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

Hao Wu*‡, Yu-Fang Chen†, Zhilin Wu*‡, Naijun Zhan*‡
* State Key Laboratory of Computer Science,
Institute of Software, Chinese Academy of Sciences, 100190, Beijing, China
Email: {wuhao, wuzl, znj}@ios.ac.cn
† Institute of Information Science, Academia Sinica, Taiwan
Email: yfc@iis.sinica.edu.tw
‡University of Chinese Academy of Sciences, Beijing, China

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. In the PLDI 2020 paper of Abdulla et al., the authors extended the flattening approach to support the string-integer conversion, which is an important function appearing in almost all scripting languages. 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 decision procedure for the linear-exponential arithmetic constraints (namely, the extension of Presburger arithmetic with exponential functions) proposed by Point in 1986. While the decision procedure by Point relies on quantifier eliminations and is expensive, we introduce various optimizations and provide an efficient implementation. We evaluate the performance of the 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 for string-integer conversion constraints as well as linear-exponential constraints, in both precision and efficiency.

I. Introduction

solve string constraint is hard

strategy: do unsat and sat separately

strategy: using different procedure to (dis)prove validaity

for disprove validatity, there are two approches, first is bound string length

cannot handle $x.y \neq z \wedge |x| > 2000$

more recent approach is flattening

it is known that word equaltion + flat regular constraints + len constraints is decidable

it was unknown that whether word equations + flat regular constraints + len constraints + parseInt is decidable

II. Preliminaries

In this section, we introduce some basic concepts and theories that will be used later.

A. Basic Concepts

a) Sets and Strings: We use $\mathbb{N}$ and $\mathbb{Z}$ to denote the set of natural numbers and integers, respectively. $\mathbb{N}^+$ stands for the set of non-zero natural numbers. Let Σ be a finite alphabet, a string w over Σ is a sequence $a_1....a_n$ of characters from Σ. Empty string is denoted by ϵ. Σ^* denotes the set of all finite strings over Σ, and Σ_ϵ stands for $\Sigma \cup \{\epsilon\}$. For any string $w_1, w_2 \in \Sigma^*$, we use $|w_1|$ to denote the length of w_1, and $w_1 \cdot w_2$ to denote the concatenation of w_1 and w_2. A language L over Σ is a subset of Σ^*.

There are two types of variables in string constraints, i.e., X, a set of string variables ranged over Σ^*, and Z, a set of integer variables ranged over $\mathbb{Z}$. As usual, an interpretation I is a mapping from the set of variables $X \cup Z$ to the respective domain, essentially a pair of two mappings I_X and I_Z, i.e., $I = (I_X, I_Z)$, where I_X is a mapping in $X \mapsto \Sigma^*$ and I_Z is a mapping in $Z \mapsto \mathbb{N}$.

b) Finite State Automata: A Finite State Automaton is a tuple $\mathcal{A} = \langle Q, \Sigma, \Delta, q_{\text{init}}, q_{\text{acc}} \rangle$, where Q is a finite set of states, Σ is the given alphabet, $\Delta \subseteq Q \times \Sigma_\epsilon \times Q$ defines the transition relations in $\mathcal{A}$. $q_{\text{init}}, q_{\text{acc}} \in Q$ is the initial state and accepting state. A sequence $q_0\langle a_1\rangle q_1 ... \langle a_n\rangle q_n$ is called accepting if $q_0 = q_{\text{init}}$, $q_n = q_{\text{acc}}$ and $q_{i-1}\langle a_i\rangle q_i \in \Delta$ for $1 \leq i \leq n$.

c) Presburger Arithmetic: The Presburger Arithmetic (PA) is a first order theory over signature $\Sigma_{\mathbb{N}} \hat{=} \{0, 1, +, =\}$, where $0, 1$ are constants, $+$ is a binary function and $=$ is a binary predicate.

PA can be axiomatized by the following axioms [?]

- $\forall x, \neg(x + 1 = 0)$
- $\forall x \forall y. x + 1 = y + 1 \to x = y$
- $F(0) \wedge (\forall x. F(x) \to F(x + 1)) \to \forall x. F(x)$
- $\forall x. x + 0 = x$
- $\forall x \forall y. x + (y + 1) = (x + y) + 1$

Given the domain $\mathbb{N}$, the standard interpretation of PA interprets $0, 1$ to $0_{\mathbb{N}}, 1_{\mathbb{N}} \in \mathbb{N}$ and $+, =$ to addition and equality over $\mathbb{N}$. We call a PA formula without quantifiers a quantifier-free PA formula.

PA is a decidable theory, and the complexity of decidability is related to the number and locations of quantifiers. Generally, the upper bound (on deterministic time and space) for deciding a formula of length n is $2^{2^{2^{pnlog(n)}}}$, where $p > 1$ is a constant [?].

d) Parikh Image: Given an alphabet Σ and a string $w \in \Sigma^*$, we define the set of Parikh variables $\Sigma^{\#} \hat{=} \{a^{\#} \mid a \in \Sigma\}$. The Parikh image of w is a function $\mathbb{P}(w) : \Sigma^{\#} \mapsto \mathbb{N}$, which maps each symbol $a^{\#} \in \Sigma^{\#}$ to the number of occurrences of a in w. For example, let $w \hat{=} aabba$, then $\mathbb{P}(w)(a^{\#}) = 3, \mathbb{P}(w)(b^{\#}) = 2$.

For a language $L \subseteq \Sigma^*$, define the Parikh image of L to be $\mathbb{P}(L) \hat{=} \{\mathbb{P}(w) | w \in L\}$. We say a language L is Parikh-definable if $\mathbb{P}(L)$ can be characterized by a quantifier-free PA formula over $\Sigma^{\#}$, where $a^{\#}$ in the formula encodes the number of occurrences of a. It is well known that any context-free language (therefore regular language) is Parikh definable [?].

e) String Terms: Given a finite alphabet Σ and a finite set X of string variables over Σ^*, we define the set of terms $\text{Terms}(\Sigma, X)$ to be the smallest set satisfying

1 $\Sigma \cup \{\epsilon\} \cup X \subseteq \text{Terms}(\Sigma, X)$;
2 if $t_1, t_2 \in \text{Terms}(\Sigma, X)$, then $t_1 \cdot t_2 \in \text{Terms}(\Sigma, X)$.

We extend I_X to $\text{Terms}(\Sigma, X)$ by letting $I_X(\epsilon) = \epsilon$, for $a \in \Sigma, I_X(a) = a$, and $I_X(t_1 \cdot t_2) = I_X(t_1) \cdot I_X(t_2)$.

f) String Constraints: Given a constraint ϕ and an interpretation I, $I \models \phi$ denotes that I satisfies ϕ, and I is called a model of ϕ. We use $\|\phi\|$ to denote the set of all models of ϕ.

We define the following three forms of atomic string constraints:

- An equality constraint ϕ_e is of the form $t_1 = t_2$, where $t_1, t_2 \in \text{Terms}(\Sigma, X)$. We define $\|\phi_e\| = \{I \mid I(t_1) = I(t_2)\}$. Inequality constraints can be defined analogously.
- A regular constraint ϕ_r is of the form $x \in L(\mathcal{A})$, where $x \in \mathbb{X}$ and $\mathcal{A}$ is a finite state automaton. We define $\|\phi_r\| = \{I \mid I(x) \in L(\mathcal{A})\}$.
- A length constraint ϕ_l is a linear constraint over $Z \cup \{|x| \mid x \in X\}$, where $|\cdot|$ is the length function. We define $\|\phi_l\| = \{I \mid I \models \phi_l\}$.

In Section 3, we will introduce a new form of atomic string constraints, i.e., string-number conversion constraints. A string constraint is a Boolean combination of atomic string constraints possibly with quantifications over $X \cup Z$. Other notions are same as in the first-order logic.

Giving a string constraint Ψ, the problem of string constraints solving is to decide whether $\|\Psi\|$ is empty, if not, to compute an interpretation I that satisfies Ψ.

B. Flat Automata and Flat Languages

Given a string constraint Ψ, the general problem of deciding whether $\|\Psi\|$ is empty is undecidable. However, the problem becomes decidable when certain restrictions are imposed. One of the restriction is by flat automata and flat languages, defined below.

a) Flat Languages and Automata: For a fixed alphabet Σ, we say a language L over Σ to be $\langle p, q\rangle$-flat if there exist strings $w_1, ..., w_q \in \Sigma^*$ such that $|w_i| \leq p$ for all $i : 1 \leq i \leq q$ and $L = (w_1)^*(w_2)^*...(w_q)^*$. We use α to denote $\langle p, q\rangle$, and call it the abstraction parameter of L. Intuitively, a flat language with abstraction parameter $\alpha = \langle p, q\rangle$ consists of q loops and the length of each loop body is equal or less than p. For example, $L = (ab)^*(a)^*(bb)^*$ is a $\langle 2, 3\rangle$-flat language.

Flat automata are a special form of finite state automata that recognize flat languages. Fix the abstraction parameter $\alpha = \langle p, q\rangle$, a flat automaton consists of q loops, each loop is a circle of p states. Formally, an α-flat automaton contains pq states at most, and we name the states from 1 to pq, 1 is the initial state and $(pq-p+1)$ is the accepting state. We use $\cdot$ as a placeholder for some symbol in Σ_ϵ, the transition relations of state i are defined as

- if i mod $p = 1$ and $i \neq pq-p+1$, then $(i, \epsilon, i+p) \in \Delta$, $(i, \cdot, i+1) \in \Delta$;
- if i mod $p = 0$, then $(i, \cdot, i-p+1) \in \Delta$;
- otherwise, $(i, \cdot, i+1) \in \Delta$.

A $\langle 2, 3\rangle$-flat automaton that recognizes $L \hat{=} (ab)^*(a)^*(bb)^*$ is shown in figure (1).

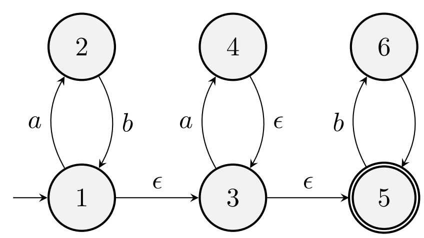

Fig. 1. A $\langle 2, 3\rangle$-flat automaton that recognizes $L \hat{=} (ab)^*(a)^*(bb)^*$

b) Generic Flat Languages and Automata: Fix $\alpha = \langle p, q\rangle$, we define the generic α-flat language is the union of all α-flat languages, denoted by $\mathbb{F}(\alpha)$. Now, we try to define an automaton that recognizes all α-flat languages, i.e., collects all behaviors of α-flat automata.

Intuitively, the generic automaton is obtained by introducing a new alphabet (a multi-set with pq copies of the original alphabet) and adding more transitions (labels), the states and the overall framework remain unchanged. In details, a generic α-flat automaton is still a finite state automaton over $\Sigma(\alpha) \hat{=} \{a(i) | (a \in \Sigma_\epsilon) \wedge i \in \mathbb{N} : 1 \leq i \leq pq\} \cup \{\epsilon\}$. The states are still named from 1 to pq, the initial state is 1 and the accepting state is $(pq - p + 1)$. The transition relations for state i are defined as

- if i mod $p = 1$ and $i \neq pq-p+1$, then $(i, \epsilon, i+p) \in \Delta$ and $\forall s \in \Sigma_\epsilon.(i, s(i), i+1) \in \Delta$;
- if i mod $p = 0$, $\forall s \in \Sigma_\epsilon.(i, s(i), i-p+1) \in \Delta$;
- otherwise, $\forall s \in \Sigma_\epsilon.(i, s(i), i+1) \in \Delta$.

For $\Sigma = \{a, b\}$, an example of generic $\langle 2, 3\rangle$-flat automaton is shown in figure (2).

However, the resulted automaton may accept languages that are not in $\mathbb{F}(\alpha)$, because in different passes inside a loop, the automaton can choose different symbols between identical pairs. To avoid this problem, we add a so-called

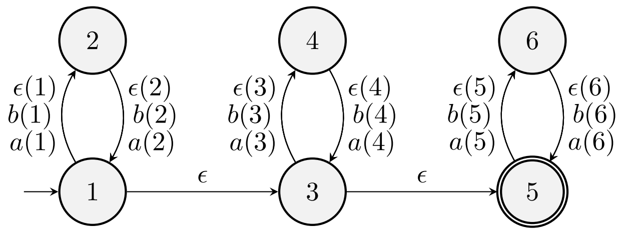

Fig. 2. The generic $\langle 2, 3\rangle$-flat automaton

purity condition on the accepting language of generic flat automata, which is equivalent to intersecting the language of a generic flat automaton with a language that encodes the purity condition.

We say a string $w \in (\Sigma(\alpha))^*$ is pure if for all $i : 1 \le i \le pq$, and $a, b \in \Sigma$, $a \neq b \wedge \#w(a(i)) > 0$ implies $\#w(b(i)) = 0$. Formally, the purity condition is defined by

$$\bigwedge_{1\le i\le pq} \bigwedge_{a,b\in\Sigma, a\neq b} (a(i)^{\#} > 0) \rightarrow (b(i)^{\#} = 0)\,. \quad (1)$$

We denote the accepting language of the generic α-flat automaton by $\mathbb{G}(\alpha)$. Note that $\mathbb{G}(\alpha)$ is a language over Σ_α, but what we want is a language over Σ. So we define a renaming function $R : \Sigma(\alpha) \mapsto \Sigma$ such that for all $a(i) \in \Sigma_\alpha, R(a(i)) = a$, and $R(\epsilon) = \epsilon$. Define $\mathbb{G}'(\alpha) \hat{=} \{R(w) \mid w \in \mathbb{G}(\alpha)\}$, for simplicity, we write $\mathbb{G}'(\alpha) = R(\mathbb{G}(\alpha))$.

The important feature of generic flat autamata is that every word $w \in \mathbb{G}(\alpha)$ is uniquely determined by its Parikh image $\mathbb{P}(w)$.

C. Flattening

The flattening technique was first introduced in [?]. Fix an alphabet Σ and an abstraction parameter α, for a given atomic string constraint ϕ, flattening ϕ with parameter α results in a new string constraint ϕ_α, such that $R(\|\phi_\alpha\|) = \|\phi\| \cap \{I \mid \forall x \in X, I(x) \in \mathbb{G}'(\alpha)\}$, where R is the renaming function with its domain extended to interpretations in the normal manner. Intuitively, it restricts ϕ to interpret string variables over $\mathbb{G}'(\alpha)$.

[?] discussed the flattening of basic string constraints including equality, integer, (regular) grammar and transducer constraints. For an atomic string constraint ϕ, the flattening ϕ_α is still an atomic string constraint and is Parikh definable, so its Parikh image can be expressed by a quantifier free PA formula. Together with the purity condition, we obtain an existential quantified PA formula ρ. ρ will be sent to a SMT solver, if the solver returns an solution θ, then we can construct an interpretation for ϕ_α from θ, otherwise it means ϕ is unsatistiable when string variables are interpreted to α-flat languages.

Take a regular constraint $\phi = x \in L(A)$ for example, the flattening of ϕ resutls in a new finite state automaton A' over $\Sigma(\alpha)$, which encodes running A "in parallel" with the generic α-flat automaton. Let ρ_1 be the formula describing the Parikh image of A', which is a formula over variable sets $\Sigma(\alpha)^{\#}$. Let ρ_2 be the purity condition (1). Then we obtain the PA formula $\exists(\Sigma(\alpha))^{\#}.\rho_1 \wedge \rho_2$. In order to distinguish between different string variables, we may replace $a(i)^{\#} \in \Sigma(\alpha)^{\#}$ by $(x, a(i))^{\#}$.

Since the structure of a flat automaton is decided by its abstraction parameter α, a Counter-Example Guided Abstraction Refinement (CEGAR) framework is designed, which contains both an under- and an over-approximation module, to search the possible values of α. The termination for the overall algorithm is not guaranteed.

III. Solving String Constraints with parseInt Function

The string-number conversion functions are commonly used functions in most of programming languages, for example, parseInt() in Java and Int() in Python. The functions usually take two parameters, a string over the agreed alphabet Σ and an optional parameter denotes the radix. They parse the string according to the rules indicated by the radix, and return an integer denoted by the string.

From the view of string constraints, string-number conversion functions give rise to a new form of string constraints and are more expressive than length constraints. So we consider extending string constraints with parseInt function. As the general problem of string constraints is undecidable, we still adopt the idea of flattening, i.e., variables are restricted to (generic) flat languages. This problem has been investigated in [?], which defined a special form of flat restriction (straight-line PFA) and proposed a heuristic search method.

In this section, we describe the problem of interest first, and then present an reduction from the problem of solving flat string constraints with parseInt function to the decidability problem of Presburger Arithmetic with exponential functions. Hence, we identify a decidable subset of string constraints, which is the largest one with decidability so far to the best of our knowledge.

A. String-Number Conversion Function

Commonly, parseInt function takes a string representation of a decimal number and returns an integer. Since we focus on the decidability, we define a binary version of parseInt, which takes a binary string and returns a decimal integer number. For example, $\text{parseInt}('111') = 7$. Clearly, our decision procedure given in this paper can be adapted to string constraints with other string to number conversion function without substantial change. Single quotation marks are used to distinguish a symbol like $'1' \in \Sigma$ from a number $1_{\mathbb{N}} \in \mathbb{N}$ when needed.

Definition III.1. Let $\Sigma_{\text{num}} = \{'0', '1'\}$, $\text{parseInt} : \Sigma_{\text{num}}^* \mapsto \mathbb{N}$ is recursively defined by: for $w \in \Sigma_{\text{num}}^*$

- if $|w| = 0$, i.e., $w = \epsilon$, $\text{parseInt}(w) = 0$;
- if $|w| = 1, \text{parseInt}('w') = w_{\mathbb{N}}$;
- for $|w| \ge 2$ and $w = w_1 \cdot w_2$, where $|w_1|, |w_2| \ge 1$, $\text{parseInt}(w) = \text{parseInt}(w_1)2^{|w_2|} + \text{parseInt}(w_2)$.

Now, we introduce a new form of atomic string constraints: a parseInt constraint ϕ is of the form $n \sim \text{parseInt}(t)$, where n is an integer term, $\sim \in \{\leq, <, =, >, \geq\}$ and $t \in \text{Terms}(\Sigma_{\text{num}}, X)$ is a string term. $\|\phi\| \hat{=} \{I \mid I(n) \sim \text{parseInt}(I(t))\}$.

In what follows, we only consider the problem in the case when $t = x$ and x is restricted to (generic) flat languages. For the general case $t \in \text{Terms}(\Sigma_{\text{num}}, X)$, it can be reduced to this special case by induction on the structure of t.

Given an α-flat language L, we assume $\alpha = \langle p, q\rangle$ and $L = (w_1)^*...(w_q)^*$, where p, q and $w_i (1 \leq i \leq q)$ are known. We further assume that $x = (w_1)^{\beta_1}...(w_q)^{\beta_q}$, then we have

$$\begin{aligned}
&\text{parseInt}(x)\\
=&\text{parseInt}((w_1)^{\beta_1}...(w_q)^{\beta_q})\\
=&\text{parseInt}((w_1)^{\beta_1}...(w_{q-1})^{\beta_{q-1}}) \cdot 2^{\beta_q|w_q|} + \text{parseInt}((w_q)^{\beta_q})
\end{aligned} \tag{2}$$

(2) is a recursive expression. So we only need to deal with the basic case $\text{parseInt}((w_q)^{\beta_q})$, where $w_q \neq \epsilon$

$$\begin{aligned}
\text{parseInt}((w_q)^{\beta_q}) &= \sum_{i=0}^{\beta_q - 1} \text{parseInt}(w_q) \cdot 2^{|w_q| \cdot i}\\
&= \text{parseInt}(w_q) \frac{2^{|w_q| \cdot \beta_q} - 1}{2^{|w_q|} - 1}
\end{aligned} \tag{3}$$

In (3), since w_q and $|w_q|$ are known, they can be regarded as constants. The only unknown variable is β_q.

Combine (2) and (3), the constraint $n = \text{parseInt}(x)$ can be expressed by an arithmetic expression with n and $(\beta_1, ..., \beta_q)$, inevitably with exponential components.

Take $n = \text{parseInt}((11)^a(10)^b)$ for example.

$$\begin{aligned}
n =& \text{parseInt}((11)^a) \cdot 2^{2b} + \text{parseInt}((10)^b)\\
=& \text{parseInt}(11) \cdot \frac{2^{2a} - 1}{2^2 - 1} \cdot 2^{2b} + \text{parseInt}(10) \cdot \frac{2^{2b} - 1}{2^2 - 1}
\end{aligned}$$

So we have

$$3 \cdot n = 3 \cdot 2^{2a+2b} - 3 \cdot 2^{2b} + 2 \cdot 2^{2b} - 2.$$

Observe the form of the above equation, a, b, n are integer variables and either occur in an exponential term or a linear term. This is always the case, so the problem can be reduced to the decidability of PA with exponential function.

When x of parseInt constraints is restricted to the generic α-flat language (α is fixed), the (2) and (3) still hold but $w_i (1 \leq i \leq q)$ is known. However, by definition, the generic α-flat language is the finite union of all α-flat languages, so we can enumerate all possible values for w_i. In this way, the problem can still be reduced to the decidability of PA with exponential function (PA_{exp}), i.e.,

Theorem III.2. If PA_{exp} is decidable, then the satisfiability (validity) of string constraints with parseInt in which all string variables are ranged over flat strings is decidable.

B. Decidability of PA_{exp}

For a first order theory T, we say theory T admits quantifier elimination (QE) if for any formula in T, there is a quantifier-free formula equivalent to it. It is well-known that if a theory admits QE, then it is a decidable theory.

The formal definition of PA is given in section 2.1. Here we add the ordering predicate $\leq$ into the signature, which can be defined by $x \leq y \hat{=} \exists z. x + z = y$. However, the theory PA so far does not admit QE, for example, consider the formula $\exists x. x = y + y$. We augment the theory with countable unary divisible predicates $n|x$, where $n \in \mathbb{N}$, $n|x$ is true if and only if $x \bmod n = 0$ holds. This structure of PA that admits QE is denoted by $(\mathbb{N}, +)$.

We then introduce a theory PA_{exp}, denoted by $(\mathbb{N}, +, 2^x)$, that we work on.

Definition III.3. Let $\mathcal{L} = \{0, 1, +, \leq, n\,|x (n \in \mathbb{N}), 2^x, l_2(x)\}$, $(\mathbb{N}, +, 2^x)$ be a $\mathcal{L}$-theory that has domain $\mathbb{N}$, where

- 2^x is interpreted to the exponential function of 2 over $\mathbb{N}$;
- interpretations of $0, 1, +, \leq, =$ are consistent with PA;
- for $n \geq 1$,$n|x$ holds iff $\exists y. x = ny$;
- $2^0 = 1$, for $n \geq 1, 2^n = 2^{n-1} + 2^{n-1}$; We further assume that if $m, n \in \mathbb{N}, m \leq n$, then $2^{m-n} = 1$
- $l_2(0) = 0$; for $n \geq 1, l_2(n) = y$ iff $2^y \leq n < 2^{y+1}$; $l_2(m - n) = 0$ if $m, n \in \mathbb{N}$ and $m \leq n$.

$\lambda_2(x) = 2^{l_2(x)}$ can be defined by $l_2(x)$, intuitively, $\lambda_2(x)$ means the maximal power of 2 that is not larger than x. Then we have $\lambda_2(x) \leq x \leq 2\lambda_2(x) - 1$, which will be useful in our proof.

Definition III.4. For a strictly increasing function $f : \mathbb{N} \mapsto \mathbb{N}$, f is said to be compatible with addition if for every $m \in \mathbb{N}^+$, f modulo m is periodic, and for any term $A(x) = \sum_{1 \leq i \leq n} a_i f(x + b_i)$, where $n \in \mathbb{N}^+, a_i, b_i \in \mathbb{Z}$, one of the following holds:

- $A(x)$ is bounded;
- there exists a constant $\Delta_A \in \mathbb{N}^+$ such that $\forall x. A(x + \Delta_A) \geq f(x)$;
- there exists a constant $\Delta_A \in \mathbb{N}^+$ such that $\forall x. -A(x + \Delta_A) \geq f(x)$.

Semenov proved that for any function f that is compatible with addition, theory $(\mathbb{N}, +, f)$ admits QE and thus the theory is complete and decidable [?]. Exponential functions are compatible with addition functions, and therefore PA_{exp} is decidable by [?]. Particularly, in [?] Point gave a detailed QE algorithm for $(\mathbb{N}, +, 2^x)$, where $f(x) = 2^x$. Unfortunately, Point's algorithm is flawed, one problem lies in the discussion to eliminate exponential terms. For example, after eliminating variable x in formula $\exists x. y \leq 2^x \wedge x \leq 10$, the expected result should be $y \leq 2^{10} = 1024$, but Francoise's algorithm will return a formula equivalent to $y \leq 2^6 = 64$.

In the next section, within Point's framework, we give a revised algorithm to the decision problem of PA_{exp} with both subtle improvements and crucial corrections, that provides a crucial step to the decision procedure of string constraints with parseInt function according to Theorem 1.

IV. Quantifier Elimination Algorithm for $(\mathbb{N}, +, 2^x)$

Based on Point's work [?], we present a revised QE algorithm for the formula of the form $Qx.\theta(x, \bar{y})$, where Q is a quantifier and $\theta(x, \bar{y})$ is a quantifier-free formula. Since $\forall x.F = \neg\exists x.\neg F$, we further assume the quantifier Q to be the existential quantifier, that is, $\exists x.\theta(x, \bar{y})$. For eliminating quantifiers in arbitrary formula, we apply the algorithm to the innermost quantified formula and repeat this procedure until all quantifiers are eliminated.

We divide the whole QE algorithm into 4 steps. The first step Normalization can be viewed as a pre-processing step, which substitutes "complex" terms like 2^{2^x}, $l_2(3x+y)$ with "simple" terms by introducing new variables x_i. Now we move forward to handle a "simpler" formula, however, at the cost of more quantified variables.

The other three steps undertake the task to eliminate the introduced variables x_i and the original variable x (will be denoted by x_0) one by one. The main body of QE-with-Order step is a loop to enumerate all possible orders among quantified variables. In each iteration, according the given (decreasing) order (corresponding to a for loop), we invoke QE-exp or QE-linear to eliminate these variables one by one. During each iteration of the inner loop (the for loop), if the maximal x_i in $\bar{x}$ occurs in an exponential term in an atomic formula, we will invoke QE-exp to produce a formula equivalent to the atom where x_i occurs linearly; otherwise, if x_i occurs linearly in the formula, QE-linear will be invoked to eliminate all occurrences of x_i, and this procedure is similar to the classic QE algorithm for PA.

A more detailed description is given below.

A. Normalization

In order to show $(\mathbb{N}, +, 2^x)$ admits QE , it is sufficient to show that any 1-existential formula $\exists x.\theta(x, \bar{y})$ indeed does, where $\theta(x, \bar{y})$ is a quantifier-free formula. However, the form of $\exists x.\theta(x, \bar{y})$ is unknown so it may contain terms difficult to handle such as 2^{3x+y+1} or $l_2(x)$. The Normalization step simplifies these terms by introducing new variables, for example, $\exists x.2^{3x+y+1} > 10$ is equivalent to

$$\exists x \exists x_1.2^{x_1} > 10 \wedge x_1 = 3x + y + 1\,.$$

Logarithm functions are replaced by exponential functions, take $\exists x.l_2(x) > 3$ for example, $l_2(x)$ is replaced by a new variable x_1, and we have

$$\exists x \exists x_1.x_1 > 3 \wedge 2^{x_1} \leq x \wedge x \leq 2^{x_1+1} - 1\,.$$

The Normalization step goes like this, first transform the given formula $\theta(x, \bar{y})$ into the Negation Normal Form (NNF). Then $\theta(x, \bar{y})$ becomes a Boolean combination (with only $\wedge$ and $\vee$) of literals.

Make substitutions by introducing new variables according to the while statements in the pseudo-code.For consistency, we rename the original x to be x_0 and assume n new quantified variables are introduced.

Algorithm 1: Normalization

```
Input: 1-existential formula ∃x.θ(x, ȳ)
Output: n-existential formula ∃x̄.θ′(x̄, ȳ)
θ′ := θ(x, ȳ) ;
Transform θ′ into NNF;
// i is used for counting the introduced variables
i = 1;
// ȳ are regarded as constants
while there is a term l_2(t), t is not a constant do
  | θ′ := θ′[x_i/l_2(t)] ∧ (2^{x_i} ≤ t) ∧ (t ≤ 2^{x_i+1} − 1);
  | i := i + 1;
end
while there is a term 2^t, t is not a variable
  x_j (j < i) or a constant do
  | θ′ := θ′[2^{x_i}/2^t] ∧ (x_i ≤ t) ∧ (t ≤ x_i);
  | i := i + 1;
end
n := i − 1;
// Collect quantified variables x_i
Transform all atoms into forms s(x̄) ≤ t(ȳ),
  k|s(x̄) + t(ȳ) or ¬(k|s(x̄) + t(ȳ));
Return ∃x_0, ..., ∃x_n.θ′
```

After introducing new variables and substitution, we obtain a formula where if an exponential term contains a quantified variable $x_i (0 \leq i \leq n)$, the term should be 2^{x_i}. We then collect all terms with $x_i (0 \leq i \leq n)$ together, it will be of the form $s(\bar{x}) \hat{=} \sum_{i=0}^{n} a_i 2^{x_i} + \sum_{i=0}^{n} b_i x_i$, where $a_i, b_i (0 \leq i \leq n)$ are all constants. Other terms including constants and terms of $\bar{y}$ are collected, denoted by $t(\bar{y})$. Since $\bar{y}$ are free variables, we will regard $t(\bar{y})$ as a constant. Inequalities and equalities will all be transformed into $s(\bar{x}) \leq t(\bar{y})$, for example, $s(\bar{x}) = t(\bar{y})$ will be replaced by $s(\bar{x}) \leq t(\bar{y}) \wedge t(\bar{y}) \leq s(\bar{x})$.

At the end, the resulted formula θ' only contains literals of the forms $s(\bar{x}) \leq t(\bar{y})$, $k|s(\bar{x})+t(\bar{y})$ and $\neg(k|s(\bar{x})+t(\bar{y}))$.

B. QE-with-order

During the Normalization step, we simplify the origin formula at the cost of more quantified variables. Suppose we get $\exists \bar{x}.\theta(\bar{x}, \bar{y})$, and it has n new variables x_i (x is denoted by x_0). As a consequence, we need to eliminate all quantified variables one by one.

To the end, we denote the set of all orders among the $n+1$ variables by S_{n+1}, and then enumerate these orders one by one in the outer for loop. For the considered order σ, we first add the ordering information to the quantifier free formula, and then according to the order, to eliminate $x_{\sigma(i)}$ for $i = n$ to 0 by invoking QE-exp first, and then invoking QE-linear (the inner for loop). In each iteration of the inner for loop, if $x_{\sigma(i)}$ occurs in an exponential term in an atomic formula, QE-exp is invoked and a formula equivalent to the atom is returned, in which x_i occurs linearly; then x_i occurs linearly in the formula, thus

Algorithm 2: QE-with-order

```
Input: a normalized (n+1)-existential formula
       ∃x̄.θ(x̄, ȳ), where x̄ = (x_0, ..., x_n)
Output: a equivalent quantifier free formula
        without x̄
Let S_{n+1} denote the group of permutations on
 {0, 1, ..., n};
φ := False;
for each σ ∈ S_{n+1} do
    θ_σ := θ(x̄, ȳ) ∧ ⋀_{j=0}^{n-1} (x_{σ(j)} ≤ x_{σ(j+1)});
    // recursively eliminate the maximal x_i in x̄
    for i from n to 0 do
        // if x_{σ(i)} occurs exponentially in θ_σ
        θ_σ := QE-exp(x_{σ(i)}, θ_σ);
        // now x_{σ(i)} occurs only linearly in θ_σ
        θ_σ := QE-linear(x_{σ(i)}, θ_σ);
    end
    φ := φ ∨ θ_σ;
end
output φ
```

QE-linear is further invoked to eliminate all occurrences of x_i. The returned formula is the disjunction of all formulas resulted in all iterations of the outer for loop.

C. QE-exp

QE-exp and QE-linear are the most technical parts of our QE algorithm. As mentioned, QE-exp takes $\theta_\sigma(\bar{x}, \bar{y})$ and variable $x_{\sigma(i)}$ (the maximal variable among all quantified variables that are not eliminated yet according to σ) as inputs, and outputs an equivalent formula in which $x_{\sigma(i)}$ occurs linearly. The ideal case is that the formula $\theta(\bar{x}, \bar{y})$ itself contains no $2^{x_{\sigma(i)}}$ terms, so we can omit this step and directly go to QE-linear. For simplicity, we will abuse x_i for $x_{\sigma(i)}$, and $\bar{x}$ for $(x_{\sigma(0)}, ..., x_{\sigma(i-1)})$ (remind that $x_{\sigma(i+1)}, ..., x_n$ have been eliminated already).

After normalization, the formula $\theta(\bar{x}, \bar{y})$ contains atoms of three forms corresponding to the predicates $\leq$, $|$ and negation of $|$, respectively. The problem will be discussed in 2 cases depending on the form of the atom τ that contains 2^{x_i}, corresponding to QE-exp-ineq or QE-exp-div.

We first give the whole procedure of QE-exp, and then provide the details of the two sub-routines.

1) QE-exp-ineq: This algorithm deals with the case where τ is an inequality of the form

$$\tau(x_i, \bar{x}, \bar{y}) \hat{=} a_i 2^{x_i} + \sum_{j=0}^{i-1} a_j 2^{dx_j} + \sum_{k=0}^{i} b_k x_k \leq t(\bar{y})$$

We now try to eliminate 2^{x_i} in τ, the idea is to find a bound for x_i, either by constants or by other variables. We will prove the following theorem.

Algorithm 3: QE-exp

```
Input: x_i and θ, x_i is larger than x_0 to x_{i-1} and
       occurs exponentially in θ
Output: a formula equivalent to θ where x_i occurs
        linearly
while there is an atom τ(x_i, x̄, ȳ) contains 2^{x_i} do
    Ψ := False;
    if τ is of the form
     a_i 2^{x_i} + Σ_{j=0}^{i-1} a_j 2^{x_j} + Σ_{k=0}^{i} b_k x_k ≤ t(ȳ) then
        // QE-exp-ineq case
        A := Σ_{j=0}^{i-1} |a_j|, B := b_i + Σ_{j=0}^{i-1} |b_j|;
        B' := 2(l_2(B) + 3), δ := l_2(A) + 3;
        If a_i > 0 then α := l_2(t(y)) − l_2(a_i) else
         α := l_2(−t(y)) − l_2(−a_i);
        ρ_1 := (x_i ≤ α−1 ∧ ⋁_{0≤k≤B'} (x_i = k ∧ τ[k/x_i]))
            ∨(x_i ≤ α − 1 ∧ x_i ≥ B' ∧ ⋁_{0≤k≤δ} (x_i =
         x_{i-1} + k ∧ τ[x_{i-1} + k/x_i]));
        ρ_2 := (x_i = α) ∧ τ[α/x_i];
        ρ_3 := (x_i = α + 1) ∧ τ[α + 1/x_i];
        ρ_4 := (x_i ≥ α+2 ∧ ⋁_{0≤k≤B'} (x_i = k ∧ τ[k/x_i]))
            ∨(x_i ≥ α + 2 ∧ x_i ≥ B' ∧ ⋁_{0≤k≤δ} (x_i =
         x_{i-1} + k ∧ τ[x_{i-1} + k/x_i]));
        if a_i > 0 then
            Ψ := ρ_1 ∨ ρ_2 ∨ ρ_3 ∨ ρ_4 ∨ [x_i ≤
             α − 1 ∧ x_i ≥ B' ∧ x_i ≥ x_{i-1} + δ]
        else
            Ψ := ρ_1 ∨ ρ_2 ∨ ρ_3 ∨ ρ_4 ∨ [x_i ≥
             α + 2 ∧ x_i ≥ B' ∧ x_i ≥ x_{i-1} + δ]
        end
    else
        // QE-exp-div case, the atom is of the form
           d|t(x_i, x̄, ȳ)
        // let d = 2^r d_0 where d_0 is odd
        ρ_5 := ⋁_{p=0}^{r-1} [τ(p, x̄, ȳ) ∧ x = p];
        ρ_6 := ⋁_{q=0}^{φ(d_0)-1} [d|(a_i 2^{r+q} + Σ_{j=0}^{i-1} a_j 2^{x_j} +
         Σ_{k=0}^{i} b_k x_k + t(ȳ)) ∧ φ(d_0)|(x_i − r − q) ∧ x_i ≥ r];
        Ψ := ρ_5 ∨ ρ_6
    end
    replace τ by Ψ in θ;
end
```

Theorem IV.1. Assume that x_i is the maximal variable in $\bar{x}$ according to the given order. Given an inequality $\tau(x_i, \bar{x}, \bar{y})$ of the form

$$\tau(x_i, \bar{x}, \bar{y}) \hat{=} a_i 2^{x_i} + \sum_{j=0}^{i-1} a_j 2^{dx_j} + \sum_{k=0}^{i} b_k x_k \leq t(\bar{y})$$

with $a_i \neq 0$, let $A \hat{=} \sum_{j=0}^{i-1} |a_j|$, $B \hat{=} |b_i| + \sum_{j=0}^{i-1} |b_j|$, $B' \hat{=} 2(l_2(B) + 3)$, $\delta \hat{=} l_2(A) + 3$, then

- if $a_i > 0$, let $\alpha \hat{=} l_2(t(y)) - l_2(a_i)$
  - if $x_i \leq \alpha - 1$, $x_i \geq B'$ and $x_i \geq x_{i-1} + \delta$, then $\tau(x_i, \bar{x}, \bar{y})$ holds.
  - if $x_i \geq \alpha + 2$, $x_i \geq B'$ and $x_i \geq x_{i-1} + \delta$, then $\tau(x_i, \bar{x}, \bar{y})$ does not hold.
- if $a_i < 0$, let $\alpha \hat{=} l_2(-t(y)) - l_2(-(a_i))$

- if $x_i \leq \alpha - 1$, $x_i \geq B'$ and $x_i \geq x_{i-1} + \delta$, then $\tau(x_i, \bar{x}, \bar{y})$ does not hold.
- if $x_i \geq \alpha + 2$, $x_i \geq B'$ and $x_i \geq x_{i-1} + \delta$, then $\tau(x_i, \bar{x}, \bar{y})$ holds.

Before proving Theorem 1, we need the following lemma to estimate linear terms.

Lemma IV.2. For any $n, m \in \mathbb{N}$, if $n \geq m \geq 1$ and $x \geq 2(l_2(n) - l_2(m) + 1)$, then $nx \leq m2^x$ holds.

Proof. First we can prove that for any $N \in \mathbb{N}$, $x \geq 2N \implies 2^N x \leq 2^x$. Let $N \hat{=} l_2(n) - l_2(m) + 1$, then we have $x \geq 2N \implies nx \leq 2\lambda_2(n)x \leq 2^N \lambda_2(m)x \leq m2^x$. □ □

Then we give the proof for Theorem 2.

Proof. We only prove for the $a_i > 0$ case, the other case is analogous. The goal is to find a bound for x_i such that the values of the atoms containing x_i keep constant when x_i is greater than the bound.

Note that x_i is the largest among $\bar{x}$. suppose $x_i > x_{i-1} + \delta$, and let $\delta \hat{=} l_2(A) + 3$, then

$$2^{-\delta} A = \frac{A}{8\lambda_2(A)} \leq \frac{1}{4}. \tag{4}$$

When $x_i \geq B' = 2(l_2(4B) - l_2(1) + 1)$, according to Lemma 1,

$$4Bx_i \leq 2^{x_i}. \tag{5}$$

When $x_i \geq \alpha + 2$,

$$\begin{aligned} a_i 2^{x_i} + \sum_{j=0}^{i-1} a_j 2^{x_j} + \sum_{k=0}^{i} b_k x_k &\geq a_i 2^{x_i} - 2^{x_i - \delta} A - B x_i \\ &\geq 2^{x_i}(a_i - \frac{1}{4} - \frac{1}{4}) && \text{(by (4) and (5))} \\ &\geq \frac{a_i}{2} \frac{4\lambda_2(t(\bar{y}))}{\lambda_2(a_i)} && \text{(Def. of } \alpha) \\ &\geq t(\bar{y}) \end{aligned}$$

So we conclude that $\tau(x_i, \bar{x}, \bar{y})$ keeps false in this case.

When $x_i \leq \alpha - 1$, similarly we have

$$\begin{aligned} \tau(x_i, \bar{x}, \bar{y}) &\leq a_i 2^{x_i} + 2^{x_i - \delta} A + B x_i \\ &\leq 2^{x_i}(a_i + \frac{1}{4} + \frac{1}{4}) && \text{(by (4) and (5))} \\ &\leq 2\lambda_2(a_i) \frac{\lambda_2(t(\bar{y}))}{2\lambda_2(a_i)} && \text{(Def. of } \alpha) \\ &\leq t(\bar{y}) \end{aligned}$$

which indicates that when $x_i \leq \alpha - 1$, $\tau(x_i, \bar{x}, \bar{y})$ keeps true. □ □

2) QE-exp-div: If x_i occurs exponentially in an atom $\tau(x_i, \bar{x}, \bar{y})$ of the form

$$\tau(x_i, \bar{x}, \bar{y}) \hat{=} d | a_i 2^{x_i} + \sum_{j=0}^{i-1} a_j 2^{x_j} + \sum_{k=0}^{i} b_k x_k + t(\bar{y}) \quad (a_i \neq 0),$$

the algorithm QE-exp-div outputs an equivalent formula without 2^{x_i} terms. The idea is that $a_i 2^x$ modulo d is a periodic function when x is large enough and the period can be computed.

Let $d = 2^r d_0$, d_0 is an odd natural number. According to Euler's Theorem, $gcd(2, d_0) = 1$ implies $2^{\phi(d_0)} \text{mod } d_0 = 1$, where ϕ is the Euler function. Consider function $f(n) = (2^n \text{ mod } d)$, when $n \geq r$, $f(n)$ becomes a periodic function and its period is a divisor of $\phi(d_0)$ because

$$\begin{aligned} f(n + \phi(d_0)) &= 2^{n+\phi(d_0)} \text{ mod } d \\ &= 2^n \cdot 2^{\phi(d_0)} \text{ mod } d \\ &= 2^n \cdot (kd_0 + 1) \text{ mod } d && (\text{assume } 2^{\phi(d_0)} = kd_0 + 1) \\ &= 2^n \text{ mod } d && (\text{when } n \geq r, 2^n d_0 \text{ mod } d = 0) \\ &= f(n) \end{aligned}$$

When $x_i \leq r - 1$, we just enumerate all possible value of x_i, i.e.,

$$\rho_5 \hat{=} \bigvee_{p=0}^{r-1} \tau(p, \bar{x}, \bar{y}).$$

When $x_i \geq r$, $\tau(x_i, \bar{x}, \bar{y})$ is equivalent to

$$\rho_6 \hat{=} \bigvee_{q=0}^{\phi(d_0)-1} [d | (a_i 2^{r+q} + \sum_{j=0}^{i-1} a_j 2^{x_j} + \sum_{k=0}^{i} b_k x_k + t(\bar{y})) \wedge \phi(d_0) | (x_i - r - q) \wedge x_i$$

Therefore, $\tau(x_i, \bar{x}, \bar{y})$ is equivalent to $\rho_5 \vee \rho_6$.

D. QE-linear

After QE-exp, we obtain a formula (still denoted by $\theta(x_i, \bar{x}, \bar{y})$) without 2^{x_i} terms, i.e., x_i occurs only linearly. Now we wish to construct a formula without x_i equivalent to $\exists x_i . \theta(x_i, \bar{x}, \bar{y})$. This can be done by following Cooper's QE algorithm for PA, treating all $x_j (j < i)$ as free variables (like $\bar{y}$). The procedure contains 3 steps.

Step 1: Put $\theta(x_i, \bar{x}, \bar{y})$ in NNF form, replace atomic formulae containing symbols other than $\leq, |$ by equivalent formulae only with $\leq$, that is, $x = a$ by $x \leq a \wedge a \leq x$, $x < a$ by $x \leq a + 1$, $x \neq a$ by $x > a \vee x < a$, $x \geq a$ by $-x \leq -a$, and $x > a$ by $-x \leq -a - 1$. In inequality atoms, collect terms of x_i to one side and guarantee the coefficients of x_i are positive.

Step 2: Let d be the least common multiple of all coefficients of x_i. For atoms with x_i, multiply all terms by a factor so that the coefficient of x_i is d. We introduce a fresh variable x'_i, replace all occurrence of dx_i by x'_i, and denote the resulted formula by $\theta'(x'_i)$. Note that x'_i is a multiple of d, so we set $\theta' = \theta' \wedge d | x'$.

We classify all atoms in θ into the following three sets: L denotes all atoms of the form $t_l(\bar{x}, \bar{y}) \leq x'_i$, U denotes all atoms of the form $x'_i \leq t_u(\bar{x}, \bar{y})$, M denotes all atoms of the form $k_m | x'_i + t_m(\bar{x}, \bar{y})$ and their negations, where $t(\bar{x}, \bar{y})$ with a subscript is any term of $\bar{x}$ and $\bar{y}$ and $k_m \in \mathbb{N}$ for $m \in M$.

Step 3: Let δ be the least common multiple of $\{k_m \mid m \in M\}$. Construct $\theta'_{+\infty}(x'_i)$ from $\theta'(x'_i)$ by replacing true for all atoms in L, and false for all atoms in U. Then $\exists x'_i . \theta'(x'_i)$ is equivalent to

$$\bigvee_{j=0}^{\delta-1} \theta'_{+\infty}[j / x'_i] \vee \bigvee_{j=0}^{\delta-1} \bigvee_{u \in U} \theta'[t_u(\bar{x}, \bar{y}) - j / x'_i]$$

Algorithm 4: QE-linear (Cooper's QE algorithm for PA)

Input: $x_i, \theta(x_i, \bar{x}, \bar{y})$, where x_i occurs linearly
Output: A equivalent formula without x_i
Collect terms of x_i in each atom;
Let d to be the least common multiple of all coefficients of x_i;
Multiply each atom by a factor so that coefficient of x_i is d;
Let $\theta'(x_i') := \theta[x_i'/dx_i]$;
$\theta'(x_i') := \theta'(x_i') \wedge d|x_i'$;
// Now the atoms have 3 different forms
// $t_l(\bar{x}, \bar{y}) \leq x_i'$, $x_i' \leq t_u(\bar{x}, \bar{y})$, $k_m|x_i' + t_m(\bar{x}, \bar{y})$(or its negation)
// where $l \in L, u \in U, m \in M$ are the index sets
$\delta := lcm\{k_m|m \in M\}$;
for all atom $\tau_l \in L$ do
  $\theta'_{+\infty} := \theta'\{\text{true}/\tau_l\}$
end
for all atom $\tau_u \in U$ do
  $\theta'_{+\infty} := \theta'_{+\infty}\{\text{false}/\tau_u\}$
end
output $\bigvee_{j=0}^{\delta-1} \theta'_{+\infty}[j/x_i'] \vee \bigvee_{j=0}^{\delta-1} \bigvee_{u \in U} \theta'[t_u(\bar{x}, \bar{y}) - j/x_i']$

The first disjunction $\bigvee_{j=0}^{\delta-1} \theta'_{+\infty}[j/x_i']$ corresponds to the case where x_i' is large enough so that all atoms in L are true and all atoms in U are false. Thus $\theta'_{+\infty}$ contains only divisibility literals. If there is a number $n (0 \leq n \leq \delta - 1)$ such that $\theta'_{+\infty}[n/x_i']$ is evaluated to be true, then for every $\lambda \in \mathbb{N}$, $\theta'_{+\infty}[n + \lambda\delta/x_i']$ is also evaluated to be true. Hence, there exists x_i' large enough to satisfy θ'.

The second disjunction corresponds to the case that for some $u \in U, x_i' \leq t_u(\bar{x}, \bar{y})$ holds. In this case, select the minimal $t_u(\bar{x}, \bar{y})$ from atoms in U that holds, then there exists a solution for x_i' such that x_i' is in the interval $[t_u(\bar{x}, \bar{y}) - \delta + 1, t_u(\bar{x}, \bar{y})]$.

Here we describe Cooper's algorithm in pseudo code. We will use $\theta\{\tau'/\tau\}$ to denote substitute all atoms τ with τ', distinguished from term substitution $\theta[t'/t]$.

E. Compared with Francoise's Origin Algorithm

Francoise's algorithm [?] has four steps, corresponding to the four steps of our algorithm respectively. So we will use the names of our steps to refer to them. The divisible predicate $n|x$ is not included in his theory, instead, the division operation $\frac{x}{n}$ where $n \in \mathbb{N}$ is allowed. The Normalization and QE-with-order steps in two algorithms are similar, with some minor differences due to the setting of the theory. The main differences between the two algorithms lies in the following aspects.

In QE-exp, we adopt the same idea to find sufficient conditions and necessary conditions for the formula to holds. But our strategy for choosing parameters are easier to understand and use. A flaw lies in the $a_i < 0$ case in Francoise's algorithm [1], our QE-exp-ineq corrects this flaw by treating $a_i < 0$ case similarly to the $a_i > 0$ case.

QE-linear in Francoise's algorithm in some sense includes our QE-exp-div, because divisible predicates is not introduced in his setting. In this step, Francoise uses the permutation groups (just like $S_{(n+1)}$ in QE-with-order) and needs recursively renaming of variables x_i. We instead invoke the QE algorithm of PA without referring to permutation groups, other improved QE algorithm can also be used to simplify this step.

Francoise's algorithm has more restrictions on the form of the formula. Before QE-linear and QE-exp step, it needs to transform the formula into disjunction normal form (DNF) and deals with the basic case when input formula θ is a conjunction of atoms. It is well known that transforming an arbitrary formula into DNF is costly and the length of the formula may increase exponentially. So our algorithm remove these limitations and assume no special forms of the formula other than NNF.

V. Complexity Analysis and Case Study

In this section, we give a complexity analysis of our decision procedure, and provide an example to demonstrate the procedure.

A. Complexity Analysis

In this part, we estimate the complexity of our algorithm. It can be divided into two part, the first part convert a string constraint ϕ to a PA formula with exponential functions, and the second part is the QE procedure on the obtained formula.

Given a finite alphabet Σ and $\alpha = \langle p, q \rangle$, the size of Σ is denoted by $|\Sigma|$. The generic α-flat language contains $O(|\Sigma|^{pq})$ α-flat languages

a) Complexity for QE: In QE-with-order, it reduces the origin problem to $(n+1)!$ sub-cases, and for each case, we recursively invoke QE-exp and QE-linear. So the key is to analysis the inner for loop in QE-with-order. Note that if we omit the QE-exp step, the sub-cases are equivalent to Cooper's algorithm, so we adopt the idea of Oppen's analysis to analyze the complexity [?].

Apply Cooper's algorithm on the quantified PA formula $Q_m x_m Q_{m-1}\ x_{m-1} ... Q_1 x_1.\ F(x_1, ..., x_m)$, the algorithm repeats m times to eliminate $x_1, ..., x_m$ one by one. Let $F_k \hat{=} Q_m x_m Q_{m-1} x_{m-1} ... Q_{k+1} x_{k+1} . F_k'(x_{k+1}, ..., x_m)$ be the formula produced after kth iterations of the algorithm. Let a_k denote the number of atoms in F_k, c_k denote the number of distinct δ_i in atom $\delta_i|t$ (and its negation) plus the number of distinct coefficients of variables. s_k denote the largest value of integer constant. Use a_0, c_0, s_0 to denote the initial value of a_k, c_k, s_k respectively. The following theorem holds.

[1] In the $a_i < 0$ case of QE-exp-ineq, Francoise's algorithm assumes falsely $t(y) \geq 0$. The reason for this mistake might be the confusion of two subtraction operators. The example mentioned in 3.2 shows this case.

Theorem V.1. for all $k : 1 \le k \le m$, the following relations holds

- $c_k \le {c_{k-1}}^4$
- $s_k \le {s_{k-1}}^{4c_{k-1}}$
- $a_k \le {a_{k-1}}^4 \cdot {s_{k-1}}^{2c_{k-1}}$

and from the above relations we have

- $c_k \le {c_0}^{4^k}$
- $s_k \le {s_0}^{4{c_0}^{4^k}}$
- $a_k \le {a_0}^{4^k} \cdot {s_0}^{4{c_0}^{4^k}}$

For each k, the space required to store F_k is $a_k \cdot (m+1) \cdot s_k \cdot q$, where $(m+1)$ is the maximum number of constants per atom and $q > 1$ is some constant. Assume $m \le n, c \le n, a \le n, s \le n$, we obtain the deterministic space complexity is $2^{2^{2^{pnlog_n}}}$, which is also the bound for deterministic time.

In our algorithm, we use the same denotations and have the following theorem. Note that each iteration corresponds to QE-exp and QE-linear.

Theorem V.2. for all $k : 1 \le k \le m$, the following relations holds

- $c_k \le {c_{k-1}}^4$
- $s_k \le (ns_{k-1})^{4c_{k-1}}$
- $a_k \le (l_2(ns_{k-1}) \cdot a_{k-1})^4 \cdot (ns_{k-1})^{2c_{k-1}}$

and from the above relations we have

- $c_k \le {c_0}^{4^k}$
- $s_k \le n^{4{c_0}^{4^k}} \cdot {s_0}^{4{c_0}^{4^k}}$
- $a_k \le {a_0}^{4^k} \cdot n^{4{c_0}^{4^k}} \cdot {s_0}^{4{c_0}^{4^k}}$

WAIT TO CHECK!

Again assume $m, c, a, s \le n$, which gives

$$\text{space} \le q \cdot n^{4^n} \cdot n^{(4n)^{4^n}} \cdot n^{(4n)^{4^n}} \cdot (n+1) \cdot \cdot n^{(4n)^{4^n}} \cdot n^{(4n)^{4^n}} \le 2^{2^{2^{pn\log n}}}$$

Then the upper bound for Cooper's algorithm still holds for the inner for loop of QE-with-order. When considering Normalization and permutations groups in QE-with-order, since the super exponential term dominates the number of permutations $n^{O(n)}$, this upper bound still holds.

B. Example

We apply our algorithm to a simple example to illustrate its function. Given $\Sigma = \{0, 1\}$ Suppose $\mathcal{A}$ is a finite state automaton that recognizes language $L = \{w \mid \#w('1') \text{ is odd}\}$. Consider the following string constraint

$$y \in L(\mathcal{A}) \wedge 4 \le \text{parseInt}(y) < 16 \,.$$

We need an abstraction parameter *alpha* to restrict interpretations for x, for simplicity, set $\alpha \hat{=} \langle 2, 1 \rangle$. The generic $\langle 2, 1 \rangle$-flat language is the union of $\langle 2, 1 \rangle$-flat languages $(11)^*, (10)^*, (01)^*, (00)^*, 1^*, 0^*$. Here we divide the problem into 6 sub-cases by restricting y to one of the $\langle 2, 1 \rangle$-flat language. Another approach is to directly restrict y to the generic $\langle 2, 1 \rangle$-flat language, at the cost of more introduced variables. Since the complexity is related to the number of variables, we wish it to be as little as possible.

When y is in language 1^*, assume $y = 1^x$ where x is a fresh integer variable. Note that the Parikh image of $L(\mathcal{A})$ is $2|(x+1)$, so the constraint is converted to a PA formula ϕ

$$\phi \hat{=} \exists x.2|(x+1) \wedge 5 \le 2^x < 17 \,.$$

Observe that when $x = 3$ the formula holds, so we can easily deduce that ϕ is true. By applying our algorithm, after Normalization, ϕ is transformed into the form

$$\exists x_0.2|(x+1) \wedge 2^x \le 16 \wedge -2^x \le -5 \,.$$

Since there is only one variable, QE-with-order can be omitted. We send $2^x \le 16$ and $-2^x \le -5$ to QE-exp and obtain

$$2^x \le 16 \equiv x \le 4$$

$$-2^x \le -5 \equiv 3 \le x \,.$$

Then we apply QE-linear on formula $\exists x_0.2|(x+1) \wedge x \le 4 \wedge 3 \le x$ and get

$$\bigvee_{j=0}^{1} (2|j+1 \wedge \text{false} \wedge \text{true}) \vee \bigvee_{j=0}^{1} (2|4-j+i \wedge 4-j \le 4 \wedge 3 \le 4-j) \,.$$

which evaluates true and we can deduce $j = 1 \implies x = 3$, so y is string 111.

Other sub-cases are similar. However, when the abstraction parameter becomes larger, say, $\alpha = \langle 2, 2 \rangle$, the complexity will grow rapidly, because QE-with-order can not always be omitted and formulae in QE-exp can not be simplified easily.

VI. Conclusion and Future Works