A Decision Procedure for String Constraints with String-Integer Conversion and Flat Regular Constraints

Hao Wu[1,2], Yu-Fang Chen[3], Zhilin Wu[1,2] and Naijun Zhan[1,2*]

[1*]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China.
[2]University of Chinese Academy of Sciences, Beijing, China.
[3]Institute of Information Science, Academia Sinica, Taiwan, Republic of China.

*Corresponding author(s). E-mail(s): znj@ios.ac.cn;
Contributing authors: wuhao@ios.ac.cn; yfc@iis.sinica.edu.tw; wuzl@ios.ac;

Abstract

String constraint solving is the core of various testing and verification approaches for scripting languages. Among algorithms for solving string constraints, flattening is a well-known approach that is particularly useful in handling satisfiable instances. As string-integer conversion is an important function appearing in almost all scripting languages, Abdulla et al. extended the flattening approach to this function recently. However, their approach supports only a special flattening pattern and leaves the support of the general flat regular constraints as an open problem. In this paper, we fill the gap and propose a complete flattening approach for the string-integer conversion. The approach is built upon a quantifier elimination procedure for the linear-exponential arithmetic (namely, the extension of Presburger arithmetic with exponential functions) proposed by Point in 1986. The complexity of the quantifier elimination procedure is analyzed for the first time and is shown to be 3-EXPSPACE if the formula contains only existential quantifiers. While the quantifier elimination procedure by Point is too expensive to be implemented efficiently, we propose various optimizations and provide a prototypical implementation. We evaluate the performance of our implementation on the benchmarks that are generated from the string hash functions as well as randomly. The experimental results show that our implementation outperforms the state-of-the-art solvers.

Keywords: String-integer conversion, Flat regular constraints, Exponential function, Presburger arithmetic, Quantifier elimination

1 Introduction

The emerging of scripting languages boosted the needs of better approaches and tools to ensure program quality. Comparing with traditional programming languages, string data type plays a more critical role in its analysis. String constraint solvers are the engine of modern scripting program analysis techniques. Due to the high demand, in recent years, there is a boosting amount of publications on this subject.

However, research progress of string constraint solving has been hampered by many major challenges in both theory and tool implementation aspects (including long-standing open problems). Logical theories over strings have to allow string concatenation, which is arguably the most fundamental operation of strings. The most celebrated result concerning theories of strings is Makanin's result on deciding the satisfiability problem for *word equations* [1]. A simple example of a word equation is $xaby = ybax$, where x, y are variables, and $a, b \in \Sigma$ are constant letters. A word equation is satisfiable if it has a solution, i.e., an assignment that maps variables to strings over the alphabet Σ which equates the left-hand side with the right-hand side of the equation. The correctness proof of Makanin's algorithm is arguably one of the most complex termination proofs in computer science. Makanin's result can be extended to include *regular constraints* (a.k.a. regular expression matching, e.g., $x \in (ba)^*$), and arbitrary Boolean connectives. This extension is called word equations with regular constraints. However, the satisfiability problem of word equations together with length constraints (e.g., $|x| = |y|+1 \wedge wx = yx$) is still open. The complexity of the satisfiability of word equations with regular constraints was proven to be PSPACE-complete by Plandowski [2], after decades of improvement of the original algorithm by Makanin.

Satisfiability of word equations is a special instance of Hilbert's 10th problem. In the past, the original motivation of studying word equations was for finding an undecidability proof of Hilbert's 10th problem. However, the motivation is no longer valid since Makanin finds a decision procedure. Recently, driven by the need for program analysis, people started to revisit the problem and its extensions to describe the complete string library APIs in conventional programming languages. Many highly efficient solvers for string constraints, to name a few, CVC4 [3], Z3 [4], Z3-Str3 [5], S3 [6], Norn [7], Ostrich [8], Sloth [9], ABC [10], Stranger [11], and Trau [12], are developed in the last decade. The satisfiability of *string-integer conversion constraints*, e.g., $wx = yx \wedge |x| > \mathsf{parseInt}(y)$, has been proven undecidable in [13]. However, this kind of constraints is pervasive in scripting language programs. For example, it is common that programs read string inputs from text files and converts a part of the string input to integers. Even more crucially, in many programming languages, the string-integer conversion is a part of the definition of their core semantics [14]. JavaScript, which powers most interactive content on the Web and increasingly server-side code with Node.js, is one of such languages.

Due to the difficulties in solving string constraints and, in practice, satisfiable string constraints are more critical for automatic testing, one idea is to have separate specialized procedures for solving satisfiable sub-problems. Currently, there are two main specialized approaches for proving satisfiability. The first is to consider only strings of bounded length. This approach is taken in the first-generation solvers such as Hampi [15] and Kuluza [16]. Although they are useful in handling many practical cases, they fail short when the all string solutions exceed the selected bound. For example, a constraint of the form $x.y \neq z \wedge |x| > 2000$ would be quite challenging to handle using those solvers.

One more recent approach is flattening [17–19]. The idea is to restrict the solution space of string variables to *flat languages* (see Section 3). The major benefit of considering this class is two-fold. First, under the restriction, the potential solution space is still infinite, which gives us a higher potential of finding solutions. For instance, we can find a solution for $x.y \neq z \wedge |x| > 2000$ under a very simple restriction: all variables are in a^*, which a is an element of the alphabet. Second, and more importantly, because we can convert the membership problem of a flat language to the satisfiability problem of a Presburger arithmetic formula, the class of word equation + flat languages + length constraints is decidable.

The paper of Abdulla et al. [17] has considered adding string-integer conversion constraints to the above class. It proposed an algorithm for a restricted form of flat languages and left the support of general flat languages as an open problem. For string-integer conversion, their approach projects the solution to string-integer conversion to a finite solution space (in a way similar to the PASS [20] approach).

In this paper, we give a complete solution to this problem. We propose a decision procedure for the class of word equation + flat languages + length constraints + string-integer conversion. The basic idea of our approach can be sketched as follows: we first reduce the satisfiability problem to the corresponding satisfiability problem of the theory of Presburger Arithmetic with exponential functions (denoted by ExpPA), more precisely, the existential fragment of ExpPA; then, according to the decidability of ExpPA, we obtain the decidability of the original satisfiability problem.

The decidability of ExpPA was first shown by Semenöv in [21]. Nevertheless, Semenöv did not provide an explicit decision procedure. To remedy this, in [22], Point presented the first quantifier elimination procedure for the satisfiability of ExpPA. Partially attributed to the fact that Point's procedure in [22] was presented in a mathematical and dense way, this quantifier elimination procedure has mostly eluded the attentions of computer science researchers[1]. To the best of our knowledge, no implementation based on Point's procedure has been available up to now.

Aiming at introducing Point's procedure to computer science researchers, we reformulate (and slightly improve) Point's procedure in a way, which, hopefully, is more accessible for computer science researchers (Section 4). This can be seen as another contribution of this paper. We also analyze the deterministic upper bound of complexity for the first time: given a ExpPA formula with only existential quantifiers, eliminating all quantifiers has a 3-EXPSPACE complexity. Furthermore, we propose

[1] There are only two citations in Google Scholar.

various optimizations (Section 6) and achieve the first prototypical implementation of Point's procedure (Section 7).

In fact, other than the theoretical difficulties, in practice, the string-integer conversion is quite challenging for state-of-the-art solvers. Here we illustrate a toy example that mimic the "mining" step of block-chain construction. Essentially, given a string hash function $\mathsf{hash}(w)$, the goal of the mining step is to find a nonce n such that when inserting n to the text to be protected, say w_1, w_2, $\mathsf{hash}(w_1.n.w_2)$ satisfies a certain pattern, e.g., the last k digits are zeros. If w_1 or w_2 are modified, one needs to compute another n satisfies the desired pattern. Below we consider a simple hash function: $\mathsf{hash}(w) = \sum_{i=1}^{n} a_i p^{n-i} \bmod m$, where $p, m \in \mathbb{Z}^+$ with $p, m \geq 2$. It is easy to see that $\mathsf{hash}(w)$ can be seen as a generalization of $\mathsf{parseInt}$ followed by a modulo operation. In particular, if $\Sigma = \Sigma_{\mathsf{num}}$ and $p = 10$, then $\mathsf{hash}(w) = \mathsf{parseInt}(w) \bmod m$. Thus, the problem of finding a suitable input w such that the last k digits of $\mathsf{hash}(w)$ are zeros can be modeled as a string constraint with $\mathsf{parseInt}$. Although the example is seemingly simple, it is already challenging for most state-of-the-art solvers, as shown by our experiment results in Section 7. With the optimizations introduced in Section 6, our implementation manages to solve several variants of the string-hash examples as well as some randomly generated problem instances better than the state-of-the-art solvers (Section 7).

Structure

After the preliminaries in Section 2, we describe how to flatten a string constraint with string-integer conversion to an existential ExpPA formula in Section 3. We present the quantifier elimination procedure for ExpPA in Section 4 and its optimizations in Section 6. Finally, we describe the implementation and experiment results in Section 7.

2 Preliminary

In this section, we fix the notations and introduce some basic concepts, including Presburger arithmetic, finite-state automata, and flat languages.

Integers, strings, and languges

Let $\mathbb{N}$ denote the set of natural numbers, $\mathbb{Z}$ denote the set of integers, and $\mathbb{Z}^+$ denote the set of positive integers. For $n \in \mathbb{Z}^+$, let $[n]$ denote $\{1, \ldots, n\}$.

An *alphabet* Σ is a finite set. Elements of Σ are called *letters*. A *string* w over Σ is a (possibly empty) finite sequence $a_1 \ldots a_n$ with $a_i \in \Sigma$ for every $i \in [n]$. Let ε denote the empty string, namely, the empty sequence. For a string $w = a_1 \ldots a_n \in \Sigma^*$, let w denote the *length* of w, i.e. n. In particular, $|\varepsilon| = 0$. For $w_1 = a_1 \ldots a_m, w_2 = b_1 \ldots b_n \in \Sigma^*$, let $w_1 \cdot w_2$ denote the *concatenation* of w_1 and w_2, that is, $a_1 \ldots a_m b_1 \ldots b_n$. Let Σ^* denote the set of all strings over Σ and Σ^+ denote the set of nonempty strings over Σ. For convenience, we also use Σ_ϵ to denote $\Sigma \cup \{\epsilon\}$. A language L over Σ is a subset of Σ^*.

Presburger Arithmetic

A Presburger Arithmetic (PA) formula is defined by the rules

$$\begin{aligned} \mathbb{t} &\hat{=} c \mid \mathbb{x} \mid \mathbb{t} + \mathbb{t} \mid \mathbb{t} - \mathbb{t}, \\ \phi &\hat{=} \mathbb{t} \odot \mathbb{t} \mid c|\mathbb{t} \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \exists \mathbb{x}.\ \phi \mid \forall \mathbb{x}.\ \phi, \end{aligned}$$

where $\odot \in \{=, <, >, \leq, \geq\}$ and $\mathbb{x}$, c are integer variables and constants, respectively. A *quantifier-free* PA (QFPA) formula is a PA formula containing no quantifiers. A PA formula is in *negation normal form* (NNF) if all occurrences of the negation symbol $\neg$ are before the atomic formulas. A PA formula is called *existential* if it is in NNF and contains no occurrences of universal quantifiers. The set of free variables of ϕ, denoted by $\mathrm{Free}(\phi)$, is defined in a standard manner. We usually write $\phi(\mathbb{x}_1, \cdots, \mathbb{x}_k)$ to denote an PA formula ϕ such that $\mathrm{Free}(\phi) \subseteq \{\mathbb{x}_1, \cdots, \mathbb{x}_k\}$. Given an PA formula ϕ, and an integer interpretation of $\mathrm{Free}(\phi)$, i.e. a function $I : \mathrm{Free}(\phi) \to \mathbb{Z}$, we denote by $I \models \phi$ that I satisfies ϕ (which is defined in the standard manner, with $+$, $-$, and $\mid$ interpreted as the integer addition, subtraction, and divisibility relation respectively), and call I a *model* of ϕ. We use $[\![\phi]\!]$ to denote the set of models of ϕ.

Finite state automata

A finite state automaton (FA) is a tuple $\mathcal{A} = \langle Q, \Sigma, \Delta, q_{init}, F\rangle$, where Q is a finite set of states, Σ is a finite alphabet, $\Delta \subseteq Q \times \Sigma_\epsilon \times Q$ is the transition relation, q_{init} is the initial state, $F \subseteq Q$ is the set of accepting states. A *run* of $\mathcal{A}$ on a string $w = a_1 \dots a_n$ is a sequence $q_0 \xrightarrow{b_1} q_1 \xrightarrow{b_2} \dots \xrightarrow{b_{m-1}} q_{m-1} \xrightarrow{b_m} q_m$ such that $q_0 = q_{init}$, $(q_{i-1}, b_i, q_i) \in \Delta$ for every $i \in [m]$, and $a_1 \dots a_n = b_1 \dots b_m$. A *run* $q_0 \xrightarrow{b_1} q_1 \xrightarrow{b_2} \dots \xrightarrow{b_{m-1}} q_{m-1} \xrightarrow{b_m} q_m$ is *accepting* if $q_m \in F$. A string w is *accepted* by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on w. Let $\mathcal{L}(\mathcal{A})$ denote the set of strings accepted by $\mathcal{A}$. A language $L \subseteq \Sigma^*$ is *regular* if it can be defined by some FA $\mathcal{A}$, namely, $L = \mathcal{L}(\mathcal{A})$.

Flat languages

We will present flat languages. This is a high level, language-theoretical view at the flat automata from [17]. A *flat language* (FL) over Σ is the set of strings that conform to a regular expression of the form $(a^1_1 \dots a^1_{\ell_1})^* \cdot \dots \cdot (a^k_1 \dots a^k_{\ell_k})^*$, where $a^i_j \in \Sigma$ for each $i \in [k]$ and $j \in [\ell_i]$. Intuitively, an FL is a sequence of loops, and the body of each loop is a string of the form $a^i_1 \dots a^i_{\ell_i}$ with ℓ_i letters. For instance, the language defined by $(ab)^*(a)^*(bb)^*$ is an FL.

3 Flattening string constraints with string-integer conversion

In this section, we first define the class of string constraints with string-integer conversion, denoted by $\mathsf{STR}_{\mathsf{parseInt}}$. Then we define the extension of Presburger arithmetic with exponential functions, denoted by ExpPA. Finally, we show how to flatten the string constraints in $\mathsf{STR}_{\mathsf{parseInt}}$ into the arithmetic constraints in ExpPA.

3.1 String constraints with string-integer conversion ($\mathsf{STR}_{\mathsf{parseInt}}$)

In the sequel, we shall define $\mathsf{STR}_{\mathsf{parseInt}}$, the class of string constraints with the string-integer conversion function $\mathsf{parseInt}$.

The function $\mathsf{parseInt}$ takes a decimal string as the input and returns the integer represented by the string[2]. For example, $\mathsf{parseInt}(\text{`}0123\text{'}) = \mathsf{parseInt}(\text{`}123\text{'}) = 10^2 + 10 * 2 + 3 = 123$. Note here we use the quotation marks to delimit the strings.

Formally, the semantics of the $\mathsf{parseInt}$ function is defined as follows. In order to simplify the presentation, we assume all string variables ranging over numerical symbols $\Sigma_{num} = \{0, 1, \ldots, 9\}$. Note that one can easily extend our approach to allow arbitrary finite alphabet. Then $\mathsf{parseInt} : \Sigma_{num}^+ \mapsto \mathbb{N}$ is recursively defined by for every $w \in \Sigma_{num}^+$,

- if $w =$`$i$' for $i \in \Sigma_{num}$, then $\mathsf{parseInt}(\text{`}i\text{'}) = i$;
- for $w = w_1$`$i$' for $i \in \Sigma_{num}$ with $|w_1| \geq 1$, $\mathsf{parseInt}(w) = 10 * \mathsf{parseInt}(w_1) + \mathsf{parseInt}(\text{`}i\text{'})$.

Note that $\mathsf{parseInt}$ is undefined with ε as the input.

In $\mathsf{STR}_{\mathsf{parseInt}}$, there are two types of variables, i.e. the string variables $x, y, \ldots \in \mathcal{X}$ and the integer variables $\mathtt{x}, \mathtt{y}, \ldots \in \mathbb{X}$. A $\mathsf{STR}_{\mathsf{parseInt}}$ formula φ is defined by the following rules, where $\mathsf{len}(t)$ denotes the length of a string t.

$$
\begin{aligned}
t &\mathrel{\hat{=}} a \mid x \mid t \cdot t,\\
\mathtt{t} &\mathrel{\hat{=}} n \mid \mathtt{x} \mid \mathsf{len}(t) \mid \mathsf{parseInt}(t) \mid \mathtt{t} + \mathtt{t} \mid \mathtt{t} - \mathtt{t},\\
\varphi &\mathrel{\hat{=}} t = t \mid x \in \mathcal{A} \mid \mathtt{t} \odot \mathtt{t} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi,
\end{aligned}
$$

where $a \in (\Sigma_{num})_\varepsilon$, $n \in \mathbb{N}$, $\mathcal{A}$ is an FA, and $\odot \in \{=, <, >, \leq, \geq\}$. Let us call t as string terms, $\mathtt{t}$ as integer terms, $t = t$ as string equality constraints, $x \in \mathcal{A}$ as regular constraints, $\mathtt{t} \odot \mathtt{t}$ as arithmetic constraints. Let $\mathsf{SVar}(\varphi)$ and $\mathsf{IVar}(\varphi)$ denote the set of string variables and integer variables respectively occurring in φ.

The $\mathsf{STR}_{\mathsf{parseInt}}$ formulas are interpreted on $I = (I_s, I_i)$ where I_s is a partial function from $\mathcal{X}$ (the set of string variables) to Σ^* and I_i is a partial function from $\mathbb{X}$ (the set of integer variables) to $\mathbb{N}$. Moreover, it is required that the domains of I_s, I_i are finite. Given $I = (I_s, I_i)$, the interpretations of string terms, integer terms, as well as the formulas φ under I are easy to comprehend, thus omitted to avoid tediousness. For $\varphi \in \mathsf{STR}_{\mathsf{parseInt}}$ and $I = (I_s, I_i)$, I satisfies φ if the interpretation of φ under I is $\mathtt{true}$. Let us use $\|\varphi\|$ to denote the set of $I = (I_s, I_i)$ satisfying φ.

Example 1 $x \in \mathcal{A} \wedge \mathsf{parseInt}(x) = 109\mathtt{x} \wedge \mathsf{len}(x) < 100$, where $\mathcal{A}$ is an FA, is an $\mathsf{STR}_{\mathsf{parseInt}}$ formula.

The satisfiability problem of $\mathsf{STR}_{\mathsf{parseInt}}$ is to decide for a given constraint $\varphi \in \mathsf{STR}_{\mathsf{parseInt}}$, whether $\|\varphi\| \neq \emptyset$.

[2]The $\mathsf{parseInt}$ function in scripting languages, e.g. Javascript, is more general in the sense that the base can be a number between 2 and 36. Although our approach works for arbitrary positive bases, we choose to focus on the base 10 in this paper, for readability.

3.2 An Extension of Presburger Arithmetic with Exponential Functions (ExpPA)

ExpPA extends Presburger arithmetic with the exponential function $10^{\mathbb{x}}$ as well as the (partial) functions $\ell_{10}(\mathbb{x})$ (cf. [22]). The function $\ell_{10}(\mathbb{x})$ is inductively defined as follows. For $n \geq 1$, $\ell_{10}(n) = m$ iff $10^m \leq n < 10^{m+1}$. (Note that $\ell_{10}(0)$ is undefined.)

The syntax of ExpPA is obtained from that of PA by adding $10^{\mathbb{t}}$ and $\ell_{10}(\mathbb{t})$ to the definition of integer terms. Specifically, ExpPA formulas are defined by the rules,

$$\begin{aligned}\mathbb{t} &\hat{=} c \mid \mathbb{x} \mid \mathbb{t}+\mathbb{t} \mid \mathbb{t}-\mathbb{t} \mid 10^{\mathbb{t}} \mid \ell_{10}(\mathbb{t}) \\ \phi &\hat{=} \mathbb{t} \odot \mathbb{t} \mid c|\mathbb{t} \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \exists \mathbb{x}.\, \phi \mid \forall \mathbb{x}.\, \phi.\end{aligned}$$

Sometimes we will denote $\neg c|\mathbb{t}$ by $c \nmid \mathbb{t}$ to avoid the negation operator. The semantics of ExpPA are defined similarly to PA with the difference that variables are interpreted over $\mathbb{N}$ (to avoid terms like 10^{-2}), that is, an interpretation is a function $I : \mathrm{Free}(\phi) \to \mathbb{N}$. The quantifier-free and existential ExpPA formulas are defined similarly to PA as well.

Example 2 $10^{\mathbb{x}+\mathbb{y}} + 3 * 10^{\mathbb{y}} = 2\mathbb{z} + 1$ is an ExpPA formula.

We will use the notation $\lambda_{10}(\mathbb{t})$ to denote $10^{\ell_{10}(\mathbb{t})}$. It is easy to show that for all $n \geq 1$, $\lambda_{10}(n) \leq n < 10\lambda_{10}(n)$ holds.

3.3 Flattening $\mathsf{STR}_{\mathsf{parseInt}}$ into ExpPA

We first recall the flattening approach for string constraints in [17], then show how to extend it to deal with parseInt.

A *flat domain restriction* for a string constraint φ is a function $\mathcal{F}_\varphi$ that maps each string variable $x \in \mathsf{SVar}(\varphi)$ to a flat language $(w_{x,1})^* \cdots (w_{x,k_x})^*$, where $w_{x,i} \in \Sigma^+_{num}$ for every $i \in [k_x]$. The flattened semantics of $\varphi \in \mathsf{STR}_{\mathsf{parseInt}}$ is defined as $[\![\varphi]\!]_{\mathcal{F}_\varphi} = \{(I_s, I_i) \in [\![\varphi]\!] \mid \forall x \in \mathsf{SVar}(\varphi).\ I_s(x) \in \mathcal{F}_\varphi(x)\}$.

The flattening of $\varphi \in \mathsf{STR}_{\mathsf{parseInt}}$ under a flat domain restriction $\mathcal{F}_\varphi$ is a ExpPA formula, denoted by $\mathit{flatten}_{\mathcal{F}_\varphi}(\varphi)$, that encodes its flattened semantics. More concretely, $\mathit{flatten}_{\mathcal{F}_\varphi}(\varphi)$ is a formula over the integer variables $\mathsf{IVar}(\varphi)$, and flattening variables $\mathsf{PVar}_{\mathcal{F}_\varphi}(\varphi) = \bigcup_{x \in \mathsf{SVar}(\varphi)} \mathsf{PVar}_{\mathcal{F}_\varphi}(x)$, where $\mathsf{PVar}_{\mathcal{F}_\varphi}(x) = \{\#_{x,i} \mid i \in [k_x]\}$, plus some other auxiliary variables, such that

$$[\![\varphi]\!]_{\mathcal{F}_\varphi} = \mathit{decode}_{\mathcal{F}_\varphi}([\![\mathit{flatten}_{\mathcal{F}_\varphi}(\varphi)]\!]|_{\mathsf{IVar}(\phi) \cup \mathsf{PVar}_{\mathcal{F}_\varphi}(\varphi)}).$$

The decoding function above decodes an interpretation of integer and flattening variables $I_e : \mathsf{IVar}(\varphi) \cup \mathsf{PVar}_{\mathcal{F}_\varphi}(\varphi) \to \mathbb{N}$ as a set $\mathit{decode}_{\mathcal{F}_\varphi}(I_e)$ of interpretations of the φ's integer and string variables (I_s, I_i) with $I_s : \mathsf{SVar}(\varphi) \to \Sigma^*$ and $I_i : \mathsf{IVar}(\phi) \to \mathbb{N}$ such that

- for every $x \in \mathsf{SVar}(\varphi)$, $I_s(x) = w_{x,1}^{I_e(\#_{x,1})} \ldots w_{x,k_x}^{I_e(\#_{x,k_x})}$,
- for every $\mathbb{x} \in \mathsf{IVar}(\varphi)$, $I_i(\mathbb{x}) = I_e(\mathbb{x})$.

The formula $\mathit{flatten}_{\mathcal{F}_\varphi}(\varphi)$ is constructed inductively by following the structure of φ: $\mathit{flatten}_{\mathcal{F}_\varphi}(\varphi_1 \mathbin{\mathfrak{o}} \varphi_2) = \mathit{flatten}_{\mathcal{F}_\varphi}(\varphi_1) \mathbin{\mathfrak{o}} \mathit{flatten}_{\mathcal{F}_\varphi}(\varphi_2)$, where $\mathfrak{o} \in \{\wedge, \vee\}$, and $\mathit{flatten}_{\mathcal{F}_\varphi}(\neg\varphi_1) = \neg\mathit{flatten}_{\mathcal{F}_\varphi}(\varphi_1)$. Therefore, it is sufficient to show how to construct $\mathit{flatten}_{\mathcal{F}_\varphi}(\varphi)$ for atomic constraints φ. In the sequel, we will show how to construct $\mathit{flatten}_{\mathcal{F}_\varphi}(\mathbb{t}_1 \odot \mathbb{t}_2)$ where $\mathsf{parseInt}(t)$ may occur in $\mathbb{t}_1$ or $\mathbb{t}_2$. The construction of $\mathit{flatten}_{\mathcal{F}_\varphi}(\varphi)$ for the other atomic constraints is essentially the same as that in [17] and thus omitted.

For simplicity, we assume that each occurrence of $\mathsf{parseInt}$ (resp. $\mathsf{len}(t)$) in $\mathbb{t}_1 \odot \mathbb{t}_2$ is of the form $\mathsf{parseInt}(x)$ (resp. $\mathsf{len}(x)$) for a string variable x. (Otherwise, we can introduce a fresh variable x' for t in $\mathsf{parseInt}(t)$ or $\mathsf{len}(t)$ and add the constraint $x' = t$.) Then $\mathit{flatten}_{\mathcal{F}_\varphi}(\mathbb{t}_1 \odot \mathbb{t}_2)$ is obtained from $\mathbb{t}_1 \odot \mathbb{t}_2$ by replacing $\mathsf{parseInt}(x)$ with $\mathit{flatten}_{\mathcal{F}_\varphi}(\mathsf{parseInt}(x))$ and $\mathsf{len}(x)$ with $\mathit{flatten}_{\mathcal{F}_\varphi}(\mathsf{len}(x))$, where

- $\mathit{flatten}_{\mathcal{F}_\varphi}(\mathsf{parseInt}(x)) \hat{=} \mathbb{t}_{x,1}$ such that $(\mathbb{t}_{x,i})_{i\in[k_x]}$ and $(\ell_{x,i})_{i\in[k_x]}$ are inductively defined as follows:
  - for $i = k_x$,
  $$\mathbb{t}_{x,i} = \frac{\mathsf{parseInt}(w_{x,k_x})(10^{|w_{x,k_x}|\#_{x,k_x}} - 1)}{(10^{|w_{x,k_x}|} - 1)}$$
  and $\ell_{x,i} = |w_{x,k_x}|\#_{x,k_x}$,
  - for $i \in [k_x - 1]$,
  $$\mathbb{t}_{x,i} = \frac{\mathsf{parseInt}(w_{x,i})(10^{|w_{x,i}|\#_{x,i}} - 1)10^{\ell_{x,i+1}}}{(10^{|w_{x,i}|} - 1)} + \mathbb{t}_{x,i+1}$$
  and $\ell_{x,i} = |w_{x,i}|\#_{x,i} + \ell_{x,i+1}$.

  Notice that here $|w_{x,\_}|$ and $\ell_{x,\_}$ are constants while $\#_{x,\_}$ are (flattening) variables.
- $\mathit{flatten}_{\mathcal{F}_\varphi}(\mathsf{len}(x)) \hat{=} \sum\limits_{i\in[k_x]} |w_{x,i}|\#_{x,i}$.

Example 3 Suppose $\mathsf{parseInt}(x) = 2\mathbb{x}$ is an atomic constraint and $\mathcal{F}_\varphi(x) = 1^*2^*$. Then

$$\begin{aligned}
&\mathit{flatten}_{\mathcal{F}_\varphi}(\mathsf{parseInt}(x) = 2\mathbb{x}) \\
\hat{=}\ & 1\tfrac{10^{\#_{x,1}}-1}{10-1}10^{\#_{x,2}} + 2\tfrac{10^{\#_{x,2}}-1}{10-1} = 2\mathbb{x} \\
\equiv\ & 10^{\#_{x,1}+\#_{x,2}} - 10^{\#_{x,2}} + 2 * 10^{2\#_{x,2}} - 2 = 18\mathbb{x} \\
\equiv\ & 10^{\#_{x,1}+\#_{x,2}} + 10^{\#_{x,2}} = 18\mathbb{x} + 2.
\end{aligned}$$

By the above reduction, and the decidability of ExpPA's satisfiability (see Theorem 2), we have

Theorem 1 *The satisfiability of* $\mathsf{STR}_{\mathsf{parseInt}}$ *under flat domain restrictions is equivalent to the satisfiability of existential* ExpPA *formulas, and thus is decidable.*

4 Decision procedure for ExpPA

Semënov first proved that ExpPA admits quantifier elimination in [21], thus its satisfiability problem is decidable. However, Semënov did not give a concrete quantifier elimination procedure. Remedying this, Point proposed a quantifier elimination procedure for ExpPA in [22].

In this section, we describe how Point's quantifier elimination procedure works.

Theorem 2 ([22]) ExpPA *admits quantifier elimination.*

Compared to [22], the presentation here is more accessible to computer science researchers. Moreover, the procedure presented here slightly improves Point's procedure, in the following two aspects: 1) DNF (disjunctive normal form) was required in Point's procedure, which is not required here, 2) in Point's procedure, the divisibility constraints produced by the elimination of linear occurrences of a variable should be converted to equality constraints (by introducing fresh variables) before the elimination of exponential occurrences of some other variable, which is unnecessary here, since the divisibility constraints are directly dealt with in the elimination of exponential occurrences of variables.

However, in the next section we analyse the complexity of Point's procedure to be 3-EXPSPACE, which is quite expensive and a faithful implementation would not scale[3]. So in Section 6, we will propose various optimizations to Point's algorithm, aiming at an efficient implementation.

As $\forall \mathbb{x}.\ \varphi$ is equivalent to $\neg\exists \mathbb{x}.\ \neg\varphi$, thus in the sequel, we only need to show that every ExpPA formula $\exists \mathbb{x}.\ \varphi \in$ ExpPA, where φ is quantifier-free, can be transformed into an equivalent quantifier-free formula $\varphi' \in$ ExpPA.

Before a formal description of the quantifier elimination procedure, let us use a simple example to illustrate the main idea and give an overview of the procedure.

4.1 An overview of the quantifier elimination procedure

Consider $\varphi \hat{=} \exists \mathbb{x}_2.\ 10^{\mathbb{x}_1+\mathbb{x}_2} - 10^{\mathbb{x}_2} \leq \mathbb{y} + 1001$.

At first, we *normalize* φ by introducing a fresh variable $\mathbb{x}_3$ for $\mathbb{x}_1 + \mathbb{x}_2$ and get the formula

$$\varphi_1 \hat{=} \exists \mathbb{x}_3 \exists \mathbb{x}_2.\ 10^{\mathbb{x}_3} - 10^{\mathbb{x}_2} \leq \mathbb{y} + 1001 \wedge \mathbb{x}_3 = \mathbb{x}_1 + \mathbb{x}_2.$$

Then, we enumerate different *orders* of the quantified variables, i.e. $\mathbb{x}_2$ and $\mathbb{x}_3$. Since $\mathbb{x}_3 = \mathbb{x}_1 + \mathbb{x}_2$, there is only one possible order, that is, $\mathbb{x}_3 \geq \mathbb{x}_2$.

Next, we illustrate how to eliminate the quantifier $\exists \mathbb{x}_3$, assuming $\mathbb{x}_3 \geq \mathbb{x}_2$. The elimination of $\exists \mathbb{x}_2$ is similar and simpler, thus omitted.

The elimination of $\exists \mathbb{x}_3$ consists of two steps, namely, eliminating the exponential occurrences of $\mathbb{x}_3$ first, and the linear occurrences next.

[3] We did implement Point's algorithm and discovered that the implementation could only solve formulas of very small size.

The main idea of the elimination of the exponential occurrences of $\mathbb{x}_3$ is to observe that if $\mathbb{x}_3 \geq \ell_{10}(\mathbb{y}+1001)+2$ and $\mathbb{x}_3 \geq \mathbb{x}_2+3$, then $10^{\mathbb{x}_3} - 10^{\mathbb{x}_2}$ is dominated by $10^{\mathbb{x}_3}$, that is, $10^{\mathbb{x}_3} - 10^{\mathbb{x}_2} \geq 10^{\mathbb{x}_3} - 10^{\mathbb{x}_3-3} = (1-10^{-3})10^{\mathbb{x}_3} \geq 10^{\ell_{10}(\mathbb{y}+1001)+1} = 10\lambda_{10}(\mathbb{y}+1001) > \mathbb{y}+1001$ (see Lemma 1 for the choice of the constants 2 and 3 in $\mathbb{x}_3 \geq \ell_{10}(\mathbb{y}+1001)+2$ and $\mathbb{x}_3 \geq \mathbb{x}_2+3$.). Therefore, a necessary condition for $10^{\mathbb{x}_3} - 10^{\mathbb{x}_2} \leq \mathbb{y}+1001$ is that either $\mathbb{x}_3 \leq \ell_{10}(\mathbb{y}+1001)+1$ or $\mathbb{x}_3 \leq \mathbb{x}_2+2$ holds.

- If $\mathbb{x}_3 \leq \ell_{10}(\mathbb{y}+1001)+1$, then we distinguish between whether $\mathbb{x}_3 \leq \ell_{10}(\mathbb{y}+1001)$ or $\mathbb{x}_3 = \ell_{10}(\mathbb{y}+1001)+1$.
  - If $\mathbb{x}_3 \leq \ell_{10}(\mathbb{y}+1001)$, then $10^{\mathbb{x}_3} - 10^{\mathbb{x}_2} \leq 10^{\ell_{10}(\mathbb{y}+1001)} = \lambda_{10}(\mathbb{y}+1001) \leq \mathbb{y}+1001$. In this case, $10^{\mathbb{x}_3} - 10^{\mathbb{x}_2} \leq \mathbb{y}+1001 \wedge \mathbb{x}_3 = \mathbb{x}_1+\mathbb{x}_2$ can simplified into `true`.
  - If $\mathbb{x}_3 = \ell_{10}(\mathbb{y}+1001)+1$, then $10^{\mathbb{x}_3} - 10^{\mathbb{x}_2} \leq \mathbb{y}+1001$ can be turned into $10^{\ell_{10}(\mathbb{y}+1001)+1} - 10^{\mathbb{x}_2} \leq \mathbb{y}+1001 \equiv 10\lambda_{10}(\mathbb{y}+1001) - 10^{\mathbb{x}_2} \leq \mathbb{y}+1001$.
- If $\mathbb{x}_3 \leq \mathbb{x}_2+2$, then $\mathbb{x}_3 = \mathbb{x}_2+j$ for $j \in \{0,1,2\}$. Thus $10^{\mathbb{x}_3} - 10^{\mathbb{x}_2} \leq \mathbb{y}+1001$ can be transformed to $\bigvee\limits_{j\in\{0,1,2\}} 10^{\mathbb{x}_2+j} - 10^{\mathbb{x}_2} \leq \mathbb{y}+1001$.

To summarize, φ_1 is transformed into

$$\begin{array}{l}
\varphi_2 \hat{=} \exists \mathbb{x}_3 \exists \mathbb{x}_2. \\
\left(\begin{array}{l}
\mathbb{x}_3 \leq \ell_{10}(\mathbb{y}+1001) \vee \\
\left(\begin{array}{l}
\mathbb{x}_3 = \ell_{10}(\mathbb{y}+1001)+1 \wedge \\
10\lambda_{10}(\mathbb{y}+1001) - 10^{\mathbb{x}_2} \leq \mathbb{y}+1001
\end{array}\right) \vee \\
\bigvee\limits_{j\in\{0,1,2\}} \left(\mathbb{x}_3 = \mathbb{x}_2+j \wedge 10^{\mathbb{x}_2+j} - 10^{\mathbb{x}_2} \leq \mathbb{y}+1001\right)
\end{array}\right) \wedge \\
\mathbb{x}_3 = \mathbb{x}_1+\mathbb{x}_2.
\end{array}$$

Note that φ_2 contains *only linear* occurrences of $\mathbb{x}_3$.

Finally, we can eliminate the linear occurrences of $\mathbb{x}_3$, thus the quantifier $\exists \mathbb{x}_3$, by applying the quantifier elimination algorithm of PA, e.g. Cooper's algorithm in [23]. The elimination of $\mathbb{x}_3$ in φ_2 here is simple, with $\mathbb{x}_3$ replaced by $\mathbb{x}_1+\mathbb{x}_2$.

In the remainder of this section, we are going to describe the aforementioned steps of the decision procedures: Normalization, the enumeration of the variable orders, the elimination of the exponential occurrences of variables. The elimination of the linear occurrences of variables is essentially the quantifier elimination of the PA and omitted.

Let us assume that $\varphi \hat{=} \exists \mathbb{x}.\ \varphi'(\mathbb{x}, \vec{\mathbb{y}})$, where φ' is a quantifier-free formula.

4.2 Normalization

The normalization step comprises the following sub-steps.

1. *NNF transformation* At first, we transform $\varphi'(\mathbb{x}, \vec{\mathbb{y}})$ into the NNF (negation normal form). Moreover, we remove the occurrences of $\neg$ by replacing (a) $\neg c|\mathbb{t}$ with $c \nmid \mathbb{t}$, (b) $\neg(\mathbb{t}_1 = \mathbb{t}_2)$ with $\mathbb{t}_1 < \mathbb{t}_2 \vee \mathbb{t}_2 < \mathbb{t}_1$, (c) $\neg(\mathbb{t}_1 < \mathbb{t}_2)$ with $\mathbb{t}_2 \leq \mathbb{t}_1$, (d) $\neg(\mathbb{t}_1 \leq \mathbb{t}_2)$ with $\mathbb{t}_2 < \mathbb{t}_1$, and so on.

2. *replace* $\ell_{10}(\mathbb{t})$ *terms* Repeat the following procedure, until there are no $\ell_{10}(\mathbb{t})$ with $\mathbb{x}$ occurs in $\mathbb{t}$: for each occurrence of $\ell_{10}(\mathbb{t})$ such that $\mathbb{x}$ occurs in $\mathbb{t}$, introduce a fresh variable, say $\mathbb{z}$, and replace all occurrences of $\ell_{10}(\mathbb{t})$ by $\mathbb{z}$, moreover, add the constraint $10^{\mathbb{z}} \leq \mathbb{t} < 10^{\mathbb{z}+1}$ as a conjunct. Note that if $\mathbb{t}$ contains no variables, then $\ell_{10}(\mathbb{t})$ is a constant. In this case, we can also assume that $\mathbb{t}$ contains $\mathbb{x}$ and perform the same replacements, which helps in the analysis of complexity in 5.1. Let the resulting formula be φ''.
3. *flatten* $10^{\mathbb{t}}$ *terms* Then repeat the following procedure to φ'', until for each occurrence of $10^{\mathbb{t}}$ with $\mathbb{x}$ occurs in $\mathbb{t}$, we have $\mathbb{t} = \mathbb{x}$: For each occurrence of the $10^{\mathbb{t}}$ in φ'', such that $\mathbb{t}$ contains $\mathbb{x}$ but is not $\mathbb{x}$, introduce a fresh variable, say $\mathbb{z}$, and replace all occurrences of $10^{\mathbb{t}}$ by $10^{\mathbb{z}}$, moreover, add the constraint $\mathbb{z} = \mathbb{t}$ as a conjunct. Let φ''' denote the resulting formula.
4. $\leq$ *transformation* Do the following replacements to φ''', so that all the atomic formulas in $\varphi^{\dagger}$ are of the form $\mathbb{t}_1 \leq \mathbb{t}_2$,$c|\mathbb{t}$ or $c \nmid \mathbb{t}$: Replace every occurrence of $\mathbb{t}_1 \geq \mathbb{t}_2$ with $\mathbb{t}_2 \leq \mathbb{t}_1$. Replace every occurrence of $\mathbb{t}_1 < \mathbb{t}_2$ (resp. $\mathbb{t}_1 > \mathbb{t}_2$) with $\mathbb{t}_1 \leq \mathbb{t}_2 - 1$ (resp. $\mathbb{t}_2 \leq \mathbb{t}_1 - 1$). Replace ever occurrence of $\mathbb{t}_1 = \mathbb{t}_2$ wtih $\mathbb{t}_2 \leq \mathbb{t}_1 \wedge \mathbb{t}_1 \leq \mathbb{t}_2$. Let $\varphi^{\dagger}$ the resulting formula.
5. Let $\vec{\mathbb{z}} = \mathbb{z}_1, \ldots, \mathbb{z}_n$ be an enumeration of the freshly introduced variables. Then the result of the normalization procedure is $\exists \vec{\mathbb{z}} \exists \mathbb{x}.\ \varphi^{\dagger}$.

Intuitively, the normalization step first absorbs all negation operators by transforming the formula into NNF. Then it removes the occurrences of $\ell_{10}(\mathbb{t})$ where $\mathbb{x}$ occurs in $\mathbb{t}$, by encoding them with the exponential function. Moreover, for each occurrence of $10^{\mathbb{t}}$ such that $\mathbb{x}$ occurs in $\mathbb{t}$, it introduces a fresh variable $\mathbb{z}$, replaces $10^{\mathbb{t}}$ with $10^{\mathbb{z}}$, and adds the equality $\mathbb{z} = \mathbb{t}$. All equalities and inequalities will be rewritten into the form $\mathbb{t}_1 \leq \mathbb{t}_2$. Finally, add quantifiers for the introduced fresh variables.

After the normalization, the resulting formula is of the following shape: 1) it is in NNF (negation normal form), 2) it contains no occurrences of $\ell_{10}(\mathbb{t})$ such that $\mathbb{x}$ occurs in $\mathbb{t}$, 3) it contains no occurrences of $10^{\mathbb{t}}$ such that $\mathbb{x}$ occurs in $\mathbb{t}$, but $\mathbb{t} \neq \mathbb{x}$, 4) all the atomic formulas are of the form $\mathbb{t}_1 \leq \mathbb{t}_2$ or $c|\mathbb{t}$. Denote the negation of $c|\mathbb{t}$ by $c \nmid t$, then the formula contains no negation symbol.

4.3 Enumeration of the variable orders

Suppose $n-1$ fresh variables are introduced in the normalization procedure, rename the original variable $\mathbb{x}$ and the $n-1$ introduced variables $\mathbb{x}_i, 1 \leq i \leq n$. Let the output of the normalization procedure be $\exists \vec{\mathbb{x}}.\ \varphi'$ with $\vec{\mathbb{x}} = (\mathbb{x}_1, \ldots, \mathbb{x}_n)$. We then enumerate all the linear orders of $\{\mathbb{x}_1, \ldots, \mathbb{x}_n\}$. Each linear order can be represented by a permutation $\sigma \in \mathcal{S}_n$ (where $\mathcal{S}_n$ is the permutation group on $[n]$), with the intention that $\mathbb{x}_{\sigma(n)} \geq \cdots \geq \mathbb{x}_{\sigma(1)}$.

Assuming a linear order $\sigma \in \mathcal{S}_n$ of $\{\mathbb{x}_1, \ldots, \mathbb{x}_n\}$, we then consider $\varphi'_\sigma = \exists \vec{\mathbb{x}}.\ \varphi' \wedge \bigwedge_{i \in [n-1]} \mathbb{x}_{\sigma(i)} \leq \mathbb{x}_{\sigma(i+1)}$ and eliminate the quantifiers $\exists \mathbb{x}_{\sigma(n)}$, …, $\exists \mathbb{x}_{\sigma(1)}$, one by one and from $\mathbb{x}_n$ to $\mathbb{x}_1$. Let φ''_σ denote the resulting formula.

Finally, $\exists \vec{\mathbb{x}}.\ \varphi'$ is transformed into the quantifier-free formula $\bigvee_{\sigma \in \mathcal{S}_n} \varphi''_\sigma$.

In the sequel, assuming a linear order $\sigma \in \mathcal{S}_n$, for $i \in [n]$, let $\exists \mathbb{x}_{\sigma(1)} \ldots \exists \mathbb{x}_{\sigma(i)}.\ \varphi''_{\sigma,i}$ be the formula obtained from φ'_σ by eliminating the quantifiers $\exists \mathbb{x}_{\sigma(n)}$, ..., $\exists \mathbb{x}_{\sigma(i+1)}$, we show how to eliminate the exponential occurrences of $\mathbb{x}_{\sigma(i)}$ in $\exists \mathbb{x}_{\sigma(1)} \ldots \exists \mathbb{x}_{\sigma(i)}.\ \varphi''_{\sigma,i}$. We would like to remark that the linear occurrences of $\mathbb{x}_{\sigma(i)}$ should be eliminated further so that the quantifier $\exists \mathbb{x}_{\sigma(i)}$ can be eliminated. The elimination of linear occurrences of $\mathbb{x}_{\sigma(i)}$ is essentially the quantifier elimination algorithm of PA.

Note that the order $\mathbb{x}_{\sigma(i)} \geq \cdots \geq \mathbb{x}_{\sigma(1)}$ guarantees the maximality of $\mathbb{x}_{\sigma(i)}$ among $\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)}$, which is essential for the elimination of $10^{\mathbb{x}_{\sigma(i)}}$ from $\varphi''_{\sigma,i}$ (see Lemma 1).

4.4 Elimination of exponential occurrences of variables

Let $i \in [n]$ and $\exists \mathbb{x}_{\sigma(1)} \ldots \exists \mathbb{x}_{\sigma(i)}.\ \varphi''_{\sigma,i}(\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)}, \vec{\mathbb{y}})$ be the formula obtained from φ'_σ by eliminating the quantifiers $\exists \mathbb{x}_{\sigma(n)}$, ..., $\exists \mathbb{x}_{\sigma(i+1)}$. We show how to eliminate the exponential occurrences of $\mathbb{x}_{\sigma(i)}$ in $\varphi''_{\sigma,i}$. The elimination is *local* in the sense that it is applied to the atomic formulas independently.

Recall that after normalization, the atomic formulas are of the form $\mathbb{t}_1 \leq \mathbb{t}_2$, $c | \mathbb{t}$ or $c \nmid \mathbb{t}$. Therefore, we can assume that the atomic formulas in $\varphi''_{\sigma,i}$ are of the form $a_i 10^{\mathbb{x}_{\sigma(i)}} + \sum_{j=1}^{i-1} a_j 10^{\mathbb{x}_{\sigma(j)}} + \sum_{k=1}^{i} b_k \mathbb{x}_{\sigma(k)} \leq \mathbb{t}(\vec{\mathbb{y}})$ or $c \mid \left(a_i 10^{\mathbb{x}_{\sigma(i)}} + \sum_{j=1}^{i-1} a_j 10^{\mathbb{x}_{\sigma(j)}} + \sum_{k=1}^{i} b_k \mathbb{x}_{\sigma(k)} + \mathbb{t}(\vec{\mathbb{y}})\right)$ (or $\nmid$).

4.4.1 Inequality atoms

In the sequel, we illustrate how to eliminate the exponential occurrences of $\mathbb{x}_{\sigma(i)}$ for these inequality atomic formulas. Let us consider

$$\begin{array}{l}\tau(\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)}, \vec{\mathbb{y}}) \hat{=} \\ a_i 10^{\mathbb{x}_{\sigma(i)}} + \sum_{j=1}^{i-1} a_j 10^{\mathbb{x}_{\sigma(j)}} + \sum_{k=1}^{i} b_k \mathbb{x}_{\sigma(k)} \leq \mathbb{t}(\vec{\mathbb{y}}).\end{array}$$

The elimination of the exponential occurrences of $\mathbb{x}_{\sigma(i)}$ in $\tau(\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)}, \vec{\mathbb{y}})$ relies on the following lemma. Intuitively, the lemma states the fact that if $\mathbb{x}_{\sigma(i)}$ is sufficiently greater than $\mathbb{x}_{\sigma(i-1)}$, then the left-hand-side of $\tau(\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)}, \vec{\mathbb{y}})$ is *dominated* by $a_i 10^{\mathbb{x}_{\sigma(i)}}$, moreover, if $a_i > 0$ and the value of $\mathbb{x}_{\sigma(i)}$ is sufficiently small (resp. big), then $\tau(\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)}, \vec{\mathbb{y}})$ holds (resp. does not hold), similarly for $a_i < 0$.

Lemma 1 Let

$$\begin{array}{l}\tau(\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)}, \vec{\mathbb{y}}) \hat{=} \\ a_i 10^{\mathbb{x}_{\sigma(i)}} + \sum_{j=1}^{i-1} a_j 10^{\mathbb{x}_{\sigma(j)}} + \sum_{k=1}^{i} b_k \mathbb{x}_{\sigma(k)} \leq \mathbb{t}(\vec{\mathbb{y}}).\end{array}$$

with $a_i \neq 0$, $A \hat{=} \sum_{j=1}^{i-1} |a_j|$, $B \hat{=} 2(\ell_{10}(\sum_{j=1}^{i} |b_j|) + 3)$, and $\delta \hat{=} \ell_{10}(A) + 3$.

- If $a_i > 0$, let $\alpha(\vec{\mathbb{y}}) \hat{=} \ell_{10}(\mathbb{t}(\mathbb{y})) - \ell_{10}(a_i)$, then
  - if $\mathbb{x}_{\sigma(i)} \leq \alpha(\vec{\mathbb{y}}) - 1$, $\mathbb{x}_{\sigma(i)} \geq B$ and $\mathbb{x}_{\sigma(i)} \geq \mathbb{x}_{\sigma(i-1)} + \delta$, then $\tau(\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)}, \vec{\mathbb{y}})$ holds,

- if $\mathbb{x}_{\sigma(i)} \geq \alpha(\vec{\mathbb{y}}) + 2$, $\mathbb{x}_{\sigma(i)} \geq B$ and $\mathbb{x}_{\sigma(i)} \geq \mathbb{x}_{\sigma(i-1)} + \delta$, then $\tau(\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)}, \vec{\mathbb{y}})$ **does not** hold.

- If $a_i < 0$, let $\alpha(\vec{\mathbb{y}}) \hat{=} \ell_{10}(-\mathbb{t}(\mathbb{y})) - \ell_{10}(-a_i)$, then
  - if $\mathbb{x}_{\sigma(i)} \leq \alpha(\vec{\mathbb{y}}) - 1$, $\mathbb{x}_{\sigma(i)} \geq B$ and $\mathbb{x}_{\sigma(i)} \geq \mathbb{x}_{\sigma(i-1)} + \delta$, then $\tau(\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)}, \vec{\mathbb{y}})$ **does not** hold,
  - if $\mathbb{x}_{\sigma(i)} \geq \alpha(\vec{\mathbb{y}}) + 2$, $\mathbb{x}_{\sigma(i)} \geq B$ and $\mathbb{x}_{\sigma(i)} \geq \mathbb{x}_{\sigma(i-1)} + \delta$, then $\tau(\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)}, \vec{\mathbb{y}})$ holds.

We need the following proposition for the proof of Lemma 1.

Proposition 3 *If $n \geq m \geq 1$ and $p \geq 2(\ell_{10}(n) - \ell_{10}(m) + 1)$, then $np \leq m10^p$ holds.*

Proof First we show that for any $n' \in \mathbb{N}$, if $p \geq 2n'$, then $10^p \geq 10^{n'}10^{(p-n')} \geq 10^{n'}10(p-n') \geq 10^{n'}(5p + 5(p-2n')) \geq 10^{n'}p$.

If $p \geq 2(\ell_{10}(n) - \ell_{10}(m) + 1)$, then $np \leq 10\lambda_{10}(n)p = 10 * 10^{\ell_{10}(n)}p = 10^{\ell_{10}(n)-\ell_{10}(m)+1}10^{\ell_{10}(m)}p$.

Because $p \geq 2(\ell_{10}(n) - \ell_{10}(m) + 1)$, we deduce that $10^{\ell_{10}(n)-\ell_{10}(m)+1}p \leq 10^p$. Therefore, $10^{\ell_{10}(n)-\ell_{10}(m)+1}10^{\ell_{10}(m)}p \leq 10^{\ell_{10}(m)}10^p \leq m10^p$. We conclude that $np \leq m10^p$. □

Proof of Lemma 1 We only prove for the case $a_i > 0$, the other case is symmetric.

Let $A \hat{=} \sum_{j=1}^{i-1} |a_j|$, $B \hat{=} 2(\ell_{10}(\sum_{j=1}^{i} |b_j|) + 3)$, and $\delta \hat{=} \ell_{10}(A) + 3$.

Suppose $\mathbb{x}_{\sigma(i)} \geq B$ and $\mathbb{x}_{\sigma(i)} \geq \mathbb{x}_{\sigma(i-1)} + \delta$. Moreover, let $\alpha(\vec{\mathbb{y}}) \hat{=} \ell_{10}(\mathbb{t}(\mathbb{y})) - \ell_{10}(a_i)$. Note that

$$A10^{-\delta} = A10^{-\ell_{10}(A)-3} = \frac{A}{1000\lambda_{10}(A)} \leq \frac{1}{100}. \tag{1}$$

From $\mathbb{x}_{\sigma(i)} \geq \mathbb{x}_{\sigma(i-1)} + \delta$ and $\mathbb{x}_{\sigma(i-1)} \geq \ldots \geq \mathbb{x}_{\sigma(1)}$, we know

$$-A10^{\mathbb{x}_{\sigma(i)}-\delta} \leq \sum_{j=1}^{i-1} a_j 10^{\mathbb{x}_{\sigma(j)}} \leq A10^{\mathbb{x}_{\sigma(i)}-\delta}$$

and

$$-(\sum_{j=1}^{i} |b_j|)\mathbb{x}_{\sigma(i)} \leq \sum_{k=1}^{i} b_k \mathbb{x}_{\sigma(k)} \leq (\sum_{j=1}^{i} |b_j|)\mathbb{x}_{\sigma(i)}.$$

Moreover, let $n = 100\sum_{j=1}^{i} |b_j|$, $m = 1$, and $p = \mathbb{x}_{\sigma(i)}$, then $n \geq m \geq 1$. From $2(\ell_{10}(n) - \ell_{10}(m) + 1) = 2(\ell_{10}(100\sum_{j=1}^{i} |b_j|) + 1) = 2(\ell_{10}(\sum_{j=1}^{i} |b_j|) + 3) = B$, we deduce that $p = \mathbb{x}_{\sigma(i)} \geq B = 2(\ell_{10}(n) - \ell_{10}(m) + 1)$. Then according to Proposition 3,

$$100(\sum_{j=1}^{i} |b_j|)\mathbb{x}_{\sigma(i)} = np \leq m10^p = 10^{\mathbb{x}_{\sigma(i)}}.$$

Thus $(\sum_{j=1}^{i} |b_j|)\mathbb{x}_{\sigma(i)} \leq \frac{1}{100}10^{\mathbb{x}_{\sigma(i)}}$.

If $\mathbb{x}_{\sigma(i)} \geq \alpha(\vec{\mathbb{y}}) + 2$, then

$a_i 10^{\mathbb{x}_{\sigma(i)}} + \sum_{j=1}^{i-1} a_j 10^{\mathbb{x}_{\sigma(j)}} + \sum_{k=1}^{i} b_k \mathbb{x}_{\sigma(k)} \geq$
$a_i 10^{\mathbb{x}_{\sigma(i)}} - A 10^{\mathbb{x}_{\sigma(i)}-\delta} - (\sum_{j=1}^{i} |b_j|)\mathbb{x}_{\sigma(i)} \geq$
$a_i 10^{\mathbb{x}_{\sigma(i)}} - \frac{1}{100} 10^{\mathbb{x}_{\sigma(i)}} - \frac{1}{100} 10^{\mathbb{x}_{\sigma(i)}} =$
$(a_i - \frac{1}{50}) 10^{\mathbb{x}_{\sigma(i)}} \geq (a_i - \frac{1}{50}) 10^{\alpha(\vec{\mathbb{y}})+2} =$
$(a_i - \frac{1}{50}) 10^{\ell_{10}(\mathbb{t}(\mathbb{y})) - \ell_{10}(a_i) + 2} =$
$\frac{10(a_i - \frac{1}{50})}{10^{\ell_{10}(a_i)}} 10^{\ell_{10}(\mathbb{t}(\mathbb{y}))+1} \geq \frac{10(a_i - \frac{1}{50})}{a_i} 10^{\ell_{10}(\mathbb{t}(\mathbb{y}))+1} \geq$
$(10 - \frac{1}{5a_i}) 10^{\ell_{10}(\mathbb{t}(\mathbb{y}))+1} > 10^{\ell_{10}(\mathbb{t}(\mathbb{y}))+1} \geq \mathbb{t}(\mathbb{y})$.

Therefore, in this case, $\tau(\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)}, \vec{\mathbb{y}})$ *does not* hold.

If $\mathbb{x}_{\sigma(i)} \leq \alpha(\vec{\mathbb{y}}) - 1$, then

$a_i 10^{\mathbb{x}_{\sigma(i)}} + \sum_{j=1}^{i-1} a_j 10^{\mathbb{x}_{\sigma(j)}} + \sum_{k=1}^{i} b_k \mathbb{x}_{\sigma(k)} \leq$
$a_i 10^{\mathbb{x}_{\sigma(i)}} + A 10^{\mathbb{x}_{\sigma(i)}-\delta} + (\sum_{j=1}^{i} |b_j|)\mathbb{x}_{\sigma(i)} \leq$
$a_i 10^{\mathbb{x}_{\sigma(i)}} + \frac{A}{10^\delta} 10^{\mathbb{x}_{\sigma(i)}} + \frac{1}{100} 10^{\mathbb{x}_{\sigma(i)}} \leq$
$(a_i + \frac{1}{100} + \frac{1}{100}) 10^{\mathbb{x}_{\sigma(i)}} = (a_i + \frac{1}{50}) 10^{\mathbb{x}_{\sigma(i)}} \leq$
$(a_i + \frac{1}{50}) 10^{\alpha(\vec{\mathbb{y}})-1} = (a_i + \frac{1}{50}) 10^{\ell_{10}(\mathbb{t}(\mathbb{y})) - \ell_{10}(a_i) - 1} =$
$\frac{a_i + \frac{1}{50}}{10^{\ell_{10}(a_i)+1}} 10^{\ell_{10}(\mathbb{t}(\mathbb{y}))} = \frac{a_i + \frac{1}{50}}{10\lambda_{10}(a_i)} 10^{\ell_{10}(\mathbb{t}(\mathbb{y}))} \leq$
$\frac{a_i + \frac{1}{50}}{a_i + 1} 10^{\ell_{10}(\mathbb{t}(\mathbb{y}))} \leq 10^{\ell_{10}(\mathbb{t}(\mathbb{y}))} \leq \mathbb{t}(\mathbb{y})$.

Therefore, in this case, $\tau(\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)}, \vec{\mathbb{y}})$ holds.

$\square$

We would like to remark that Lemma 1 still holds when the base of exponential function is changed to any natural number $n \geq 2$.

If $a_i > 0$, then the exponential occurrences of $\mathbb{x}_{\sigma(i)}$ in $\tau(\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)}, \vec{\mathbb{y}})$ can be eliminated by utilizing Lemma 1 and enumerating the constraints on $\mathbb{x}_{\sigma(i)}$ and $\mathbb{x}_{\sigma(i-1)}$. Specifically, $\tau(\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)}, \vec{\mathbb{y}})$ is equivalent to

$$
\begin{aligned}
& \bigvee_{p=0}^{B-1} a_i 10^p + \sum_{j=1}^{i-1} a_j 10^{\mathbb{x}_{\sigma(j)}} + b_i p + \sum_{k=1}^{i-1} b_k \mathbb{x}_{\sigma(k)} \leq \mathbb{t}(\vec{\mathbb{y}}) \\
& \bigvee \left(\mathbb{x}_{\sigma(i)} \geq B \wedge \mathbb{x}_{\sigma(i)} \leq \alpha(\vec{\mathbb{y}}) - 1 \wedge \mathbb{x}_{\sigma(i)} \geq \mathbb{x}_{\sigma(i-1)} + \delta\right) \\
& \bigvee \bigvee_{p=0}^{\delta-1} \begin{pmatrix} \mathbb{x}_{\sigma(i)} \geq B \wedge \mathbb{x}_{\sigma(i)} \leq \alpha(\vec{\mathbb{y}}) - 1 \wedge \\ \mathbb{x}_{\sigma(i)} = \mathbb{x}_{\sigma(i-1)} + p \wedge \\ \tau(\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)}, \vec{\mathbb{y}})[\mathbb{x}_{\sigma(i-1)} + p / \mathbb{x}_{\sigma(i)}] \end{pmatrix} \\
& \bigvee \begin{pmatrix} \mathbb{x}_{\sigma(i)} \geq B \wedge \mathbb{x}_{\sigma(i)} = \alpha(\vec{\mathbb{y}}) \wedge \\ \tau(\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)}, \vec{\mathbb{y}})[\alpha(\vec{\mathbb{y}}) / \mathbb{x}_{\sigma(i)}] \end{pmatrix} \\
& \bigvee \begin{pmatrix} \mathbb{x}_{\sigma(i)} \geq B \wedge \mathbb{x}_{\sigma(i)} = \alpha(\vec{\mathbb{y}}) + 1 \wedge \\ \tau(\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)}, \vec{\mathbb{y}})[\alpha(\vec{\mathbb{y}}) + 1 / \mathbb{x}_{\sigma(i)}] \end{pmatrix} \\
& \bigvee \bigvee_{p=0}^{\delta-1} \begin{pmatrix} \mathbb{x}_{\sigma(i)} \geq B \wedge \mathbb{x}_{\sigma(i)} \geq \alpha(\vec{\mathbb{y}}) + 2 \wedge \\ \mathbb{x}_{\sigma(i)} = \mathbb{x}_{\sigma(i-1)} + p \wedge \\ \tau(\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)}, \vec{\mathbb{y}})[\mathbb{x}_{\sigma(i-1)} + p / \mathbb{x}_{\sigma(i)}] \end{pmatrix},
\end{aligned}
$$

where the exponential occurrences of $\mathbb{x}_{\sigma(i)}$ disappear. The elimination of the exponential occurrences of $\mathbb{x}_{\sigma(i)}$ for the case $a_i < 0$ is similar.

4.4.2 Divisibility atoms

Consider a divisiblilty atomic formula

$$\begin{aligned}&\tau(\mathbb{x}_{\sigma(i)},\ldots,\mathbb{x}_{\sigma(1)},\vec{\mathbb{y}}) \hat{=} \\ &d \mid \left(a_i 10^{\mathbb{x}_{\sigma(i)}} + \sum_{j=1}^{i-1} a_j 10^{\mathbb{x}_{\sigma(j)}} + \sum_{k=1}^{i} b_k \mathbb{x}_{\sigma(k)} + \mathbb{t}(\vec{\mathbb{y}})\right)\end{aligned}$$

with $a_i \neq 0$. The indivisibility case can be treated analogously.

Let $d = 2^{r_1}5^{r_2}d_0$ such that d_0 is divisible by neither 2 nor 5. Moreover, let $r = \max(r_1, r_2)$. Then $d|(10^r d_0)$.

If $d_0 = 1$, then 10^r is divisible by $d = 2^{r_1}5^{r_2}$. Thus for every $n \geq r$, $d \mid 10^n$. Therefore, in this case, $\tau(\mathbb{x}_{\sigma(i)},\ldots,\mathbb{x}_{\sigma(1)},\vec{\mathbb{y}})$ is equivalent to

$$\begin{aligned}&\bigvee_{p=0}^{r-1} d \left| \left(a_i 10^p + \sum_{j=1}^{i-1} a_j 10^{\mathbb{x}_{\sigma(j)}} + b_k p + \sum_{k=1}^{i-1} b_k \mathbb{x}_{\sigma(k)} + \mathbb{t}(\vec{\mathbb{y}})\right)\right. \\ &\vee \left(\mathbb{x}_{\sigma(i)} \geq r \wedge d \left| \left(\sum_{j=1}^{i-1} a_j 10^{\mathbb{x}_{\sigma(j)}} + \sum_{k=1}^{i} b_k \mathbb{x}_{\sigma(k)} + \mathbb{t}(\vec{\mathbb{y}})\right)\right.\right),\end{aligned}$$

where the exponential occurrences of $\mathbb{x}_{\sigma(i)}$ disappear.

Next, let us assume $d_0 > 1$. Since 10 and d_0 are relatively prime, according to Euler's theorem (cf. [24]), $10^{\phi(d_0)} \equiv 1 \bmod d_0$, where ϕ is the Euler function. Suppose $10^{\phi(d_0)} = kd_0 + 1$ for some $k \in \mathbb{N}$. Then for every $n \in \mathbb{N}$ with $n \geq r$,

$$\begin{aligned}&10^{n+\phi(d_0)} \bmod d = 10^{n-r}10^r(kd_0+1) \bmod d = \\ &10^{n-r}(k10^r d_0 + 10^r) \bmod d = \\ &10^{n-r}(0 + 10^r) \bmod d = 10^n \bmod d.\end{aligned}$$

Then $\tau(\mathbb{x}_{\sigma(i)},\ldots,\mathbb{x}_{\sigma(1)},\vec{\mathbb{y}})$ is equivalent to

$$\begin{aligned}&\bigvee_{p=0}^{r-1} \tau(\mathbb{x}_{\sigma(i)},\ldots,\mathbb{x}_{\sigma(1)},\vec{\mathbb{y}})[p/\mathbb{x}_{\sigma(i)}] \vee \\ &\left(\begin{array}{l}\mathbb{x}_{\sigma(i)} \geq r \wedge \\ \bigvee_{q=0}^{\phi(d_0)-1} \left(\begin{array}{l}\phi(d_0) \mid (\mathbb{x}_{\sigma(i)} - r - q) \wedge \\ d \left| \left(\begin{array}{l} a_i 10^{r+q} + \sum_{j=1}^{i-1} a_j 10^{\mathbb{x}_{\sigma(j)}} + \\ \sum_{k=1}^{i} b_k \mathbb{x}_{\sigma(k)} + \mathbb{t}(\vec{\mathbb{y}})\end{array}\right)\right.\end{array}\right)\end{array}\right),\end{aligned}$$

where the exponential occurrences of $\mathbb{x}_{\sigma(i)}$ disappear.

5 Complexity Analysis

5.1 Complexity analysis

In this part, we analyse the complexity of the quantifier elimination procedure applied on a formula with one existential quantifer, that is, a formula of the form $\exists \mathbb{x}.\varphi(\mathbb{x},\vec{\mathbb{y}})$. We will prove that the complexity of eliminating one quantifer has a 3-EXPSPACE upper bound. Since the quantifier elimination procedure for PA works as a subprocedure (in eliminating linear occurrences of variables) and also has a 3-EXPSPACE complexity, the bound may not be easily improved.

We first analyse the changes of formula length in the normalization step. For the rest of the decision procedure, we adopt the style in the complexity analysis of PA (cf. [25]). The idea is that the upper bound of the formula length can be expressed using the product of the number of atoms, the number of coefficients and the length of the maximum constant. So we track the changes of such quantities throughout the procedure.

Given a formula $\exists \mathbb{x}.\varphi(\mathbb{x}, \vec{\mathbb{y}})$ with length n. After the normalization step, suppose we obtain a formula with m quantified variables $\exists \mathbb{x}_1 \ldots \exists \mathbb{x}_m.\varphi'(\mathbb{x}_1, \ldots, \mathbb{x}_m, \vec{\mathbb{y}})$. We show that the length of the new formula is at most $10n$ and the number of quantified variables $m \leq n$. In each sub-step of normalization, we analyse the worst situation: (1) *NNF transformation* suppose all atoms are of the form $\mathbb{t}_1 = \mathbb{t}_2$, then taking negation will double the number of atoms, the length increases to $2n$ at most. (2) *replace* $\ell_{10}(\mathbb{t})$ *terms* the original formula has at most n terms of the form $\ell_{10}(\mathbb{t})$, for each of them, we introduce a fresh variable and two conjuncts. The formula length increases to $4n$. (3) *flatten* $10^{\mathbb{t}}$ *terms* similar to (2), the formula has at most n such terms and for each term, a fresh variable and a conjunct are added. The formula length increases to $5n$. (4) $\leq$ *transformation* similar to (1), here we assume all atoms are of the form $\mathbb{t}_1 = \mathbb{t}_2$, so the length of the formula with increase to $10n$ at most.

The analysis for the normalization step is coarse, for example, the worst case in (1)&(4) or (2)&(3) can not happen at the same time. However, what we need is that the increased length of the formula is bounded by a constant factor,i.e. 10. In addition, we can also conclude that the number of fresh variables, m, is less than n since there are at most n different forms of terms.

After the normalization step, denote the obtained formula by $\exists \vec{\mathbb{x}}.\varphi(\vec{\mathbb{x}}, \vec{\mathbb{y}})$ with quantified variables $\vec{\mathbb{x}} = (\mathbb{x}_1, \ldots, \mathbb{x}_m)$ and free variables $\vec{\mathbb{y}}$. φ contains no quantifiers. Let n' denote the length of the formula.

According to our quantifier elimination procedure,we eliminate the quantified variables one by one: each time select the largest variable among $\mathbb{x}_i, 1 \leq i \leq m$, first eliminate the exponential occurrences, then linear occurrences. The linear order of quantified variables are given by a permutation. For m quantified variables, there are $m!$ possible linear orders.

Suppose the specified linear order is $\mathbb{x}_m \geq \cdots \geq \mathbb{x}_1$. Let $\exists \mathbb{x}_1 \ldots \exists \mathbb{x}_{m-k}.\varphi_k$ denote the formula obtained from $\exists \vec{\mathbb{x}}.\varphi(\vec{\mathbb{x}}, \vec{\mathbb{y}})$ by eliminating quantifiers $\exists \mathbb{x}_m, \ldots, \exists \mathbb{x}_{m-k+1}$. Let $\exists \mathbb{x}_1 \ldots \exists \mathbb{x}_{m-k+1}.\varphi'_k$ denote the formula obtained from $\exists \mathbb{x}_1 \ldots \exists \mathbb{x}_{m-k+1}.\varphi_{k-1}$ by eliminating linear occurrences of $\mathbb{x}_{m-k+1}$. Let φ_0 denote φ.

Let c_k be the number of distinct d in atoms of the form $d \mid \mathbb{t}$ plus the number of distinct coefficients of *linear occurrences* of quantified variables in φ_k. Let s_k be the largest constant (including coefficients) and a_k be the number of atomic formulas in φ_k. Similarly we define c'_k, s'_k and a'_k for φ'_k. Let c, s, a be c_0, s_0, a_0 respectively.

First we analyse the sub-procedure to eliminate exponential occurrences of $\mathbb{x}_m$. We prove the following lemma.

Lemma 2

$$c'_1 \leq c^2$$

$$s'_1 \leq ms^2$$
$$a'_1 \leq sa$$

Proof The analysis is separated into 2 cases where all atoms are of the same form (inequalities atoms or divisiblilty atoms).

If all atoms are inequalities atomic formulas of the form in Lemma 1. We know that each atomic formula τ with exponential occurrence of $\mathbb{x}_m$ is replaced by a new formula. Only the coefficients of linear occurrences of $\mathbb{x}_m$ and $\mathbb{x}_{m-1}$ will be changed: constant coefficient 1 is introduced, and if we substitute $\mathbb{x}_m$ by $\mathbb{x}_{m-1} + p$ for some p, coefficient of linear occurrence of $\mathbb{x}_{m-1}$ will become $b_m + b_{m-1}$ (b_m, b_{m-1} are coefficients for $\mathbb{x}_m$ and $\mathbb{x}_{m-1}$ in τ, see Lemma 1). Since the new coefficient is obtained by adding two linear coefficient together, we have $c'_1 \leq c^2$. Note that δ and B are at most $\ell_{10}(ms)$, when we substitute $\mathbb{x}_m$ by $\mathbb{x}_{m-1}+\delta-1$ or by $B - 1$, the largest constant in the formula is at most $s \cdot 10^{\ell_{10}(ms)} \leq ms^2$. And an inequality is replaced by at most $4\ell_{10}(ms)$ atomic formulas, so $a'_1 \leq 4\ell_{10}(ms)a$.

If all atoms are divisiblilty atomic formulas of the form $d \mid \mathbb{t}$ or $d \nmid \mathbb{t}$. We have $c'_1 < 2c$ since a divisibility atomic formula will produce at most two forms of atomic formulas $d \mid \mathbb{t}$ and $\phi(d) \mid \mathbb{t}$. Note that any constant in $\mathbb{t}$, say l, can be replaced by ($l \bmod d$), so $s'_1 \leq s$. When d is a large prime number, $\phi(d) = d - 1$, a divisibility atomic formula is replaced by roughly d atomic formulas, so $a'_1 \leq sa$.

Choose larger upper bound for c'_1, s'_1 and a'_1 respectively, then the lemma is proved.

□

Since linear occurrences of $\mathbb{x}_m$ are eliminated using Cooper's algorithm, we combine Oppen's analysis for this sub-procedure:

Lemma 3

$$c_1 \leq c'^4$$
$$s_1 \leq s'^{4c'}$$
$$a_1 \leq a'^4 s'^{2c'}$$

From Lemma 2 and Lemma 3, we have

$$c_1 \leq c^8$$
$$s_1 \leq (ms^2)^{4c^2}$$
$$a_1 \leq (sa)^4 (ms^2)^{2c^2}$$

Assuming $m \leq n$, by induction on k we get

Lemma 4

$$c_k \leq c^{8^k}$$
$$s_k \leq n^{(4c)^{8^k}} s^{(8c)^{8^k}}$$

$$a_k \le a^{4^k} n^{(4c)^{8^k}} s^{(8c)^{8^k}}$$

If we assume $c \le n$, $a \le n$ and $s \le n$, the space required to store the quantifier free formula, φ_k, is bounded by the product of the number of linear orders $m!$, the number of atoms a_k, the maximum number of constants $2m + 2$ per atom, the maximum amount of space s_k to store each constant and some constant q:

$$\mathsf{space} \le q \cdot m! \cdot a_k \cdot (2m + 2) \cdot s_k \le 2^{2^{2^{pn \log n}}}$$

for some large constant p.

REMARK: When the original formula has no alternating quantifiers, for example, the formula of the form $\exists \mathbb{x}_m \exists \mathbb{x}_{m-1} \ldots \exists \mathbb{x}_1 . \varphi(\mathbb{x}_1, \ldots, \mathbb{x}_{m-1}, \mathbb{x}_m, \vec{\mathbb{y}})$. Since we can carry out the normalization step for all quantified variable $\mathbb{x}_i (1 \le i \le m)$ at the same time and enumerate the linear order for all introduced variables, the complexity of eliminating all quantifers is still 3-EXPSPACE. However, if there are alternating quantifers at the very beginning, say the formula is of the form $\forall \mathbb{x}_2 \exists \mathbb{x}_1 . \varphi(\mathbb{x}_1, \mathbb{x}_2, \vec{\mathbb{y}})$. We have to eliminate the occurrences of $\mathbb{x}_1$ first, treating $\mathbb{x}_2$ as a free variable like $\vec{\mathbb{y}}$. When eliminating exponential occurrences of variables in inequalities (see Lemma 1), $\mathbb{x}_2$ will be collected in term $\mathbb{t}(\mathbb{y})$. Since $\mathbb{t}(\mathbb{y})$ may change due to the substitutions of $\mathbb{x}_1$, when we carry out normalization step for $\mathbb{x}_2$, the number of introduced fresh variables is not bounded by the length of original formula. Hence, we conjecture that the overall complexity for eliminating all quantifiers in a formula will be non-elementary.

6 Optimizations

In the last section, we illustrated the main idea of the decision procedure for ExpPA. The decision procedure has a high complexity in the sense that the elimination of the exponential occurrences of each variable incurs an exponential blow-up, similarly for linear occurrences. Note that this high complexity holds even for quantifier-free formulas containing exponential terms: For a quantifier-free formula φ containing exponential terms, we solve its satisfiability problem by adding the existential quantifiers for all the variables occurring in φ, then eliminate the quantifiers one by one, resulting into `true` or `false` in the end. The original formula φ is satisfiable if `true` is obtained in the end.

In this section, we focus on quantifier-free ExpPA formulas (or existential ExpPA formulas since they are satisfiability-equivalent), and present various optimizations of the quantifier elimination procedure for ExpPA, aiming at an efficient implementation. The focus on quantifier-free ExpPA formulas is motivated by the following two facts: 1) the flattening of $\mathsf{STR}_{\mathsf{parseInt}}$ constraints results into such formulas, 2) these formulas are already challenging for state-of-the-art SMT solvers (with exponential functions defined as recursive functions).

Let φ be a quantifier-free ExpPA formula in the remainder of this section. Moreover, we assume that φ is normalized since the optimizations presented in the sequel are for normalized formulas. Furthermore, for technical convenience, we assume that all the inequality atomic formulas are of the form $\sum_{j=1}^{n} a_j 10^{\mathbb{x}_j} + \sum_{k=1}^{n} b_k \mathbb{x}_k \leq c$, where c is an integer constant. (Implicitly, we assume that there are no free variables and all the variables are existentially quantified.)

In the sequel, we will explain two major optimizations in 6.1 and 6.2. Additional optimizations are listed in 6.3.

6.1 Reduce the number of enumerated variable orders by over approximation

Recall that in the decision procedure for ExpPA, after the normalization, the variable orders are enumerated and for each order, the exponential and linear occurrences of variables are eliminated. Since the quantifier elimination is expensive and applied to each possible order of variables, if we could reduce the candidate variable orders in the very beginning, it would facilitate considerable speed-up for the decision procedure.

Our main idea is to consider an over approximation of φ, which is a PA formula φ', and use φ' to remove the infeasible candidate variable orders.

Note that all the exponential terms in φ is of the form $10^{\mathbb{z}}$ for some integer variable $\mathbb{z}$. The over approximation is based on the observation that $10^n \geq 9n + 1$ for every $n \in \mathbb{N}$. Then we obtain the over approximation φ' from φ by replacing each exponential term $10^{\mathbb{z}}$ with a fresh variable $\mathbb{z}'$ and add $\mathbb{z}' \geq 9\mathbb{z} + 1$ as a conjunct.

Then during the enumeration of the linear orders for the variables $\mathbb{x}_1, \ldots, \mathbb{x}_n$, we can quickly remove those infeasible candidates σ such that $\varphi' \wedge \bigwedge\limits_{i \in [n-1]} \mathbb{x}_{\sigma(i)} \leq \mathbb{x}_{\sigma(i+1)}$ is unsatisfiable. A special case is that if φ' is unsatisfiable, then we can directly conclude that original formula φ is unsatisfiable.

6.2 Avoid the elimination of linear occurrences of variables by under approximation

The decision procedure of ExpPA in Section 4 requires the elimination of both exponential and linear occurrences of variables. Considering the fact that PA formulas can be solved efficiently by the state-of-the-art solvers, e.g. CVC4 and Z3, one natural idea is to try to only eliminate the exponential occurrences, but not the linear occurrences, of variables, and obtain the PA formulas in the end, which can then be solved by the state-of-the-art solvers.

Recall that Lemma 1 enables us to eliminate the exponential occurrences of $\mathbb{x}_{\sigma(i)}$ from

$$\begin{array}{l}\tau(\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)}) \hat{=} \\ \quad a_i 10^{\mathbb{x}_{\sigma(i)}} + \sum_{j=1}^{i-1} a_j 10^{\mathbb{x}_{\sigma(j)}} + \sum_{k=1}^{i} b_k \mathbb{x}_{\sigma(k)} \leq c.\end{array}$$

Actually, Lemma 1 does more in the sense that all occurrences of $\mathbb{x}_{\sigma(i)}$, including the linear ones, are eliminated from the atomic formulas resulted from $\tau(\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)})$, e.g. $\tau(\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)})[\mathbb{x}_{\sigma(i-1)} + p/\mathbb{x}_{\sigma(i)}]$. Then we can continue eliminating the exponential occurrences of $\mathbb{x}_{\sigma(i-1)}$ from $\tau(\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)})[\mathbb{x}_{\sigma(i-1)} + p/\mathbb{x}_{\sigma(i)}]$, provided that the coefficient of $\mathbb{x}_{\sigma(i-1)}$ therein is nonzero. Iterating this process would produce a PA formula eventually.

Nevertheless, the side condition of Lemma 1, namely $a_i \neq 0$, undermines the aforementioned natural idea. If $a_i = 0$, but $b_i \neq 0$, then we are unable to utilize Lemma 1 to eliminate the linear occurrences of $\mathbb{x}_{\sigma(i)}$ from $\tau(\mathbb{x}_{\sigma(i)}, \dots, \mathbb{x}_{\sigma(1)})$. In this case, the quantifier elimination algorithm of PA has to be applied to eliminate $\mathbb{x}_{\sigma(i)}$ from $\tau(\mathbb{x}_{\sigma(i)}, \dots, \mathbb{x}_{\sigma(1)})$, so that later on, we can eliminate the exponential occurrences of $\mathbb{x}_{\sigma(i-1)}$, which requires that $\mathbb{x}_{\sigma(i-1)}$ is the maximum variable in the left-hand side of the inequality.

To avoid applying the quantifier elimination algorithm of PA, we consider the following under approximation of $\tau(\mathbb{x}_{\sigma(i)}, \dots, \mathbb{x}_{\sigma(1)})$, namely, we additionally assume that $\mathbb{x}_{\sigma(i)} \leq 10^u$ for some constant bound $u \in \mathbb{N}$ with $u \geq 1$. Then $\tau(\mathbb{x}_{\sigma(i)}, \dots, \mathbb{x}_{\sigma(1)})$ can be rewritten as

$$\tau'(\mathbb{x}_{\sigma(i)}, \dots, \mathbb{x}_{\sigma(1)}) \hat{=} \sum_{j=1}^{i-1} a_j 10^{\mathbb{x}_{\sigma(j)}} + \sum_{k=1}^{i-1} b_k \mathbb{x}_{\sigma(k)} \leq c - b_i \mathbb{x}_{\sigma(i)}.$$

Let us assume $a_{i-1} > 0$. Define c_1, c_2 as follows: If $b_i > 0$, then $c_1 = c - b_i 10^u$ and $c_2 = c$, otherwise, $c_1 = c$ and $c_2 = c - b_i 10^u$. It is easy to observe that $c_1 \leq c - b_i \mathbb{x}_{\sigma(i)} \leq c_2$. Then we can apply Lemma 1 to the following two inequalities to eliminate $10^{\mathbb{x}_{\sigma(i-1)}}$,

$$\sum_{j=1}^{i-1} a_j 10^{\mathbb{x}_{\sigma(j)}} + \sum_{k=1}^{i-1} b_k \mathbb{x}_{\sigma(k)} \leq c_1 \tag{2}$$

and

$$\sum_{j=1}^{i-1} a_j 10^{\mathbb{x}_{\sigma(j)}} + \sum_{k=1}^{i-1} b_k \mathbb{x}_{\sigma(k)} \leq c_2. \tag{3}$$

Let $\alpha_1 = \ell_{10}(c_1) - \ell_{10}(a_{i-1})$ and $\alpha_2 = \ell_{10}(c_2) - \ell_{10}(a_{i-1})$. Then from Lemma 1,

- if $\mathbb{x}_{\sigma(i-1)} \geq B$, $\mathbb{x}_{\sigma(i-1)} \leq \alpha_1 - 1$, and $\mathbb{x}_{\sigma(i-1)} \geq \mathbb{x}_{\sigma(i-2)} + \delta$, then inequality (2), thus also $\tau'(\mathbb{x}_{\sigma(i)}, \dots, \mathbb{x}_{\sigma(1)})$, is evaluated to `true`,
- if $\mathbb{x}_{\sigma(i-1)} \geq B$, $\mathbb{x}_{\sigma(i-1)} \geq \alpha_2 + 2$, and $\mathbb{x}_{\sigma(i-1)} \geq \mathbb{x}_{\sigma(i-2)} + \delta$, then inequality (3), thus also $\tau'(\mathbb{x}_{\sigma(i)}, \dots, \mathbb{x}_{\sigma(1)})$, is evaluated to `false`.

Therefore, $\tau'(\mathbb{x}_{\sigma(i)}, \dots, \mathbb{x}_{\sigma(1)})$, thus also $\tau(\mathbb{x}_{\sigma(i)}, \dots, \mathbb{x}_{\sigma(1)})$, is equivalent to

$$\begin{array}{l}
\bigvee_{p=0}^{B-1} (\mathbb{x}_{\sigma(i-1)} = p \wedge \tau'(\mathbb{x}_{\sigma(i)}, \dots, \mathbb{x}_{\sigma(1)})[p/\mathbb{x}_{\sigma(i-1)}]) \\
\bigvee (\mathbb{x}_{\sigma(i-1)} \geq B \wedge \mathbb{x}_{\sigma(i-1)} \leq \alpha_1 - 1 \wedge \mathbb{x}_{\sigma(i-1)} \geq \mathbb{x}_{\sigma(i-2)} + \delta) \\
\bigvee \bigvee_{p=0}^{\delta-1} \left(\begin{array}{l} \mathbb{x}_{\sigma(i-1)} \geq B \wedge \mathbb{x}_{\sigma(i-1)} \leq \alpha_1 - 1 \wedge \\ \mathbb{x}_{\sigma(i-1)} = \mathbb{x}_{\sigma(i-2)} + p \wedge \\ \tau'(\mathbb{x}_{\sigma(i)}, \dots, \mathbb{x}_{\sigma(1)})[\mathbb{x}_{\sigma(i-2)} + p/\mathbb{x}_{\sigma(i-1)}] \end{array} \right) \\
\bigvee \bigvee_{p=0}^{\delta-1} \left(\begin{array}{l} \mathbb{x}_{\sigma(i-1)} \geq B \wedge \mathbb{x}_{\sigma(i-1)} \geq \alpha_2 + 2 \wedge \\ \mathbb{x}_{\sigma(i-1)} = \mathbb{x}_{\sigma(i-2)} + p \wedge \\ \tau'(\mathbb{x}_{\sigma(i)}, \dots, \mathbb{x}_{\sigma(1)})[\mathbb{x}_{\sigma(i-2)} + p/\mathbb{x}_{\sigma(i-1)}] \end{array} \right) \\
\bigvee \bigvee_{p=\alpha_1}^{\alpha_2+1} \left(\begin{array}{l} \mathbb{x}_{\sigma(i-1)} \geq B \wedge \mathbb{x}_{\sigma(i-1)} = p \wedge \\ \tau'(\mathbb{x}_{\sigma(i)}, \dots, \mathbb{x}_{\sigma(1)})[p/\mathbb{x}_{\sigma(i-1)}] \end{array} \right),
\end{array}$$

where all exponential occurrences of $\mathbb{x}_{\sigma(i-1)}$ are eliminated.

Similarly, we can eliminate the exponential occurrences of $\mathbb{x}_{\sigma(i-2)}$ from $\tau'(\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)})[\mathbb{x}_{\sigma(i-2)}+p/\mathbb{x}_{\sigma(i-1)}]$ as well as $\tau'(\mathbb{x}_{\sigma(i)}, \ldots, \mathbb{x}_{\sigma(1)})[p/\mathbb{x}_{\sigma(i-1)}]$, and so on. Eventually, we obtain a PA formula.

6.3 Additional optimization techniques

Synchronize the elimination of exponential occurrences of the same variable in different atomic formulas

Although Lemma 1 is stated for a single atomic formula, the elimination of the exponential occurrences of the same variable in different atomic formulas can actually be synchronized. That is, let $\alpha_1^\tau, \alpha_2^\tau, B^\tau, \delta^\tau$ be the constants as stated in the aforementioned under-approximation of an inequality τ, define $\alpha_1^{\min}, \alpha_2^{\max}, B^{\max}, \delta^{\max}$ as the minimum of α_1^τ, the maximum of α_2^τ, the maximum of B^τ, and the maximum of δ^τ respectively with τ ranging over the inequalities of φ. Then we can use the same constants $\alpha_1^{\min}, \alpha_2^{\max}, B^{\max}, \delta^{\max}$ for different inequalities when eliminating the exponential occurrences of the same variable.

Avoid the formula-size blow-up by depth-first search

The PA formula resulting from the elimination of exponential occurrences is essentially a big disjunction of the formulas of small size. If we store this big disjunction naively, then the formula size quickly blows up and exhausts the memory. Instead, we choose to do a depth-first search (DFS) and consider the disjuncts, which are of small sizes, one by one, and solve the satisfiability problem for these disjuncts. If during the search, a satisfiable disjunct is found, then the search terminates and "SAT" is reported.

Preprocess with small upper bound

We believe that if a quantifier-free ExpPA formula is satisfiable, then most probably it is satisfiable with small values assigned to variables. Consequently, as a preprocessing step, we put a small upper bound, e.g. the biggest constant occurring in the formula, on the values of variables, and perform a depth-first search, so that a model can be quickly found, if there is any. If this preprocessing is unsuccessful, then we continue the search with the greater upper bound 10^u for some proper $u \geq 1$.

7 Implementation and Experiments

7.1 Implementation

We implemented the decision procedure in Wolfram Mathematica, and obtain a solver, called the ExpPA-solver, which is able to solve the satisfiability of ExpPA formulas.

The ExpPA-solver takes a quantifier-free ExpPA formula as the input. Moreover, it allows specifying a constant upper bound 10^u for the values of variables. If a constant upper bound 10^u is specified, then the problem is to decide whether there is an assignment of values no more than 10^u to variables satisfying the given ExpPA formula. The outputs of the ExpPA-solver are either "SAT", "UNSAT", "B-UNSAT", or “TIMEOUT”, corresponding to the facts that the given formula is satisfiable, unsatisfiable, unsatisfiable up to 10^u, or the search goes beyond the predetermined time limit. If the output is “SAT”, then a model (namely, an assignment of values to variables) is also returned.

7.2 Benchmarks

To evaluate the performance of the ExpPA-solver, we created two benchmark suites, ARITHMETIC and STRINGHASH[4].

The ARITHMETIC benchmark suite

This suite comprises three groups of randomly generated ExpPA formulas. Each group is characterized by four parameters (EV, LV, EI, LI), where EV, LV represent the number of variables with exponential occurrences and with only linear occurrences respectively, and EI, LI represent the number of inequalities with exponential terms and with only linear terms respectively. We consider three parameter classes, $(2, 3, 3, 4)$, $(3, 4, 4, 5)$, and $(4, 5, 5, 6)$. Each group of the benchmark suite consists of 200 randomly generated problem instances. The coefficients of exponential terms are randomly selected from the interval $[-10^2, 10^2]$ and the other coefficients/constants are randomly selected from $[-10^5, 10^5]$. The two intervals are chosen with the intention that the coefficients of exponential terms are smaller so that they do not always dominate the left-hand side of the inequalities. Moreover, aiming at a better coverage of the syntactical ingredients of ExpPA, we randomly choose some problem instances and replace the $\leq$ symbol of their first inequalities by $=$. The constant upper bound for the values of variables is set to be 10^{20}, motivated by the fact that the largest 64-bit integer is less than 10^{20}. We also create an SMTLib2 file for each problem instance, to facilitate the comparison with the state-of-the-art of SMT solvers CVC4 and Z3. Because neither CVC4 nor Z3 supports the exponential functions natively, in the SMTLib2 files, we encode 10^x as a recursive function $f(x)$ defined by: $f(0) = 1$ and $f(n + 1) = 10 * f(n)$.

The STRINGHASH benchmark suite

This suite comprises two groups of string constraints generated from the string hash functions $\mathsf{hash}(w)$ encoded by $\mathsf{parseInt}$. For one of them, we restrict the nonce string conforming some flat pattern, while for the other one, we allow any word from Σ^*_{num} to be used as the nonce.

The string constraints in the STRINGHASH benchmark suite are of the form $x \in \mathcal{A} \wedge (\mathsf{parseInt}(x) \bmod m) \bmod m' = 0 \wedge \mathsf{len}(x) < 100$, where $\mathcal{A}$ is an FA, $m, m' \geq 2$. The two groups of string constraints are characterized by flat and non-flat regular constraints respectively. The flat group comprises 300 problem instances, where the flat languages are of the form $12345 w_1^+ w_2^+$, $12345 w_1^+ w_2^+ 6789$, or $w_1^+ w_2^+ 6789$, with $w_1, w_2 \in \Sigma^+_{\mathsf{num}}$, where 12345 and 6789 are the text to be protected, and $w_1^+ w_2^+$ is the pattern for nonce string. The non-flat group comprises 300 problem instances, where the non-flat languages are of the form $12345\Sigma^*_{\mathsf{num}}$,

[4]The benchmarks are available at https://github.com/EcstasyH/PAexp-Solver/tree/main/Benchmark

$12345\Sigma_{\mathsf{num}}^*6789$, or $\Sigma_{\mathsf{num}}^*6789$. Moreover, the number m is a randomly chosen prime number in the interval $[10^2, 10^5]$ and $1 \leq m' < m$ (m' is not necessarily a prime number). We generate the SMTLib2 files for these string constraints, as inputs to the string constraint solvers. On the other hand, for the ExpPA-solver, we do the following:

- For flat instances, we generate the ExpPA formulas corresponding to the string constraints, as the inputs to the ExpPA-solver.
- For non-flat instances, we use flat languages $a^*(b_1 \ldots b_k)$ to under approximate Σ_{num}^*, where $a, b_1, \ldots, b_k \in \Sigma_{\mathsf{num}}$. We iterate the following procedure until a model is found or the time limit is reached: Initially, set $k = 1$ and iterate by assigning $0, \ldots, 9$ to a. For each assignment, we encode the resulting string constraint into an ExpPA formula with only one exponential variable. If the resulting ExpPA formula is unsatisfiable, then we increase k by 1 and repeat this process.

We would like to remark that the flattening strategy for non-flat regular constraints here is a strict generalization of that in [17]: Patterns of the form $0^*(b_1...b_k)$ were considered therein and PA formulas are sufficient to encode such patterns. On the other hand, we consider patterns of the form e.g. $(a)^*(b_1 \ldots b_k)$ (where $a \in \Sigma_{\mathsf{num}}$ can be nonzero), which requires ExpPA formulas to encode in general.

7.3 Experiments

We compare the ExpPA-solver against the state-of-the-art SMT solvers on the generated benchmarks. Specifically,

- over the ARITHMETIC benchmark suite, we compare the ExpPA-solver against CVC4 (version 1.8) and Z3 (version 4.8.10),
- over the STRINGHASH benchmark suite, we compare the ExpPA-solver against CVC4, Z3, and Trau[5].

All the experiments are run on a lap-top with the Intel i5 1.4GHz CPU and 8GB memory. We set the time limit as 60 seconds per problem instance.

The experiment results are summarized in Table 1, where “Fail” means either timeout, unknown, or wrong answers.

On the ARITHMETIC benchmark suite, the ExpPA-solver solves around 20%-60% more instances than Z3, and 30%-100% more instances than CVC4. Moreover, the gap becomes bigger as the the sizes of the formulas increase, which demonstrates that the ExpPA-solver is more scalable in solving formulas of greater sizes. The average time of the ExpPA-solver is comparable with Z3 and CVC4. The ExpPA-solver reports “B-UNSAT” for 47 instances of the $(2, 3, 3, 4)$-group, while it does not report “B-UNSAT” (except one) for the other two groups. If more time is allowed, the ExpPA-solver is able to report “B-UNSAT” for the “TIMEOUT” instances.

On the STRINGHASH benchmark suite, in overall, the ExpPA-solver solves significantly more problem instances, especially those satisfiable instances, than Z3, CVC4, and Trau. For instance, for flat regular constraints, the ExpPA-solver solves

[5] https://github.com/guluchen/z3/tree/new_trau

almost all 300 problem instances, except 3 of them[6], while Z3, CVC4, Trau solve only 34, 89, 187 instances respectively. Trau gets wrong answers for some problem instances, e.g. it reports “UNSAT” for some satisfiable instances. From the results, we can see that the ExpPA-solver achieves a good tradeoff between precision and efficiency, although it is slower than the other solvers. (More detailed experiment results on STRINGHASH can be found in Table 2.) WH: The two tables may be too big. :HW

8 Conclusion

In this paper, we proposed a complete flattening approach for string constraints with string-integer conversion and flat regular constraints, based on a quantifier elimination procedure by Point in 1986, for the extension of Presburger arithmetic with exponential functions. We gave a more accessible reformulation of Point’s procedure and proposed various optimizations. Moreover, we achieved the first prototypical implementation of Point’s procedure. We also did extensive experiments to evaluate the performance of the implementation. The experiment results show the efficacy of our implementation, compared with the state-of-the-art solvers.

References

[1] Makanin, G.S.: The problem of solvability of equations in a free semigroup. Matematicheskii Sbornik **145**(2), 147–236 (1977)

[2] Plandowski, W.: Satisfiability of word equations with constants is in pspace. In: 40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039), pp. 495–500 (1999). IEEE

[3] Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: International Conference on Computer Aided Verification, pp. 171–177 (2011). Springer

[4] De Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, pp. 337–340 (2008). Springer

[5] Zheng, Y., Zhang, X., Ganesh, V.: Z3-str: A Z3-based string solver for web application analysis. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, pp. 114–124 (2013). ACM

[6] Trinh, M.-T., Chu, D.-H., Jaffar, J.: S3: A symbolic string solver for vulnerability detection in web applications. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 1232–1243 (2014). ACM

[6] These three instances can actually be solved in 70 seconds.

Table 1 Experimental Results, O: Output, S:SAT, U: UNSAT, B: Bounded UNSAT, F: Fail, #: number of problems, T: average time in seconds

Group	O	Z3		CVC4		Trau		ExpPA	
		#	T	#	T	#	T	#	T
(2,3,3,4)	S	56	**0.4**	42	2.3	-	-	**64**	**0.4**
	U	69	**0.1**	72	**0.1**	-	-	**89**	**0.1**
	B	-	-	-	-	-	-	47	9.5
	F	75	-	86	-	-	-	**0**	-
(3,4,4,5)	S	33	**1.4**	25	2.9	-	-	**52**	3.3
	U	59	**0.1**	60	**0.1**	-	-	**88**	**0.1**
	B	-	-	-	-	-	-	1	54.0
	F	108	-	115	-	-	-	**59**	-
(4,5,5,6)	S	35	**1.8**	19	6.6	-	-	**47**	22.4
	U	36	0.3	39	0.4	-	-	**72**	**0.1**
	B	-	-	-	-	-	-	0	-
	F	129	-	142	-	-	-	**81**	-
Flat	S	34	19.0	88	12.7	5	**0.1**	**115**	12.3
	U	0	-	1	4.0	182	**2.5**	**182**	47.7
	F	266	-	211	-	113	-	**3**	-
Non-flat	S	210	7.8	144	4.9	55	5.9	**300**	16.7
	U	0	-	0	-	0	-	0	-
	F	90	-	156	-	245	-	**0**	-

[7] Abdulla, P.A., Atig, M.F., Chen, Y.-F., Holík, L., Rezine, A., Rümmer, P., Stenman, J.: Norn: An SMT solver for string constraints. In: International Conference on Computer Aided Verification, pp. 462–469 (2015). Springer

[8] Chen, T., Chen, Y., Hague, M., Lin, A.W., Wu, Z.: What is decidable about string constraints with the replaceall function. Proceedings of the ACM on Programming Languages **2**(POPL), 3 (2017)

Table 2 Experimental results on STRINGHASH, O: Output, S:SAT, U: UNSAT, B: Bounded UNSAT, F: Fail, #: number of problems, T: average time in seconds

Group	O	Z3		CVC4		Trau		ExpPA	
		#	T	#	T	#	T	#	T
$12345(w_1)^+(w_2)^+$	S	5	14.0	29	8.5	3	**0.1**	**37**	9.9
	U	0	-	0	-	**60**	**1.3**	**60**	47.2
	F	95	-	71	-	37	-	**3**	-
$12345(w_1)^+ (w_2)^+6789$	S	11	13.0	29	12.0	0	-	**37**	**10.6**
	U	0	-	0	-	**63**	**1.2**	**63**	50.0
	F	89	-	71	-	37	-	**0**	-
$(w_1)^+(w_2)^+6789$	S	18	24.0	30	9.3	2	**0.1**	**41**	16.1
	U	0	-	1	4.0	**59**	**2.5**	**59**	45.8
	F	82		69		39		**0**	
$12345\Sigma_{\mathsf{num}}^*$	S	82	8.7	**100**	**2.2**	28	5.9	**100**	18.5
	U	0	-	0	-	0	-	0	-
	F	18	-	**0**	-	72	-	**0**	-
$12345\Sigma_{\mathsf{num}}^*6789$	S	60	9.3	17	7.8	3	**0.3**	**100**	16.0
	U	0	-	0	-	0	-	0	-
	F	40	-	83	-	97	-	**0**	-
$\Sigma_{\mathsf{num}}^*6789$	S	68	5.5	27	13.0	24	**9.0**	**100**	15.7
	U	0	-	0	-	0	-	0	-
	F	32	-	73	-	76	-	**0**	-

[9] Holík, L., Janků, P., Lin, A.W., Rümmer, P., Vojnar, T.: String constraints with concatenation and transducers solved efficiently. Proceedings of the ACM on Programming Languages **2**(POPL), 4 (2017)

[10] Aydin, A., Eiers, W., Bang, L., Brennan, T., Gavrilov, M., Bultan, T., Yu, F.: Parameterized model counting for string and numeric constraints. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 400–410 (2018). ACM

[11] Yu, F., Alkhalaf, M., Bultan, T.: Stranger: An automata-based string analysis tool for php. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, pp. 154–157 (2010). Springer

[12] Abdulla, P.A., Atig, M.F., Chen, Y.-F., Diep, B.P., Holík, L., Rezine, A., Rümmer, P.: Trau: SMT solver for string constraints. In: 2018 Formal Methods in Computer Aided Design (FMCAD), pp. 1–5 (2018). IEEE

[13] Day, J.D., Ganesh, V., He, P., Manea, F., Nowotka, D.: The satisfiability of word equations: Decidable and undecidable theories. In: Potapov, I., Reynier, P. (eds.) Reachability Problems - 12th International Conference, RP 2018, Marseille, France, September 24-26, 2018, Proceedings. Lecture Notes in Computer Science, vol. 11123, pp. 15–29. Springer, ??? (2018). https://doi.org/10.1007/978-3-030-00250-3_2. https://doi.org/10.1007/978-3-030-00250-3_2

[14] ECMAScript, E., Association, E.C.M., et al.: Ecmascript language specification (2019). https://www.ecma-international.org/ecma-262/

[15] Kiezun, A., Ganesh, V., Artzi, S., Guo, P.J., Hooimeijer, P., Ernst, M.D.: HAMPI: A solver for word equations over strings, regular expressions, and context-free grammars. ACM Trans. Softw. Eng. Methodol. **21**(4), 25–12528 (2012). https://doi.org/10.1145/2377656.2377662

[16] Saxena, P., Akhawe, D., Hanna, S., Mao, F., McCamant, S., Song, D.: A symbolic execution framework for JavaScript. In: 31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berleley/Oakland, California, USA, pp. 513–528. IEEE Computer Society, ??? (2010). https://doi.org/10.1109/SP.2010.38. https://doi.org/10.1109/SP.2010.38

[17] Abdulla, P.A., Atig, M.F., Chen, Y., Diep, B.P., Dolby, J., Janku, P., Lin, H., Holík, L., Wu, W.: Efficient handling of string-number conversion. In: Donaldson, A.F., Torlak, E. (eds.) Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020, pp. 943–957. ACM, ??? (2020). https://doi.org/10.1145/3385412.3386034. https://doi.org/10.1145/3385412.3386034

[18] Abdulla, P.A., Atig, M.F., Chen, Y., Diep, B.P., Holík, L., Rezine, A., Rümmer, P.: Trau: SMT solver for string constraints. In: Bjørner, N., Gurfinkel, A. (eds.) 2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 - November 2, 2018, pp. 1–5. IEEE, ??? (2018). https://doi.org/10.23919/FMCAD.2018.8602997. https://doi.org/10.23919/FMCAD.2018.8602997

[19] Abdulla, P.A., Atig, M.F., Chen, Y., Diep, B.P., Holík, L., Rezine, A., Rümmer, P.: Flatten and conquer: a framework for efficient analysis of string constraints. In: Cohen, A., Vechev, M.T. (eds.) Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2017, Barcelona, Spain, June 18-23, 2017, pp. 602–617. ACM, ??? (2017). https://doi.org/10.1145/3062341.3062384. https://doi.org/10.1145/3062341.3062384

[20] Li, G., Ghosh, I.: PASS: string solving with parameterized array and interval automaton. In: Bertacco, V., Legay, A. (eds.) Hardware and Software: Verification and Testing - 9th International Haifa Verification Conference, HVC 2013, Haifa, Israel, November 5-7, 2013, Proceedings. Lecture Notes in Computer Science, vol. 8244, pp. 15–31. Springer, ??? (2013)

[21] Semënov, A.L.: Logical theories of one-place functions on the set of natural numbers. Mathematics of the USSR-Izvestiya **22**(3), 587–618 (1984). https://doi.org/10.1070/im1984v022n03abeh001456

[22] Point, F.: On the expansion $(\mathbb{N}, +, 2^x)$ of Presburger arithmetic. In: Proceedings of the Fourth Easter Conference on Model Theory, Gross Koris, pp. 17–34 (1986)

[23] Cooper, D.C.: Theorem proving in arithmetic without multiplication. Machine intelligence **7**(91-99), 300 (1972)

[24] G. H. Hardy, E.M.W.: An Introduction to the Theory of Numbers (5th Edition). Oxford University Press, ??? (1980)

[25] Oppen, D.C.: Elementary bounds for Presburger arithmetic. In: Proceedings of the Fifth Annual ACM Symposium on Theory of Computing. STOC ’73, pp. 34–37. Association for Computing Machinery, New York, NY, USA (1973)