical breakthroughs. So as it stands, we cannot hope for a complete solver as soon as we go beyond Semenov arithmetic.

An alternative way to solve SMT problems with exponentiation is to provide the recursive definition of exp as universally quantified axioms. Then SMT solvers with support for quantifiers try to instantiate these axioms suitably in order to derive a conflict. To find suitable instantiations, various heuristics have been proposed [38, 42–45]. However, such approaches are only suitable for proving *un*satisfiability, and for that purpose, our evaluation shows that our approach outperforms quantifier instantiation techniques, see Sect. 7.3.

7 Implementation and Evaluation

We implemented our approach in our novel tool SwInE-Z3, which is based on Z3 4.14.1 [39]. For more information on SwInE-Z3, including precompiled releases for all platforms, we refer to [26].

We compare SwInE-Z3 with the default configuration of the original implementation from [24], called SwInE-Legacy in the sequel. In contrast to SwInE-Z3, SwInE-Legacy does not support prime lemmas, induction lemmas, and phasing. While SwInE-Z3 directly uses the API of Z3, SwInE-Legacy is based on SMT-Switch [37], a library that offers a unified interface for various SMT solvers. By default, SwInE-Legacy also uses Z3. For more information on SwInE-Legacy, we refer to [26].

7.1 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 [20] on the benchmarks for *linear Constrained Horn Clauses (CHCs)* with *linear integer arithmetic* from the *CHC Competitions* 2022 and 2023 as well as on the benchmarks for *Termination* and *Complexity of Integer Transition Systems* from the *Termination Problems Database (TPDB)*, the benchmark set of the *Termination and Complexity Competition* [30], 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 (called LoAT *Problems* below), which are available online [25]:

- 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*)

Unfortunately, these sets do not contain challenging unsatisfiable benchmarks: As shown in our evaluation from [24], the unsatisfiable instances can mostly be solved without reasoning about exponentials at all. More precisely, most of these benchmarks are not just unsatisfiable in EIA (where the interpretation of exp is fixed), but they are also unsatisfiable in UFNIA, where exp is an uninterpreted function. To obtain challenging unsatisfiable examples, we generated one more set of benchmarks by validating the results of the recurrence solver PURRS [2]. More precisely, we considered all recurrence relations from PURRS's test suite[12] where both the recurrence relation and its expected solution can be expressed with polynomials and exponentials. Consider such a recurrence relation

$$f(n) = t \tag{7.1}$$

where f is the inductively defined function, i.e., the term t contains subterms of the form $f(n-c)$, $c \in \mathbb{N}_{>0}$. Let k be the maximal natural number such that $f(n-k)$ is a subterm of t. As an example, consider the recurrence relation (4.12) in Thm. 4.15 where t is $2 \cdot f(n-1) + 2^n$ and thus, $k = 1$. Moreover, let s be the expected solution for (7.1) given in the test suite. For such expected solutions, we always

[12] https://github.com/aprove-developers/LoAT-purrs/blob/master/tests/recurrences

have $n \in \mathcal{V}(s) \subseteq \mathcal{V}(t)$, and for all subterms $f(q)$ of s, we have $q \in \{0, \ldots, k-1\}$. Here, $\mathcal{V}(s)$ and $\mathcal{V}(t)$ denote the sets of variables occurring in s and t, respectively. For the solution (4.13) in Thm. 4.15, we have $s = (f(0) + n) \cdot 2^n$, i.e., here the only f-subterm of s is $f(0)$. To construct new benchmarks, we considered the following SMT problem:

$$n \geq k \wedge s_{\mathsf{exp}} \neq t_{\mathsf{exp}}[f(n-i)/s_{\mathsf{exp}}[n/n-i] \mid 1 \leq i \leq k] \tag{7.2}$$

where s_{exp} and t_{exp} result from s and t by replacing all exponentials (i.e., all subterms of the form p^q where $q \notin \mathbb{Z}$) with $\mathsf{exp}(p, q)$. Moreover, $s_{\mathsf{exp}}[n/n-i]$ results from s_{exp} by replacing n with $n-i$ and $t_{\mathsf{exp}}[f(n-i)/s_{\mathsf{exp}}[n/n-i] \mid 1 \leq i \leq k]$ results from t_{exp} by replacing each subterm $f(n-i)$ with $s_{\mathsf{exp}}[n/n-i]$, for all $1 \leq i \leq k$. In this way, we obtained the formula (4.14) in Thm. 4.15. If (7.2) is unsatisfiable, then the given solution is correct. More precisely, let $\vec{x} = \mathcal{V}(s) \setminus \{n\}$. Then unsatisfiability of (7.2) implies that the equation (7.1) holds for all $n \in \mathbb{N}$, all $\vec{x} \in \mathbb{Z}^{|\vec{x}|}$, and all $c_0, \ldots, c_{k-1} \in \mathbb{Z}$ when interpreting f as

$$n \mapsto \begin{cases} c_n & \text{if } n < k \\ s[f(i)/c_i \mid 0 \leq i < k] & \text{otherwise.} \end{cases}$$

In this way, we obtained 37 hard unsatisfiable benchmarks (called PURRS *Problems* below), which are also available online [25].

7.2 Evaluation on the LoAT Problems

We first report on our experiments on the LoAT Problems. To evaluate the impact of the different components of our approach, we tested with configurations where we disabled rewriting, symmetry lemmas, monotonicity lemmas, bounding lemmas, prime lemmas, induction lemmas, interpolation lemmas, or phasing. All experiments were performed on the CLAIX-2023-HPC nodes of the RWTH University High Performance Computing Cluster[13] with a memory limit of 10560 MiB ($\approx$ 11GB) and a timeout of 10 s 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.

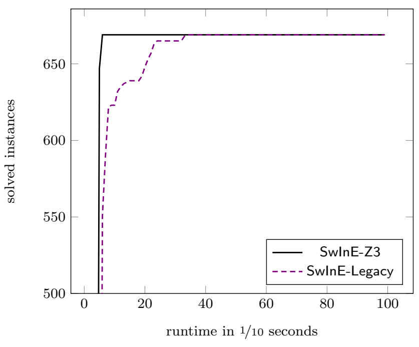
The results can be seen in Fig. 7.1, 7.2, 7.3, and 7.4. All but two of the 4627 benchmarks can be solved by SwInE-Z3. Most configurations, including SwInE-Legacy, can solve all CHC Comp and Termination Problems. The only exceptions are the configurations without bounding or interpolation lemmas, which clearly shows their importance.

The Complexity Problems (Fig. 7.3) are by far the hardest, and thus also the most interesting. Here, SwInE-Z3 clearly outperforms SwInE-Legacy (2 vs. 58 unsolved instances). Moreover, Fig. 7.3 shows that all components of our approach except for monotonicity lemmas, prime lemmas, and induction lemmas are crucial for the performance of SwInE-Z3 on this benchmark set. Thus, *phasing* is the main reason why SwInE-Z3 is superior to SwInE-Legacy on the satisfiable instances from our benchmark suite.

[13] `https://help.itc.rwth-aachen.de/service/rhr4fjjutttf`

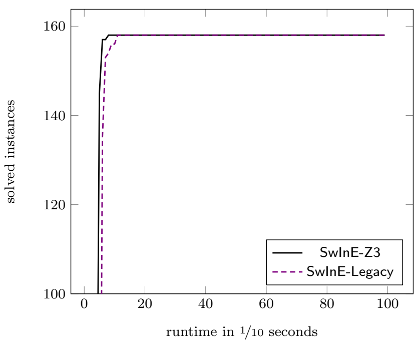
solver	configuration	**sat**	**unsat**	**unknown**
SwInE-Z3	default	296	373	0
SwInE-Legacy		296	373	0
SwInE-Z3	no rewriting	296	373	0
	no symmetry	296	373	0
	no monotonicity	296	373	0
	no bounding	111	373	185
	no prime	296	373	0
	no induction	296	373	0
	no interpolation	252	373	44
	no phasing	296	373	0

Fig. 7.1: CHC Comp '22



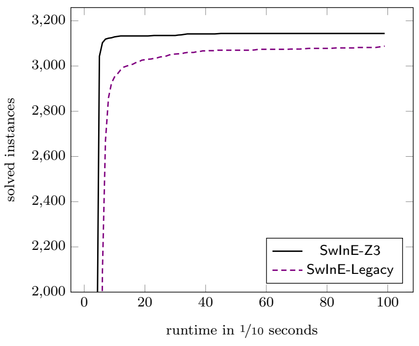
solver	configuration	**sat**	**unsat**	**unknown**
SwInE-Z3	default	87	71	0
SwInE-Legacy		87	71	0
SwInE-Z3	no rewriting	87	71	0
	no symmetry	87	71	0
	no monotonicity	87	71	0
	no bounding	79	71	8
	no prime	87	71	0
	no induction	87	71	0
	no interpolation	38	71	49
	no phasing	87	71	0

Fig. 7.2: CHC Comp '23



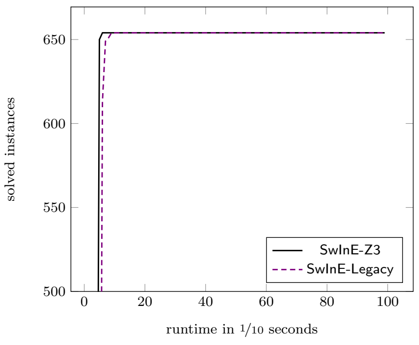
solver	configuration	**sat**	**unsat**	**unknown**
SwInE-Z3	default	1355	1789	2
SwInE-Legacy		1299	1789	58
SwInE-Z3	no rewriting	1195	1789	162
	no symmetry	951	1789	406
	no monotonicity	1355	1789	2
	no bounding	890	1789	467
	no prime	1355	1789	2
	no induction	1355	1789	2
	no interpolation	1241	1788	117
	no phasing	1271	1789	86

Fig. 7.3: Complexity



solver	configuration	**sat**	**unsat**	**unknown**
SwInE-Z3	default	223	431	0
SwInE-Legacy		223	431	0
SwInE-Z3	no rewriting	223	431	0
	no symmetry	223	431	0
	no monotonicity	223	431	0
	no bounding	190	429	35
	no prime	223	431	0
	no induction	223	431	0
	no interpolation	155	429	70
	no phasing	223	431	0

Fig. 7.4: Termination



The fact that monotonicity lemmas are of little importance here is not surprising: Monotonicity lemmas were already of little use when Z3 was used as backend solver in the evaluation of SwInE-Legacy [24]. However, these lemmas significantly improved the performance of SwInE-Legacy with CVC5 [3] as backend solver. Hence, the usefulness of monotonicity lemmas depends on the details of the underlying solver. Moreover, recall that our leading example cannot be solved without monotonicity lemmas, see Thm. 4.23.

It is also not surprising that induction lemmas do not contribute much to the performance of SwInE-Z3 on this benchmark set, as their main purpose is proving unsatisfiability, see Sect. 7.3.

However, it is slightly disappointing that prime lemmas are not needed for the LoAT problems. Nevertheless, we believe that prime factorizations are of such fundamental relevance that prime lemmas are likely to be useful for benchmarks from other sources in the future.

The runtime of SwInE can be seen in the plots on the right-hand side of Page 27. Most instances can be solved in a fraction of a second, as desired for our use case. On the CHC Comp '23 and the Termination Problems, SwInE-Z3 and SwInE-Legacy are equally efficient. On the CHC Comp '22 and the Complexity Problems, SwInE-Z3 is slightly faster than SwInE-Legacy. We refer to [25] for more details on our evaluation.

7.3 Evaluation on the PURRS Problems

We continue with the evaluation on the PURRS Problems, where we used a wall clock timeout of 300s. All of them are unsatisfiable (provided that the test cases from PURRS's test suite are correct), and most of these problems require inductive reasoning, so they are particularly suitable to illustrate the usefulness of induction lemmas. Hence, we considered two configurations of SwInE-Z3: The default configuration (where all features are enabled), and a configuration without induction lemmas.

In contrast to those LoAT Problems that are unsatisfiable, the PURRS Problems require reasoning about exp for proving unsatisfiability. An alternative approach to tackle such problems is to use standard SMT solvers, and to add the (universally quantified) recursive definition of exp explicitly. Thus, we also considered Z3 4.14.1, where we added the following assertions to each benchmark:

$$\begin{aligned}\forall x \in \mathbb{Z}.\ \mathsf{exp}(x,0) &= 1\\ \forall x,y \in \mathbb{Z}.\ y>0 \implies \mathsf{exp}(x,y) &= x \cdot \mathsf{exp}(x,y-1)\\ \forall x,y \in \mathbb{Z}.\ y<0 \implies \mathsf{exp}(x,y) &= x \cdot \mathsf{exp}(x,y+1)\end{aligned}$$

Note that SMT solvers use quantifier instantiation to derive contradictions, but adding such quantified assertions prevents them from proving satisfiability. Hence, we did not include Z3 in the evaluation on the LoAT problems in Sect. 7.2.

The results can be seen in Table 7.1. With induction lemmas, SwInE-Z3 solves all instances that Z3 can solve as well as 6 additional instances, and it only fails for 3 instances. Regarding the benchmarks that are solved by both tools, SwInE-Z3 is substantially faster in some cases (`purrs07`, `purrs36`), but sometimes it is the other way around (`purrs35`), so there is no clear picture. Without induction lemmas, SwInE-Z3 is clearly not suitable for this benchmark set.

tool	SwInE-Z3		Z3		SwInE-Z3	
configuration	default		default		no induction	
purrs01	✓	0.45	✓	0.66	🕒	–
purrs02	✓	0.40	✓	0.66	🕒	–
purrs03	🕒	–	🕒	–	🕒	–
purrs04	✓	0.44	✓	0.65	🕒	–
purrs05	✓	0.43	✓	0.44	🕒	–
purrs06	✓	0.45	✓	0.45	🕒	–
purrs07	✓	0.60	✓	31.64	✗	26.7
purrs08	✓	0.58	✓	0.46	✓	0.49
purrs09	✓	0.52	✗	104.58	🕒	–
purrs10	✓	0.50	✗	108.85	🕒	–
purrs11	✓	0.48	✓	0.49	🕒	–
purrs12	✓	3.12	✓	2.17	🕒	–
purrs13	✓	0.49	✓	0.61	🕒	–
purrs14	✓	0.52	✓	0.52	✓	0.42
purrs15	✓	0.43	🕒	–	🕒	–
purrs16	✓	1.35	🕒	–	🕒	–
purrs17	✓	1.42	🕒	–	🕒	–
purrs18	✓	0.54	✓	0.48	🕒	–
purrs19	✓	0.52	✓	0.48	🕒	–
purrs20	✓	0.53	✓	0.49	🕒	–
purrs21	✓	0.41	✓	0.43	🕒	–
purrs22	✓	0.43	✓	0.46	🕒	–
purrs23	✓	0.47	✓	0.44	🕒	–
purrs24	✓	0.46	✓	0.45	🕒	–
purrs25	✓	0.43	✓	0.44	🕒	–
purrs26	✓	0.43	✓	0.42	🕒	–
purrs27	✓	0.40	✓	0.46	🕒	–
purrs28	✓	0.44	✓	0.51	🕒	–
purrs29	✓	0.44	✓	0.53	✗	239.34
purrs30	✓	2.21	🕒	–	🕒	–
purrs31	✗	18.93	🕒	–	✗	87.65
purrs32	✗	77.34	🕒	–	🕒	–
purrs33	✓	0.43	✓	0.46	🕒	–
purrs34	✓	0.50	✓	0.51	✓	0.38
purrs35	✓	21.39	✓	6.12	🕒	–
purrs36	✓	4.65	✓	38.30	🕒	–
purrs37	✓	1.36	✓	1.11	🕒	–
solved	34		28		3	

✓: **unsat**, ✗: **unknown**, 🕒: timeout

Table 7.1: PURRS Problems

8 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 for proving satisfiability, both in practice (see our evaluation in Sect. 7) and in theory, as interpolation lemmas are essential for being able to eliminate *any* counterexample (see Thm. 4.26). Finally, we evaluated the implementation of our approach in our novel tool SwInE-Z3 on thousands of EIA problems that were synthesized from verification tasks using our verification tool LoAT, and on challenging unsatisfiable benchmarks that result from the verification of solutions for recurrence relations. Our evaluation shows that SwInE-Z3 is highly effective for both use cases. In particular, it shows that *induction lemmas* are crucial for proving unsatisfiability, and that our new *phasing* technique allows our novel implementation SwInE-Z3 to outperform its predecessor on satisfiable instances.

Consequently, we integrated SwInE-Z3 into our verification tool LoAT, where it now serves as the default solver for all techniques that require reasoning about exponentials [21, 23].

With SwInE-Z3, we provide an SMT-LIB compliant open-source solver for EIA [26]. In this way, we hope to attract users with applications that give rise to challenging benchmarks, and we hope that other solvers with support for integer exponentiation will follow, with the ultimate goal of standardizing EIA.

Acknowledgements This work has been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 235950644 (Project GI 274/6-2).

References

1. Akbarpour B, Paulson LC (2010) MetiTarski: An automatic theorem prover for real-valued special functions. J Autom Reason 44(3):175–205, doi:10.1007/S10817-009-9149-2
2. Bagnara R, Pescetti A, Zaccagnini A, Zaffanella E (2005) PURRS: Towards computer algebra support for fully automatic worst-case complexity analysis. CoRR abs/cs/0512056, doi:10.48550/arXiv.cs/0512056
3. Barbosa H, Barrett CW, 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 (2022) CVC5: A versatile and industrial-strength SMT solver. In: TACAS '22, LNCS 13243, pp 415–442, doi:10.1007/978-3-030-99524-9_24
4. Bardin S, Finkel A, Leroux J, Petrucci L (2008) FAST: Acceleration from theory to practice. Int J Softw Tools Technol Transf 10(5):401–424, doi:10.1007/s10009-008-0064-3
5. Barrett C, Fontaine P, Tinelli C The Satisfiability Modulo Theories Library (SMT-LIB). `https://www.SMT-LIB.org`. `www.SMT-LIB.org`
6. Benedikt M, Chistikov D, Mansutti A (2023) The complexity of Presburger arithmetic with power or powers. In: ICALP '23, LIPIcs 261, pp 112:1–112:18, doi:10.4230/LIPIcs.ICALP.2023.112
7. Berger Z, Zohar Y, Niemetz A, Preiner M, Reynolds A, Barrett CW, Tinelli C (2025) Bit-precise reasoning with parametric bit-vectors. In: SAT '25, LIPIcs 341, pp 4:1–4:24, doi:10.4230/LIPICS.SAT.2025.4
8. Bozga M, Iosif R, Konečný F (2010) Fast acceleration of ultimately periodic relations. In: CAV '10, LNCS 6174, pp 227–242, doi:10.1007/978-3-642-14295-6_23
9. Bozga M, Iosif R, Konečný F (2012) Relational analysis of integer programs. Tech. Rep. TR-2012-10, VERIMAG, URL `https://www-verimag.imag.fr/TR/TR-2012-10.pdf`
10. Chistikov D, Mansutti A, Starchak MR (2024) Integer linear-exponential programming in NP by quantifier elimination. In: ICALP '24, LIPIcs 297, pp 132:1–132:20, doi:10.4230/LIPICS.ICALP.2024.132
11. Cimatti A, Griggio A, Schaafsma BJ, Sebastiani R (2013) The MathSAT5 SMT solver. In: TACAS '13, LNCS 7795, pp 93–107, doi:10.1007/978-3-642-36742-7_7
12. Cimatti A, Griggio A, Irfan A, Roveri M, Sebastiani R (2018) Experimenting on solving nonlinear integer arithmetic with incremental linearization. In: SAT '18, LNCS 10929, pp 383–398, doi:10.1007/978-3-319-94144-8_23
13. Cimatti A, Griggio A, Irfan A, Roveri M, Sebastiani R (2018) Incremental linearization for satisfiability and verification modulo nonlinear arithmetic and transcendental functions. ACM Trans Comput Log 19(3):19:1–19:52, doi:10.1145/3230639
14. Cimatti A, Griggio A, Lipparini E, Sebastiani R (2022) Handling polynomial and transcendental functions in SMT via unconstrained optimisation and topological degree test. In: ATVA '22, LNCS 13505, pp 137–153, doi:10.1007/978-3-031-19992-9_9
15. Clarke EM, Grumberg O, Jha S, Lu Y, Veith H (2000) Counterexample-guided abstraction refinement. In: CAV '00, LNCS 1855, pp 154–169, doi:10.1007/10722167_15

16. Fränzle M, Herde C, Teige T, Ratschan S, Schubert T (2007) Efficient solving of large non-linear arithmetic constraint systems with complex Boolean structure. J Satisf Boolean Model Comput 1(3-4):209–236, doi:10.3233/sat190012
17. Frohn F, Giesl J (2019) Proving non-termination via loop acceleration. In: FMCAD '19, pp 221–230, doi:10.23919/FMCAD.2019.8894271
18. Frohn F, Giesl J (2019) Termination of triangular integer loops is decidable. In: CAV '19, LNCS 11562, pp 426–444, doi:10.1007/978-3-030-25543-5_24
19. Frohn F, Naaf M, Brockschmidt M, Giesl J (2020) Inferring lower runtime bounds for integer programs. ACM Trans Program Lang Syst 42(3):13:1–13:50, doi:10.1145/3410331
20. Frohn F, Giesl J (2022) Proving non-termination and lower runtime bounds with LoAT. In: IJCAR '22, LNCS 13385, pp 712–722, doi:10.1007/978-3-031-10769-6_41
21. Frohn F, Giesl J (2023) ADCL: Acceleration Driven Clause Learning for constrained Horn clauses. In: SAS '23, LNCS 14284, pp 259–285, doi:10.1007/978-3-031-44245-2_13
22. Frohn F, Giesl J (2023) Proving non-termination by Acceleration Driven Clause Learning. In: CADE '23, LNCS 14132, pp 220–233, doi:10.1007/978-3-031-38499-8_13
23. Frohn F, Giesl J (2024) Integrating loop acceleration into bounded model checking. In: FM '24, LNCS 14933, pp 73–91, doi:10.1007/978-3-031-71162-6_4
24. Frohn F, Giesl J (2024) Satisfiability modulo exponential integer arithmetic. In: IJCAR '24, LNCS 14739, pp 344–365, doi:10.1007/978-3-031-63498-7_21
25. Frohn F, Giesl J Evaluation of "Satisfiability modulo exponential integer arithmetic". URL `https://aprove-developers.github.io/swine-journal-eval`
26. Frohn F, Giesl J SwInE. URL `https://ffrohn.github.io/swine`
27. Frohn F, Giesl J SwInE on GitHub. URL `https://github.com/ffrohn/swine`
28. Frohn F, Giesl J SwInE-Z3 on GitHub. URL `https://github.com/ffrohn/swine-z3`
29. Gao S, Avigad J, Clarke EM (2012) δ-complete decision procedures for satisfiability over the reals. In: IJCAR '12, LNCS 7364, pp 286–300, doi:10.1007/978-3-642-31365-3_23
30. Giesl J, Rubio A, Sternagel C, Waldmann J, Yamada A (2019) The termination and complexity competition. In: TACAS '19, LNCS 11429, pp 156–166, doi:10.1007/978-3-030-17502-3_10
31. Hark M, Frohn F, Giesl J (2025) Termination of triangular polynomial loops. Form Methods Syst Des 65(1):70–132, doi:10.1007/s10703-023-00440-z
32. Hurd J (2003) First-order proof tactics in higher-order logic theorem provers. In: STRATA '03, Report NASA/CP-2003-212448, pp 56–68, URL `https://apps.dtic.mil/sti/pdfs/ADA418902.pdf`
33. Jeannet B, Schrammel P, Sankaranarayanan S (2014) Abstract acceleration of general linear loops. In: POPL '14, pp 529–540, doi:10.1145/2535838.2535843
34. Karimov T, Luca F, Nieuwveld J, Ouaknine J, Worrell J (2025) On the decidability of Presburger arithmetic expanded with powers. In: SODA '25, pp 2755–2778, doi:10.1137/1.9781611978322.89
35. Kroening D, Lewis M, Weissenbacher G (2015) Under-approximating loops in C programs for fast counterexample detection. Formal Methods Syst Des 47(1):75–92, doi:10.1007/s10703-015-0228-1

36. Lipparini E, Ratschan S (2023) Satisfiability of non-linear transcendental arithmetic as a certificate search problem. In: NFM '23, LNCS 13903, pp 472–488, doi:10.1007/978-3-031-33170-1_29
37. Mann M, Wilson A, Zohar Y, Stuntz L, Irfan A, Brown K, Donovick C, Guman A, Tinelli C, Barrett CW (2021) SMT-Switch: A solver-agnostic C++ API for SMT solving. In: SAT '21, LNCS 12831, pp 377–386, doi:10.1007/978-3-030-80223-3_26
38. de Moura L, Bjørner N (2007) Efficient E-matching for SMT solvers. In: CADE '07, LNCS 4603, pp 183–198, doi:10.1007/978-3-540-73595-3_13
39. de Moura L, Bjørner N (2008) Z3: An efficient SMT solver. In: TACAS '08, LNCS 4963, pp 337–340, doi:10.1007/978-3-540-78800-3_24
40. Popescu P, Díaz-Barrero, Luis J (2006) Certain inequalities for convex functions. JIPAM 7(2), URL `http://eudml.org/doc/128518`
41. Pritchard P (1981) A sublinear additive sieve for finding prime numbers. Commun ACM 24(1):18–23, doi:10.1145/358527.358540
42. Reynolds A, Tinelli C, Goel A, Krstic S, Deters M, Barrett CW (2013) Quantifier instantiation techniques for finite model finding in SMT. In: CADE '24, LNCS 7898, pp 377–391, doi:10.1007/978-3-642-38574-2_26
43. Reynolds A, Tinelli C, de Moura L (2014) Finding conflicting instances of quantified formulas in SMT. In: FMCAD '14, pp 195–202, doi:10.1109/FMCAD.2014.6987613
44. Reynolds A, Deters M, Kuncak V, Tinelli C, Barrett CW (2015) Counterexample-guided quantifier instantiation for synthesis in SMT. In: CAV '15, LNCS 9207, pp 198–216, doi:10.1007/978-3-319-21668-3_12
45. Reynolds A, Barbosa H, Fontaine P (2018) Revisiting enumerative instantiation. In: TACAS '18, LNCS 10806, pp 112–131, doi:10.1007/978-3-319-89963-3_7
46. Vasić PM, Pečarić JE (1979) On the Jensen inequality. Publikacije Elektrotehničkog fakulteta Serija Matematika i fizika 634/677:50–54, URL `https://www.jstor.org/stable/43668091`
47. Weidenbach C, Dimova D, Fietzke A, Kumar R, Suda M, Wischnewski P (2009) SPASS version 3.5. In: CADE '22, LNCS 5663, pp 140–145, doi:10.1007/978-3-642-02959-2_10

A Missing Proofs

A.1 Proof of Thm. 4.4

Lemma A.1 *We have*

$$\varphi \equiv_{\mathrm{EIA}} \textsc{FoldConstants}(\varphi) \qquad \textit{and} \qquad \varphi \equiv_{\mathrm{EIA}} \textsc{Rewrite}(\varphi).$$

Proof In Sect. 4, we already showed the soundness of the first rewrite rule. For the second rewrite rule $\mathsf{exp}(\mathsf{exp}(x, y), z) \to \mathsf{exp}(x, y \cdot z)$, we have

$$\begin{aligned} [\![\mathsf{exp}(\mathsf{exp}(x,y),z)]\!] &= [\![\mathsf{exp}(x,y)]\!]^{|[\![z]\!]|} = \left([\![x]\!]^{|[\![y]\!]|}\right)^{|[\![z]\!]|} = [\![x]\!]^{|[\![y]\!]|\cdot|[\![z]\!]|} \\ &= [\![x]\!]^{|[\![y]\!]\cdot[\![z]\!]|} = [\![x]\!]^{|[\![y\cdot z]\!]|} = [\![\mathsf{exp}(x, y\cdot z)]\!]\,. \end{aligned}$$

For the third rewrite rule $\mathsf{exp}(x, y) \cdot \mathsf{exp}(z, y) \to \mathsf{exp}(x \cdot z, y)$, we have

$$\begin{aligned} [\![\mathsf{exp}(x,y)\cdot\mathsf{exp}(z,y)]\!] &= [\![\mathsf{exp}(x,y)]\!]\cdot[\![\mathsf{exp}(z,y)]\!] = [\![x]\!]^{|[\![y]\!]|}\cdot[\![z]\!]^{|[\![y]\!]|} = ([\![x]\!]\cdot[\![z]\!])^{|[\![y]\!]|} \\ &= [\![x\cdot z]\!]^{|[\![y]\!]|} = [\![\mathsf{exp}(x\cdot z, y)]\!]\,. \end{aligned}$$

A.2 Proof of Thm. 4.8

Lemma A.2 *Let s, t be terms of sort* **Int**. Then SYM_1 – SYM_3 are EIA-valid.

Proof We already showed the soundness of SYM_1 in Sect. 4. For SYM_2, assume $[\![\mathsf{divisible}_2(t)]\!] = \mathbf{false}$, i.e., assume that $[\![t]\!]$ is odd. Then it remains to show

$$[\![\mathsf{exp}(s,t) = -\mathsf{exp}(-s,t)]\!] = \mathbf{true}, \quad \text{i.e.,} \quad [\![\mathsf{exp}(s,t)]\!] = [\![-\mathsf{exp}(-s,t)]\!]\,.$$

We have:

$$\begin{aligned}[\![\mathsf{exp}(s,t)]\!] &= [\![s]\!]^{|[\![t]\!]|} \\ &= -\,[\![-s]\!]^{|[\![t]\!]|} && \text{(as } [\![t]\!]\text{, and thus also } |\,[\![t]\!]\,|\text{, is odd)} \\ &= -\,[\![\mathsf{exp}(-s,t)]\!] \\ &= [\![-\mathsf{exp}(-s,t)]\!]\end{aligned}$$

For SYM_3, we have to show

$$[\![\mathsf{exp}(s,t) = \mathsf{exp}(s,-t)]\!] = \mathbf{true}, \quad \text{i.e.,} \quad [\![\mathsf{exp}(s,t)]\!] = [\![\mathsf{exp}(s,-t)]\!]\,.$$

We have:

$$[\![\mathsf{exp}(s,t)]\!] = [\![s]\!]^{|[\![t]\!]|} = [\![s]\!]^{|-[\![t]\!]|} = [\![s]\!]^{|[\![-t]\!]|} = [\![\mathsf{exp}(s,-t)]\!]$$

A.3 Proof of Thm. 4.12

Lemma A.3 *Let s, t be terms of sort* **Int**. Then BND_1 – BND_5 are EIA-valid.

Proof We already showed EIA-validity of BND_5 in Sect. 4. For BND_1, assume

$$[\![t = 0]\!] = \mathbf{true}, \quad \text{i.e.,} \quad [\![t]\!] = 0.$$

Then it remains to show

$$[\![\mathsf{exp}(s,t) = 1]\!] = \mathbf{true}, \quad \text{i.e.,} \quad [\![\mathsf{exp}(s,t)]\!] = 1.$$

We have

$$[\![\mathsf{exp}(s,t)]\!] = [\![s]\!]^{|[\![t]\!]|} = [\![s]\!]^{|0|} = [\![s]\!]^0 = 1.$$

For BND_2, assume

$$[\![t = 1]\!] = \mathbf{true}, \quad \text{i.e.,} \quad [\![t]\!] = 1.$$

Then it remains to show

$$[\![\mathsf{exp}(s,t) = s]\!] = \mathbf{true}, \quad \text{i.e.,} \quad [\![\mathsf{exp}(s,t)]\!] = [\![s]\!]\,.$$

We have

$$[\![\mathsf{exp}(s,t)]\!] = [\![s]\!]^{|[\![t]\!]|} = [\![s]\!]^{|1|} = [\![s]\!]^1 = [\![s]\!]\,.$$

For BND_3, we first show the implication from left to right. So assume

$$[\![s = 0 \land t \neq 0]\!] = \mathbf{true}, \quad \text{i.e.,} \quad [\![s]\!] = 0 \text{ and } [\![t]\!] \neq 0.$$

Then it remains to show

$$[\![\mathsf{exp}(s,t) = 0]\!] = \mathbf{true}, \quad \text{i.e.,} \quad [\![\mathsf{exp}(s,t)]\!] = 0.$$

We have

$$[\![\mathsf{exp}(s,t)]\!] = [\![s]\!]^{|[\![t]\!]|} = 0^{|[\![t]\!]|} = 0$$

where the last equality holds as we have $[\![t]\!] \neq 0$, and thus $|\,[\![t]\!]\,| > 0$.

For the implication from right to left, assume

$$[\![\mathsf{exp}(s,t) = 0]\!] = \mathbf{true}, \quad \text{i.e.,} \quad [\![\mathsf{exp}(s,t)]\!] = 0.$$

Then it remains to show

$$[\![s = 0 \land t \neq 0]\!] = \mathbf{true}, \quad \text{i.e.,} \quad [\![s]\!] = 0 \text{ and } [\![t]\!] \neq 0.$$

We have

$$[\![\mathsf{exp}(s,t)]\!] = [\![s]\!]^{|[\![t]\!]|} = 0.$$

So as $[\![s]\!]^{|[\![t]\!]|} \neq 1$, we get $|\,[\![t]\!]\,| \neq 0$ and thus also $[\![t]\!] \neq 0$. As a product is only 0 if one of its factors is 0, we also get $[\![s]\!] = 0$.

For BND$_4$, assume

$$[\![s = 1]\!] = \mathbf{true}, \quad \text{i.e.,} \quad [\![s]\!] = 1.$$

Then it remains to show

$$[\![\mathsf{exp}(s,t) = 1]\!] = \mathbf{true}, \quad \text{i.e.,} \quad [\![\mathsf{exp}(s,t)]\!] = 1.$$

We have

$$[\![\mathsf{exp}(s,t)]\!] = [\![s]\!]^{|[\![t]\!]|} = 1^{|[\![t]\!]|} = 1.$$