Satisfiability of Context-Free String Constraints with Subword-Ordering and Transducers

C. Aiswarya
Chennai Mathematical Institute, India
CNRS, ReLaX, IRL 2000, Chennai, India

Soumodev Mal
Chennai Mathematical Institute, India

Prakash Saivasan
The Institute of Mathematical Sciences, HBNI, Chennai, India
CNRS, ReLaX, IRL 2000, Chennai, India

Abstract

We study the satisfiability of string constraints where context-free membership constraints may be imposed on variables. Additionally a variable may be constrained to be a subword of a word obtained by shuffling variables and their transductions. The satisfiability problem is known to be undecidable even without rational transductions. It is known to be NEXPTIME-complete without transductions, if the subword relations between variables do not have a cyclic dependency between them. We show that the satisfiability problem stays decidable in this fragment even when rational transductions are added. It is 2NEXPTIME-complete with context-free membership, and NEXPTIME-complete with only regular membership. For the lower bound we prove a technical lemma that is of independent interest: The length of the shortest word in the intersection of a pushdown automaton (of size $\mathcal{O}(n)$) and n finite-state automata (each of size $\mathcal{O}(n)$) can be double exponential in n.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases satisfiability, subword, string constraints, context-free, transducers

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.5

Related Version *Full Version*: https://arxiv.org/abs/2401.07996

Funding *Prakash Saivasan*: MATRICS GRANT (MTR/2022/000312).

Acknowledgements We thank Paul Gastin for helpful discussions.

1 Introduction

The theory of strings has always been an important and active area of research for long. In fact, as Hilbert notes, it is the very foundation of mathematical logic itself [45, 25]. The recent successes in employing the theory for practical verification has only re-iterated its importance. The study of the theory of string constraints dates back to Tarski and Hermes [44, 33], who in 1933 provided the axiomatic foundation for it. There have been several other advancements of string theories since then, some of the notable ones include [25, 43, 41, 40, 42, 18]. In 1977, Makanin studied the algorithmic aspect of the word equations (equation involving concatenation and equality) and showed that the satisfiability problem is decidable [40]. The complexity for this problem was improved in [42]. Despite receiving much attention, the theory of strings has long standing unsolved open problems, indicating the intrinsic difficult nature of the theory.

One important aspect of the study here is the satisfiability of string constraints. The question here asks whether it is possible to assign a word to each variable such that the given set of string constraints is satisfied. The constraints themselves can be either relational, which relate variables or membership, that define the domain for each variable.

In the recent years, the constraint satisfaction problem of strings (CSPS) has received much attention from verification community due to its usefulness in modeling and reasoning about programs. This problem has particularly been useful in verifying web services [32] and database applications from injection attacks [11]. In such attacks, the attacker constructs an input string in such a way that the underlying semantics of the interpretation is changed. The CSPS, and more importantly its implementations in solvers [38, 1, 17, 36, 37, 26, 35] have provided the much needed power to model and verify programs for such vulnerabilities. This in turn has directed the study to explore the boundaries of solvability.

However one impediment for this has been the theoretical limitation. For instance, with respect to word equations, adding a transducer renders the model undecidable. Similarly introducing membership in context free language also renders the model undecidable (see [28], [30] for more details). Despite this, there have been several advancements in this regard [20, 39, 34, 23, 22, 19, 5, 8, 29].

The context-free membership constraints are particularly useful feature to have since checking vulnerabilities include checking for programs, that are inherently context-free, masquerading as string queries. In [9], the authors provided first such model that could handle context-free membership queries and yet has decidability for CSPS, under some restrictions. They showed that if every relational constraint has sub-word relation instead of equality and assuming an *acyclicity* restriction, the satisfaction problem is NExptime complete. In fact, the authors in their model include a more powerful shuffle operator against the usual concatenation. Further they show that the complexity of the satisfiability problem when only regular membership is involved is also the same i.e, NExptime complete. They also provide an interesting connection of their model with lossy channel systems that include pushdown automata.

Yet another feature in string solvers that has been much desired is that of transductions. As noted in [34, 22], most modern applications, especially browsers include implicit transductions that mutates the input string. To verify such applications, one also needs the power of transductions. There have been very few successful attempt towards decidability of string constraints that involve transductions, some of them being [34, 8, 22, 20].

We investigate string constraints when sub-word ordering, context-free membership and transducers are involved. Unfortunately, in its full generality this problem is undecidable. However we show that imposing the same acyclicity restriction as in [9] gives decidability under this setting. This extends the decidability result of [9] to include transductions.

In [9] the satisfiability of the acyclic variant of the string constraints without transducers was shown to be inter-reducible with the control-state reachability problem of acyclic networks of pushdown systems communicating over lossy fifo channels. They showed that both these problems are NExptime-complete. In our setting, with the additional feature of transductions, we can enrich the model of communicating pushdown to allow transductions to be sent in the channels. Such transductions naturally model encoders such as error correcting codes or injection of noise.

We show that, when only regular membership is allowed, adding transductions do not alter the complexity. It is still NExptime-complete. Interestingly when context-free membership is involved, it becomes 2NExptime-complete.

Our 2NExptimelower bound argument relies on a new technique that is of independent interest. In fact, we show that we can count exactly 2^{2^n} using one pushdown automaton with a binary stack alphabet and 3 states, and n finite state automata each of size $\mathcal{O}(n)$. Along the way we also show that 1) we can count exactly 2^n using a pushdown automaton with $\mathcal{O}(n)$ states and a binary stack alphabet, and 2) we can count exactly 2^n using n finite state automata each of size $\mathcal{O}(n)$.

As an application of this, we obtain a tight bound on the size of the smallest DFA of the downward closure, upward closure and the Parikh image closure of the intersection language of n finite state automata, each of size $\mathcal{O}(n)$. This size is $\Theta(2^n)$. Likewise, the size of the smallest DFA of the downward, upward and Parikh image closure for the intersection of language of n finite state automata with the language of a pushdown automaton, each of size $\mathcal{O}(n)$ is $\Theta(2^{2^n})$.

Related work

Apart from the work mentioned in the introduction, there are several other work on string constraints. In [19], the authors consider word equations equipped with replace all function and show decidability for the acyclic fragment.

In [3], the authors develop an uniform framework to decide the satisfiability and unsatisfiability of string constraints based on identifying patterns. In [2], the authors consider string constraints extended with negation and show how to solve them. In [23], the authors provide a semantic restriction on string manipulating programs that guarantees decidability for checking path feasibility. In [7], the authors study the problem of regular separability of the language of two word equations. In [27], the authors compare the expressive power of the logical theories built around word equations.

In [34], word equations with equality, transducers and regular membership is considered. This problem in full generality is undecidable. The authors consider a straight line fragment and show that the satisfiability problem is Expspace complete. In [23], the authors investigated the decidability of string constraints in the presence of regular membership constraints, `replaceAll` operator involving regular expressions and straight line restriction. In [21], the authors consider a stronger match and replace operator and show decidability. In [8], word equations with equality, transducers, length constraints and regular membership is considered and a chain free fragment of it was shown to be decidable. The authors show that the chain-free fragment of the satisfiability problem in this setting is decidable.

All of these work consider word equation (uses equality for comparison) in the model, our work uses subword ordering as the comparison operator. Further more, none of the work mentioned above considers context free membership constraints. In [9], subword ordering and context free membership is considered, where as it does not include transductions.

Apart from these, there are several approaches which attempts to solve the problem from a practical perspective, some of them being [4, 6, 2, 5, 17, 16, 34].

2 Preliminaries

Sets, Multisets, Functions

We denote the set of natural number $\{1, 2, \dots\}$ by $\mathbb{N}$. For $n \in \mathbb{N}$, we denote by $[n]$ the set of natural numbers up to n: $\{1, 2, \dots, n\}$. Let $\mathbb{N}_0$ denote the set $\{0, 1, 2, \dots\}$. That is, $\mathbb{N}_0 = \{0\} \cup \mathbb{N}$.

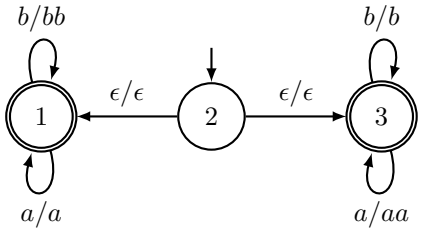

Figure 1 A transducer. Here the label x/y on a transition t indicates that $\mathsf{label}(t) = x$ and $\mathsf{out}(t) = y$. It nondeterministically chooses to duplicates as leaving bs as such, or duplicates bs leaving as as such.

Let S be any set. A *multiset* X of S assigns a multiplicity $X(s) \in \mathbb{N}_0$ to each element $s \in S$. We say that $s \in X$ if $X(s) > 0$. For a usual subset X, the multiplicity $X(s) \in \{0, 1\}$. A multiset X may also be written as $\{\!\{s_1, s_2, \dots\}\!\}$, by listing each element s, $X(s)$ many times. The set of all multisets of S is denoted $\mathbb{N}_0^S$, and the set of all usual subsets of S is denoted by 2^S. The size of a multiset X, denoted $|X|$ is the sum of the multiplicities of the elements. That is, $|X| = \sum_{s \in X} X(s)$.

Word, Subword, Shuffle, Projection

Let Σ be an alphabet. Σ^* denotes the set of all words over Σ, ϵ denotes the empty word, and $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$. For a word $w = a_1 a_2 \dots a_n \in \Sigma^*$, we denote by $\mathsf{len}(w)$, the *length* of w ($\mathsf{len}(w) = n$) and by $w[i]$ its ith letter a_i. The set of positions of w is denoted $\mathsf{pos}(w)$. That is, $\mathsf{pos}(w) = [\mathsf{len}(w)]$. For $Y \subseteq \mathsf{pos}(w)$, we denote by $w_{\downarrow Y}$ the *projection* of w to the positions in Y. If $Y = \{i_1, i_2, \dots i_m\}$ with $0 < i_1 < i_2 < \dots < i_m \leq n$, then $w_{\downarrow Y} = a_{i_1} a_{i_2} \dots a_{i_m}$. For $u, v \in \Sigma^*$, we say u is a *(scattered) subword* of v, denoted $u \preceq v$, if there is $Y \subseteq \mathsf{pos}(v)$ such that $u = v_{\downarrow Y}$. In this case we say v is a *superword* of u. Let $\Sigma' \subseteq \Sigma$ be a sub-alphabet and let $w \in \Sigma^*$. Projection of w to Σ', denoted $w_{\downarrow \Sigma'}$, is defined to be $w_{\downarrow Y}$ where $Y = \{i \mid w[i] \in \Sigma'\}$.

Let X be a finite multiset of words from Σ^* given by $X = \{\!\{w_1, \dots w_n\}\!\}$. We define the *shuffle* of X, denoted $\mathsf{Shuffle}(X)$ to be the set $\{w \mid$ there are $Y_1, Y_2 \dots Y_n \subseteq \mathsf{pos}(w)$ forming a partition of $\mathsf{pos}(w)$ and $w_i = w_{\downarrow Y_i}$ for all $i \in [n]\}$.

Finite-state automaton, Transducers, Pushdown Automaton

A (nondeterministic) *finite-state automaton* (NFA) over an alphabet Σ is given by a tuple $A = (\mathsf{States}, \mathsf{Trans}, s_{\text{in}}, F)$ where States is the finite set of states, $\mathsf{Trans} \subseteq \mathsf{States} \times \Sigma_\epsilon \times \mathsf{States}$ is the set of transitions, $s_{\text{in}} \in \mathsf{States}$ is the initial state, and $F \subseteq \mathsf{States}$ is the set of final/accepting states. We write $s \xrightarrow{t} s'$ for some $t \in \mathsf{Trans}$ if t is of the form (s, a, s'). Define the homomorphism $\mathsf{label} : \mathsf{Trans}^* \to \Sigma^*$ given by $\mathsf{label}((s, a, s')) = a$. The *language* of an NFA A, denoted $L(A)$ is given by $L(A) = \{w \mid w = \mathsf{label}(t_1 t_2 \dots t_n)$ and $s_{\text{in}} \xrightarrow{t_1} s_1 \xrightarrow{t_2} s_2 \dots s_{n-1} \xrightarrow{t_n} s_n$ with $s_n \in F\}$.

A *transducer* from Σ^* to Σ^* is a tuple $T = (\mathsf{States}, \mathsf{Trans}, s_{\text{in}}, F, \mathsf{out})$ where $A = (\mathsf{States}, \mathsf{Trans}, s_{\text{in}}, F)$ is an NFA, and $\mathsf{out} : \mathsf{Trans} \to \Sigma^*$ defines the outputs on each transition. The function out defines a homomorphism $\mathsf{out} : \mathsf{Trans}^* \to \Sigma^*$. The *relation* $R \subseteq \Sigma^* \times \Sigma^*$ recognized by T, denoted $R(T)$ is given by $\{(u, v) \mid u = \mathsf{label}(t_1 t_2 \dots t_n), v = \mathsf{out}(t_1 t_2 \dots t_n)$ and $s_{\text{in}} \xrightarrow{t_1} s_1 \xrightarrow{t_2} s_2 \dots s_{n-1} \xrightarrow{t_n} s_n$ with $s_n \in F\}$. The equality relation is realised by a transducer T_{id}. A transducer is depicted in Figure 1.

A *pushdown automaton* over Σ is given by a tuple $P = (\mathsf{States}, \mathsf{Trans}, s_{\text{in}}, F, \mathsf{op}, \Gamma)$ where $A = (\mathsf{States}, \mathsf{Trans}, s_{\text{in}}, F)$ is an NFA, Γ is the finite set of stack symbols, and $\mathsf{op} : \mathsf{Trans} \to \mathsf{Ops}$ defines the stack operation of each transition, where $\mathsf{Ops} = \{\mathsf{push}(\gamma) \mid \gamma \in \Gamma\} \cup \{\mathsf{pop}(\gamma) \mid \gamma \in \Gamma\} \cup \{\mathsf{nop}\}$. When depicting the pushdown automaton pictorially, we represent a transition $t = (s, a, s')$ as $s \xrightarrow{a|op(t)} s'$. When $\mathsf{op}(t) = \mathsf{nop}$, we may simply write $s \xrightarrow{a} s'$. Further if $a = \epsilon$ then we may write it as $s \xrightarrow{op(t)} s'$. A configuration of a PDA is a pair $(s, w) \in \mathsf{States} \times \Gamma^*$, indicating the current state and the stack contents. For two configurations (s, w) and (s', w') we write $(s, w) \xrightarrow{t} (s', w')$ for some $t \in \mathsf{Trans}$ if t is of the form (s, a, s') and 1) $\mathsf{op}(t) = \mathsf{push}(\gamma)$ and $w' = \gamma \cdot w$, or 2) $\mathsf{op}(t) = \mathsf{pop}(\gamma)$ and $w = \gamma \cdot w'$, or 3) $\mathsf{op}(t) = \mathsf{nop}$ and $w = w'$. The *language* of a PDA P, denoted $L(P)$ is given by $L(P) = \{w \mid w = \mathsf{label}(t_1 t_2 \dots t_n)$ and $(s_{\text{in}}, \epsilon) \xrightarrow{t_1} (s_1, w_1) \xrightarrow{t_2} (s_2, w_2) \dots (s_{n-1}, w_{n-1}) \xrightarrow{t_n} (s_n, \epsilon)$ with $s_n \in F\}$.

The set of all NFA / transducers / PDA over the alphabet Σ is denoted $\mathsf{NFA}(\Sigma)$ / $\mathsf{TRANSD}(\Sigma)$ / $\mathsf{PDA}(\Sigma)$. A language $L \subseteq \Sigma^*$ is said to be context-free (resp. regular) if there is a PDA (resp. NFA) A such that $L = L(A)$. A relation $R \subseteq \Sigma^* \times \Sigma^*$ is said to be rational if it is recognized by some transducer T.

Given an NFA A (resp. transducer T), its number of states is denoted by $\mathsf{state\text{-}size}(A)$ (resp. $\mathsf{state\text{-}size}(T)$). Given a PDA P by $\mathsf{state\text{-}size}(P)$ we denote the sum of the number of states and number of stack symbols. That is $\mathsf{state\text{-}size}(P) = |\mathsf{States}| + |\Gamma|$.

3 String constraints

A string constraint over a set of variables $\mathcal{V}$ and an alphabet Σ is given by a set of membership constraints and a set of subword ordering constraints. The membership constraint is given by associating a pushdown automaton to each variable, indicating that the word assigned to the variable must belong to the language of the pushdown automaton. A subword order constraint is given by a pair (x, Y) where $x \in \mathcal{V}$ and Y is a finite multiset over $\mathcal{V} \times \mathsf{TRANSD}(\Sigma)$.

For example, the constraint $(x, \{\!\{(y, T_1), (y, T_2), (y, T_2), (z, T_2)\}\!\})$ means that the words assigned to x, y and z, say w_x, w_y and w_z respectively, must satisfy $w_x \preceq w$ for some $w \in \mathsf{Shuffle}(\{\!\{u_1, u_2, u_3, u_4\}\!\})$, where $(w_y, u_1) \in R(T_1)$, $(w_y, u_2) \in R(T_2)$, $(w_y, u_3) \in R(T_2)$, and $(w_z, u_4) \in R(T_2)$. Note that the transducers can be identity in which case the input and the output are the same. For instance, if $T_1 = T_{\text{id}}$ then u_1 must be same as w_y.

We sometimes denote the constraint (x, Y) by $x \preceq \mathsf{Shuffle}(Y)$. Abusing notation, we may write a pair $(x, T) \in \mathcal{V} \times \mathsf{TRANSD}(\Sigma)$ as $T(x)$. If $Y = \{\!\{(y, T)\}\!\}$ (i.e., a singleton), then we may simply write $x \preceq T(y)$ instead of $x \preceq \mathsf{Shuffle}(\{\!\{T(y)\}\!\})$. Further, we may simply write x for (x, T_{id}). For instance, $(x, \{\!\{(y, T_{\text{id}})\}\!\})$ may be also written as $x \preceq y$.

▶ **Definition 1.** *A string constraint C is a tuple $(\Sigma, \mathcal{V}, \mathsf{Mem}, \mathsf{Rel})$ where $\mathsf{Mem} : \mathcal{V} \to \mathsf{PDA}(\Sigma) \cup \mathsf{NFA}(\Sigma)$ assigns a PDA or an NFA to each variable, and $\mathsf{Rel} \subseteq \mathcal{V} \times \mathbb{N}_0^{\mathcal{V} \times \mathsf{TRANSD}(\Sigma)}$ is a finite set of subword-order constraints.*

We denote by $\textsc{trset}(C)$ the finite set of transducers occurring in the string constraint C. That is, $\textsc{trset}(C) = \{T \mid \exists (x, Y) \in \mathsf{Rel}, y \in \mathcal{V}, (y, T) \in Y\}$. Similarly, $\textsc{autset}(C)$ is the finite set of PDA/NFA occurring in C. That is, $\textsc{autset}(C) = \{\mathsf{Mem}(x) \mid x \in \mathcal{V}\}$. A string constraint is *regular* if for every $v \in \mathcal{V}$, $\mathsf{Mem}(v)$ is an NFA, (equivalently, if $\textsc{autset}(C) \subseteq \mathsf{NFA}(\Sigma)$). An important parameter for our complexity considerations will be the number of times a variable is used in the right hand side (RHS). We denote it by $\mathsf{multiplicity}_C(x) = \sum_{T \in \textsc{trset}(C), (y,Y) \in \mathsf{Rel}} Y((x, T))$. We omit the subscript and simply write $\mathsf{multiplicity}(x)$ when C is clear from the context.

▶ **Definition 2.** *A string constraint C is* satisfiable *if there exists an assignment $\sigma : \mathcal{V} \to \Sigma^*$ that satisfies every membership and relational constraints in C – that is,*

1. $\sigma(v) \in L(\mathsf{Mem}(v))$ *for all* $v \in \mathcal{V}$
2. *For every* $(x, Y) \in \mathsf{Rel}$*, if* $Y = \{\!\{(y_1, T_1), (y_2, T_2), \dots (y_n, T_n)\}\!\}$*, then there are words* $u_1, u_2, \dots, u_n$ *such that* $(\sigma(y_i), u_i) \in R(T_i)$ *for each* $i \in \{1, 2, \dots, n\}$*, and there is a word* $w \in \mathsf{Shuffle}(\{\!\{u_1, u_2, \dots, u_n\}\!\})$ *such that* $\sigma(x) \preceq w$ *.*

Such an assignment σ is called a satisfying assignment*.*

▶ **Example 3.** Consider a string constraint on two variables x and y. The membership constraints are as follows. $\mathsf{Mem}(x)$ is an NFA for $\{ababab\}$, and $\mathsf{Mem}(y)$ is an NFA for $\{ab\}$. There is only one relational constraint: $x \preceq \mathsf{Shuffle}(\{\!\{(y, T)(y, T)\}\!\})$, where T is the transducer defined in Figure 1. This string constraint is satisfiable.

▶ **Definition 4** (Satisfiability Problem for String Constraints)**.**
Input: *A string constraint C.*
Question: *Is C satisfiable?*

The satisfiability problem is undecidable already for regular string constraints without transducers (or, equivalently, when only T_{id} is allowed) [9]. To circumvent undecidability, acyclic fragment of string constraints were considered in [9]. Formally, let $x < y$ if $(x, Y) \in \mathsf{Rel}$ with $(y, T) \in Y$ for some transducer T. The string constraint is acyclic if $<$ is acyclic. For the acyclic fragment without transducers, satisfiability was shown in [9] to be NExptime-complete. The lower bound already holds for regular acyclic string constraints without transducers.

We study the satisfiability problem for acyclic string constraints in the presence of transducers. Our main results are:

▶ **Theorem 5.** *Satisfiability problem for acyclic context-free string constraints with transducers is* 2NExptime*-complete.*

▶ **Theorem 6.** *Satisfiability problem for acyclic regular string constraints with transducers is* NExptime*-complete.*

▶ Remark 7. Our result shows an interesting contrast with string equations (with equality instead of subword order in relational constraints). Satisfiability of string equations (with concatenation, no shuffle) is decidable, when regular membership constraints are allowed. Adding transducers on top however render the satisfiability undecidable. In our setting, where subword order is used instead of equality, adding transducers to the acyclic fragment retains decidability.

▶ Remark 8. Without transducers, regular and context-free string constraints have the same complexity. In the presence of transducers they are in different complexity classes.

▶ Remark 9. It was shown in [9] that concatenation can be expressed by shuffle. This simulation is only linear and furthermore it preserves acyclicity. Thus our complexity upper bounds already hold for string constraints which uses the more popular concatenation operation instead of shuffle. Interestingly, the lower bounds in Theorem 5 and Theorem 6 already hold for the variant without shuffle.

In Section 4 and Section 5 we prove the lower bound and upper bound claimed in Theorem 5 respectively. The proof of Theorem 6, as well as other missing details can be found in the full version of this paper [10]. In Section 6, we discuss some implications of our results and conclude.

4 2NEXPTIME Hardness

We prove the hardness by giving a reduction from a bounded variant of the PCP problem that is 2NExptime-complete.

4.1 (Double-exponentially) Bounded PCP problem

In this decidable variant of the PCP problem, we are also given a parameter ℓ as part of the input in unary, and we ask whether there is a solution of length 2^{2^ℓ}. Formally the problem is stated as follows.

▶ **Definition 10** ((Double-exponentially) Bounded PCP problem (2eBPCP)).
Input: $(\Sigma_1, \Sigma_2, f, g, \ell)$ *where* Σ_1 *and* Σ_2 *are two disjoint finite alphabets,* $f, g : \Sigma_1 \to \Sigma_2^*$ *are two functions which naturally extend to a homomorphism from* $\Sigma_1^* \to \Sigma_2^*$*, and* $\ell \in \mathbb{N}$ *is a natural number.*
Question: *Is there a word* $w \in \Sigma_1^+$ *with* $|f(w)| = 2^{2^\ell}$ *and* $f(w) = g(w)$ *?*

The above problem is 2NExptime-complete. The proof can be found in the full version [10]. If the problem asked for the length of $f(w)$ to be ℓ, it would be NP-complete [31], and if it was 2^ℓ it would be NExptime-complete [9].

▶ **Theorem 11.** *(Double-exponentially) Bounded PCP problem is* 2NExptime*-complete.*

4.2 Towards a reduction

Our idea is to use 4 variables x_1, x_2, x_f, x_g. The membership constraint for x_f is a PDA for the language $L_f = \{w \cdot \#^* \cdot f(w^r) \mid w \in \Sigma_1^*\}$, and that for x_g is a PDA for the language $L_g = \{w \cdot \#^* \cdot g(w^r) \mid w \in \Sigma_1^*\}$. Recall that w^r denotes the reverse of w, and $\#$ is a special symbol not in Σ_1 or Σ_2. Suppose x_1 and x_2 are constrained to the language $\Sigma_1^m \#^n \Sigma_2^{2^{2^\ell}}$ such that $m + n = 2^{2^\ell}$, by polynomial-sized constraints. Then with the relational constraints 1) $x_1 \preceq x_f$ 2) $x_f \preceq x_g$ and 3) $x_g \preceq x_2$, we will achieve our reduction. Recall that $x \preceq y$ is a short hand for $x \preceq \mathsf{Shuffle}(\{\!\{(y, T_{\mathrm{id}})\}\!\})$. Indeed these constraints are satisfiable if and only if the 2eBPCP has a solution.

Notice that our constraints for x_1 and x_2 requires *counting* exactly 2^{2^ℓ}. This is not possible with a polynomial-sized PDA. In the above paragraph we did not use transducers either. Without transducers, the satisfiability problem of string constraints is not 2NExptime-hard, it is indeed in NExptime[9].

However, with the help of ℓ many transducers (or FSA) of size $\mathcal{O}(\ell)$ we can have a PDA that counts 2^{2^ℓ}. We will describe this technique with PDA and DFA in the next subsection, and in the following subsection using this idea, we complete the reduction.

4.3 Counting 2^{2^ℓ} using one PDA and ℓ DFA

Let $\Gamma_1 = \{0, 1, \mathsf{inc}, \mathsf{dec}\}$, and let Γ_2 be another finite alphabet disjoint from Γ_1. Our objective is to come up with a PDA A and ℓ DFAs $A_1, A_2, \ldots A_\ell$ over the alphabet $\Gamma_1 \cup \Gamma_2$, each of size $\mathcal{O}(\ell)$ such that any word accepted by all of them (i.e., in $\cap_i L(A_i) \cap L(A)$) has 2^{2^ℓ} occurrences of letters from Γ_2.

First we give a PDA with $3n + 3$ states and stack symbols $\{0, 1\}$ that accepts $(\Gamma_2)^{2^{n+1}}$.

▷ Claim 12. There is a PDA with $3n + 3$ states and stack symbols $\{0, 1\}$ with stack-height never exceeding n that accepts $(\Gamma_2)^{2^{n+1}}$.

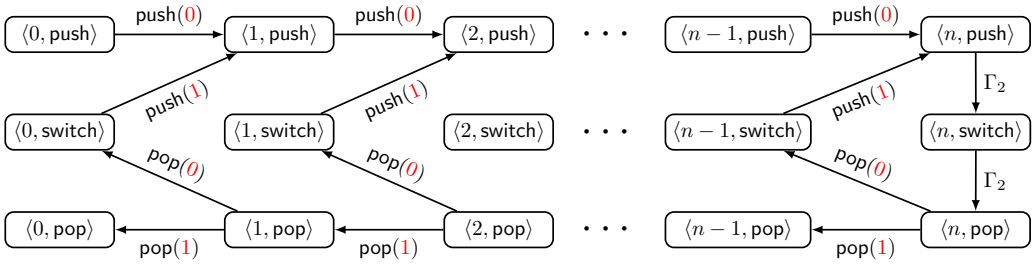

Figure 2 A PDA with $3n + 3$ states that accepts $(\Gamma_2)^{2^{n+1}}$.

Proof. Such a PDA is depicted in Figure 2. In this PDA the stack height never exceeds n. The PDA has three modes - a push mode where it keeps pushing 0s until the stack height is n, a pop mode where it keeps popping the symbol 1, and a switch mode that switches from a pop mode to push mode by replacing a 0 on the top of the stack by a 1. The states then represent the current stack height and the mode.

When the PDA is in the state $\langle n, \mathsf{push}\rangle$, the stack contents represents an n-bit binary number. At this point it reads two symbols from Γ_2 and goes to the state $\langle n, \mathsf{pop}\rangle$. From there, it does a sequence of transitions such that the next time it reaches $\langle n, \mathsf{push}\rangle$, the binary number in the stack would be incremented. For this it replaces the 01^m suffix with a 10^m.

The initial state is $\langle 0, \mathsf{push}\rangle$. The first time it reaches $\langle n, \mathsf{push}\rangle$, the stack content would be 0^n. The last time it reaches $\langle n, \mathsf{push}\rangle$ (or $\langle n, \mathsf{pop}\rangle$) the stack content would be 1^n, and from there it reaches $\langle 0, \mathsf{pop}\rangle$ by popping the entire stack. The state $\langle 0, \mathsf{pop}\rangle$ is accepting. Note that this PDA reaches the state $\langle n, \mathsf{push}\rangle$ exactly once for every n-bit binary number, and each time it reaches $\langle n, \mathsf{push}\rangle$ it reads two symbols from Γ_2. Thus any accepting run will read 2^{n+1} Γ_2 symbols. ◁

If $n = 2^\ell - 1$ we will be able to get 2^{2^ℓ} length words from Γ_2 as we wanted. However, we are allowed to use only $\mathcal{O}(\ell)$ states. To overcome this, we will use ℓ length binary numbers to indicate the current stack height. We then use ℓ DFAs, one for each bit, to update the binary numbers representing the stack height as required. We will next describe the ℓ DFAs that succinctly record the stack height. We then give a PDA that, along with these ℓ DFAs, accepts words with 2^{2^ℓ} occurrences of letters from Γ_2.

Let $\{\mathsf{inc}, \mathsf{dec}\}$ disjoint from Γ_2 be the increment and the decrement operators on integers. Further, we may treat symbols from Γ_2 as "keep unchanged" operators. That is, $\mathsf{inc}(n) = n + 1$, $\mathsf{dec}(n) = n - 1$ and $a(n) = n$ for all $a \in \Gamma_2$. Consider the following language over the alphabet $\Gamma = \{0, 1, \mathsf{inc}, \mathsf{dec}\} \cup \Gamma_2$ of alternating sequences of ℓ-bit numbers and operators, where each operator when applied on the previous number gives the next number. Here, the binary numbers are written with the most-significant bit on the left. That is $\mathsf{val}(b_1 b_2 \dots b_\ell) = \sum_i b_i \times 2^{\ell - i}$.

$$L_\ell = \{n_0 o_1 n_1 o_2 n_2 \cdots o_k n_k \quad | \quad k \geq 0, n_i \in (0+1)^\ell \text{ for all } i : 0 \leq i \leq k \\ o_i \in \{\mathsf{inc}, \mathsf{dec}\} \cup \Gamma_2 \text{ for all } i : 0 < i \leq k \\ \mathsf{val}(n_i) \equiv o_i(\mathsf{val}(n_{i-1})) \mod 2^\ell \text{ for all } i : 0 < i \leq k\}$$

▷ Claim 13. There are ℓ DFAs $B_1, B_2, \dots B_\ell$, each with $\mathcal{O}(\ell)$ states such that $L_\ell = \bigcap_i L(B_i)$.

Proof. We describe the ℓ DFAs $B_1, B_2, \dots B_\ell$ below.

The ith DFA B_i guarantees that the ith bit takes the correct value. This DFA is depicted in the Figure 3. The automaton has two disconnected "forks" (the top one starting at s_0^i and the bottom one starting at s_1^i). In the top fork, the ith bit read is always 0, and in the

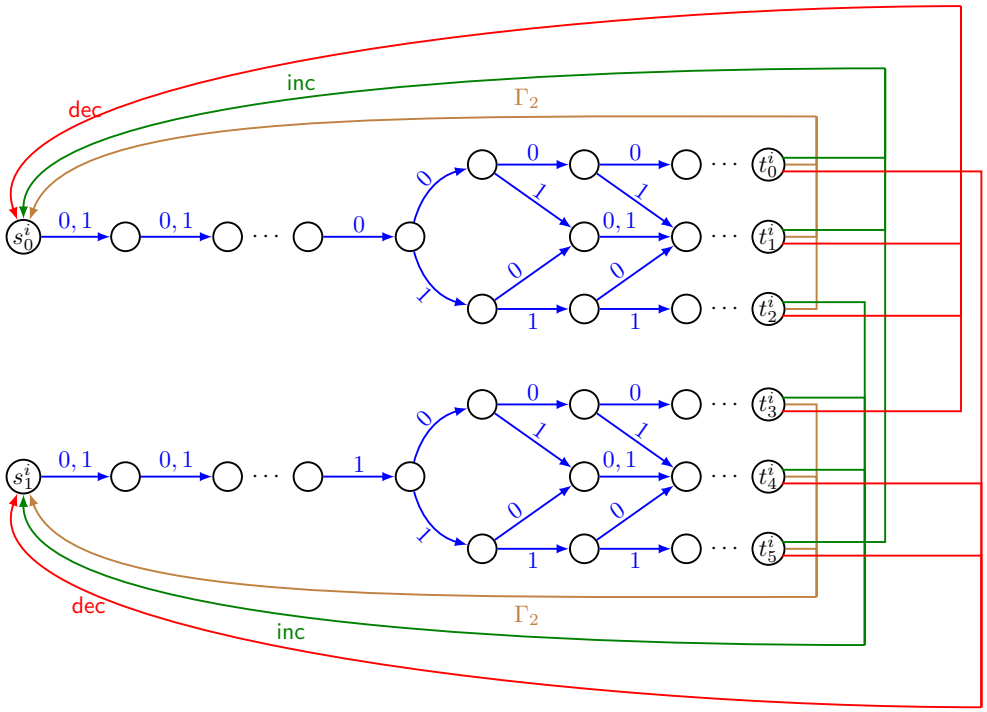

Figure 3 The automaton B_i. The transitions on $\{0, 1\}$ are depicted in blue. The transitions on Γ_2 (resp. inc, dec) are depicted in brown (resp. green, red).

bottom fork the ith bit read is always 1. Consider $n_{j-1}o_jn_j$ occurring in the above sequence. Let $n_{j-1} = b_1b_2 \dots b_\ell$ and let $n_j = b'_1b'_2 \dots b'_\ell$. If o_j is inc, the ith bit b_i is toggled ($b'_i \neq b_i$) iff $b_m = 1$ for all $m : m > i$. If o_j is dec, the ith bit b_i is toggled ($b'_i \neq b_i$) iff $b_m = 0$ for all $m : m > i$. If $o_j \in \Gamma_2$ the ith bit is never toggled. The initial states are s_0^i and s_1^i, and the accepting states are $t_0^i, \dots t_5^i$. Clearly $L_\ell = \bigcap_i L(B_i)$. ◁

However, for succinctly simulating the PDA given in Figure 2, we need the ℓ DFAs to faithfully reflect the stack height. For this we consider a slight modification of L_ℓ.

$$L'_\ell = \{n_0o_1n_1o_2n_2 \cdots o_kn_k \mid \begin{array}{l} k \geq 0, n_i \in (0+1)^\ell \text{ for all } i : 0 \leq i \leq k \\ o_i \in \{\mathsf{inc}, \mathsf{dec}\} \cup \Gamma_2 \text{ for all } i : 0 < i \leq k \\ \mathsf{val}(n_i) \equiv o_i(\mathsf{val}(n_{i-1})) \text{ for all } i : 0 < i \leq k \\ n_0 = 0^\ell = n_k, \text{ if } o_i = \mathsf{inc} \text{ then } n_{i-1} \neq 1^\ell \\ \text{if } o_i = \mathsf{dec} \text{ then } n_{i-1} \neq 0^\ell\} \end{array}$$

This ensures that the PDA starts and ends with an empty stack. Further inc after 1^ℓ and dec after 0^ℓ are forbidden. Otherwise, the value will not faithfully represent the stack height.

▷ Claim 14. There are ℓ DFAs $A_1, A_2, \dots A_\ell$, each with $\mathcal{O}(\ell)$ states such that $L'_\ell = \bigcap_i L(A_i)$.

Proof. The states of A_i are exactly those of B_i. For $i \geq 2$, the transitions of A_i is exactly the same as that of B_i. The transitions for A_1 is obtained by removing two transitions from that of B_1, namely the outgoing inc transition from t_5^1 and the outgoing dec transition from t_0^1. For all $i \geq 1$, the initial state of A_i is s_0^i and the final state is t_0^i. ◁

▷ Claim 15. There is a PDA A with 3 states and stack symbols $\{0, 1\}$ such that $L(A) \cap L'_\ell$ when projected to Γ_2 is exactly $(\Gamma_2)^{2^{2^\ell}}$.

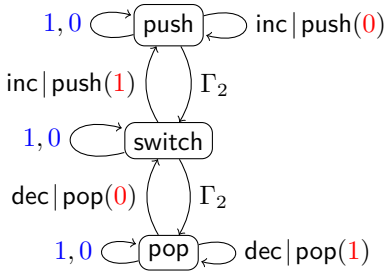

Figure 4 The PDA A. This PDA and the ℓ DFAs together faithfully encode the accepting runs of the PDA in Figure 2 with $n = 2^\ell - 1$. Thus they accept words with exactly 2^{2^ℓ} occurrences of Γ_2.

Proof. The PDA A is depicted in Figure 4. The three states represents the three modes of the PDA in Figure 2. The ℓ DFAs will guarantee that we start with the number 0^ℓ. Because of A_1, the PDA cannot take the inc transition from the state $\langle$push$\rangle$ immediately after the number 1^ℓ. It will have to read a Γ_2 symbol and move to the state $\langle$switch$\rangle$. The PDA will loop in this state once by reading the same number (1^ℓ) as mandated by the Γ_2 transitions of the DFAs. From this state, again inc is disabled by A_1, and hence the PDA will read another Γ_2 symbol and go to the state $\langle$pop$\rangle$. The PDA will read 1^ℓ staying in the state $\langle$pop$\rangle$ after which it can take a dec transition. ◁

4.4 Completing the reduction

Before giving the reduction, let us first define a PDA and the transducers that we use in the reduction. Let P_1 be the PDA in Claim 15 with $\Gamma_2 = \Sigma_1 \cup \{\#\}$. Let P_2 the PDA for $L(P_1) \cap (\Sigma_1^*\#^*)$. Let P_3 be the PDA in Claim 15 with $\Gamma_2 = \Sigma_2$. Let P_4 be the PDA for $L(P_2) \cdot L(P_3)$. Let $T_1, \ldots T_\ell$ be ℓ transducers. The input automaton of T_i is exactly the DFA A_i from Claim 14 with $\Gamma_2 = \Sigma_1 \cup \Sigma_2 \cup \{\#\}$. The output of the transducer T_i on every transition is ϵ.

Now we are ready to give the reduction. Let $\mathcal{P} = (\Sigma_1, \Sigma_2, f, g, \ell)$ be an input to a 2eBPCP problem. We describe how to obtain a string constraint C_P from this. We use the alphabet $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Gamma_1 \cup \{\#\}$, and the variable set $\mathcal{V} = \{x_0, x_1, x_2, x_f, x_g\}$. Next we define the membership constraints Mem. Let $\mathsf{Mem}(x_0) = A_\epsilon$ where A_ϵ is an NFA for $\{\epsilon\}$. We have $\mathsf{Mem}(x_1) = P_4, \mathsf{Mem}(x_2) = P_4$. Now we need to augment the language of x_f and x_g to also account for the letters from Γ_1, which can be achieved by adding Γ_1 self loops in all the states in the PDA for L_f and L_g respectively. Thus the language for x_f is $\mathfrak{W}(L_f, \Gamma_1^*)$, where $\mathfrak{W}(L_f, \Gamma_1^*) = \{w \mid w \in \mathsf{Shuffle}(u, v), u \in L_f, v \in \Gamma_1^*\}$. Similarly the language for x_g is $\mathfrak{W}(L_g, \Gamma_1^*)$. Let P_5 and P_6 be PDAs recognizing $\mathfrak{W}(L_f, \Gamma_1^*)$, and $\mathfrak{W}(L_g, \Gamma_1^*)$ respectively. We have $\mathsf{Mem}(x_f) = P_5$ and $\mathsf{Mem}(x_g) = P_6$. Let Rel be the following relational constraints: 1)$x_0 \preceq T_i(x_1)$, for all $i : 1 < i < \ell$, 2) $x_0 \preceq T_i(x_2)$, for all $i : 1 < i < \ell$, 3) $x_1 \preceq x_f$, 4) $x_f \preceq x_g$ and 5) $x_g \preceq x_2$. We have $2\ell + 3$ relational constraints. Further $\mathsf{multiplicity}(x_1) = \ell = \mathsf{multiplicity}(x_2)$ in our construction. Let $C_\mathcal{P} = (\Sigma, \mathcal{V}, \mathsf{Mem}, \mathsf{Rel})$.

▷ Claim 16. The string constraint $C_\mathcal{P}$ is satisfiable if and only if the 2eBPCP instance $\mathcal{P}$ has a solution.

Notice that we construct $C_{\mathcal{P}}$ from $\mathcal{P}$ in polynomial time, and it is acyclic. Hence, it follows that the satisfiability checking of acyclic string constraints is 2NExptime-hard, proving the lower bound of Theorem 5.

5 Satisfiability is in 2NEXPTIME

We will show that if an acyclic string constraint is satisfiable, then there is a satisfying assignment of double exponential size.

▶ **Theorem 17** (Small model property). *Let C be a satisfiable acyclic string constraint. Then C has a satisfying assignment σ such that $\mathsf{len}(\sigma(x)) \leq B$ where B is*

$$2^{2^{\mathcal{O}(m \log t + \log p + \log k)}}$$

where

- $m = \max_{x \in \mathcal{V}} \mathit{multiplicity}(x)$*, the maximum multiplicity of any variable,*
- $t = \max_{T \in TRSET(C)} \mathit{state\text{-}size}(T)$*, the maximum number of states of any transducer,*
- $p = \max_{P \in AUTSET(C)} \mathit{state\text{-}size}(P)$*, the maximum number of states (and stack symbols) of any automaton.*
- $k = |\mathcal{V}|$*, the number of variables.*

Our aim, towards a 2NEXPTIME procedure, is to non-deterministically guess an assignment of size at most double exponential, and check that it satisfies the Conditions 1 and 2 (see Definition 2). However, Condition 2 uses more existentially quantified variables, and it is not evident that verifying Condition 2 can be done within the complexity limits. Towards this, we define an extended assignment which considers the values given to these existentially quantified variables as well, and show that every word used in this extended assignment is of length at most B. We define the extended assignment and the related notions and notations, and state the small model property for the extended assignment.

Recall that $\mathsf{multiplicity}(x)$ denotes the number of times a variable occurs on the RHS of the constraints. In each such occurrence, the variable occurs in a pair along with a transducer (of the form (x, T)), which belongs to the RHS of a constraint from Rel of the form (y, Y). Let us fix some enumeration of these occurrences, and define the respective transducer and constraint of the ith occurrence of x by T_i^x and $\mathsf{constraint}(x, i)$. Now, as per Condition 2, there are words (output words of the respective transducers), that witness the transduction. For every $x \in \mathcal{V}$ and $i \in [\mathsf{multiplicity}(x)]$, let o_i^x be a new variable. This variable is intended to take as value a witness word for the output of the transducer T_i^x on the word provided by x, so that the constraint $\mathsf{constraint}(x, i)$ is satisfied. Let $\widehat{\mathcal{V}}(C)$, or simply $\widehat{\mathcal{V}}$ when C is clear from the context, contain the output variables in addition to the original variables. That is, $\widehat{\mathcal{V}} = O \cup \mathcal{V}$, where $O = \{o_i^x \mid x \in \mathcal{V}, i \in [\mathsf{multiplicity}(x)]\}$. An extended assignment $\widehat{\sigma} : \widehat{\mathcal{V}} \to \Sigma^*$ *satisfies* a string constraint C if

- **E1** $\widehat{\sigma}(x) \in \mathsf{Mem}(x)$ for all $x \in \mathcal{V}$
- **E2** $(\widehat{\sigma}(x), \widehat{\sigma}(o_i^x)) \in R(T_i^x)$, for all $x \in \mathcal{V}$, $i \in [\mathsf{multiplicity}(x)]$
- **E3** For every $(y, Y) \in \mathsf{Rel}$, we have $\widehat{\sigma}(y) \preceq \mathsf{Shuffle}(\widehat{\sigma}(Y))$ where $\widehat{\sigma}(Y)$ is an overloaded notation for the multiset

$$\{\!\{\widehat{\sigma}(o_i^x) \mid x \in \mathcal{V}, i \in [\mathsf{multiplicity}(x)], \mathsf{constraint}(x, i) = (y, Y)\}\!\}. \tag{1}$$

We will actually prove the small model property for the extended assignments.

▶ **Lemma 18.** *Let C be a satisfiable acyclic string constraint. Then C has a satisfying extended assignment $\widehat{\sigma}$ such that $\mathsf{len}(\widehat{\sigma}(x)) \leq B$ for all $x \in \mathcal{V}(C)$, and $\mathsf{len}(\widehat{\sigma}(y)) \leq 2ctB$ for all $y \in O$. The parameters B and t are as defined in Theorem 17, and $c = \max\{\mathsf{len}(\mathsf{out}(tr)) \mid tr \text{ is a transition of } T, \text{ and } T \in \textsc{trset}(C)\}$.*

With this, our non-deterministic procedure guesses an extended assignment and verifies that it satisfies the conditions 1, 2 and 3. In fact, checking whether a given word w is a subword of some word in $\mathsf{Shuffle}(W)$ where W is a finite mutliset of words is NP-complete [9, 24]. It remains to prove Lemma 18.

Proof of Lemma 18. Consider an acyclic string constraint C. Recall that we write $x < y$ if $(x, Y) \in \mathsf{Rel}$ with $(y, T) \in Y$ for some $T \in \textsc{trset}(C)$. Consider a topological sorting of the variables respecting the relation $<$, say $x_1, x_2, \ldots, x_k$. Note that x_1 does not appear in the RHS of any subword order constraint (in other words, $\mathsf{multiplicity}(x_1) = 0$). If (x_i, T) appears on the RHS of any constraint for some T, then the LHS of that constraint is x_j for some $j < i$.

Suppose C is satisfiable. Let $\widehat{\sigma}_0$ be a satisfying extended assignment. In order to get the $\widehat{\sigma}$ as per Lemma 18, we will construct a sequence of k extended assignments, each progressively modifying the previous one until we reach our goal. That is, we will construct the sequence, $\widehat{\sigma}_0, \widehat{\sigma}_1, \ldots, \widehat{\sigma}_k = \widehat{\sigma}$ such that for each $i \in [k]$

I1 $\widehat{\sigma}_i$ is satisfiable. That is, 1) membership constraints are satisfied (Condition 1), 2) transductions are accepted by the transducers (Condition 2), and 3) the relational constraints are satisfied (Condition 3).

I2 for all $j : j \leq i$, $\mathsf{len}(\widehat{\sigma}_i(x_j)) \leq B_j$ and for each $\ell \in \mathsf{multiplicity}(x_j)$, $\mathsf{len}(\widehat{\sigma}_i(o_\ell^{x_j})) \leq 2ctB_j$. We define B_n as follows. $B_1 = 2^{p^3}$, and for $n > 1$, $B_n = 2m \cdot t^{2m} \cdot p^3 \cdot B_{n-1} \cdot 2^{p^3 t^{2m}}$.

Note that, B_n increases with n and B_k is at most B.

Base cases. We consider $\widehat{\sigma}_0$ and $\widehat{\sigma}_1$ as base cases. For $\widehat{\sigma}_0$, it is given to be satisfiable, and Condition 2 above holds vacuously.

Towards constructing $\widehat{\sigma}_1$, consider the variable x_1, and a context-free grammar G_1 for $\mathsf{Mem}(x_1)$ in Chomsky Normal Form with at most p^3 non-terminals [15]. Note that G_1 can be constructed in polynomial time. Since $\widehat{\sigma}_0$ is satisfying, the word $w_1 = \widehat{\sigma}_0(x_1)$ has a valid parse tree in G_1. If a non terminal repeats in any leaf to root path in this tree, say at node n_1 and node n_2 with n_2 an ancestor of n_1, then we can shrink the parse tree (pump down) by replacing the subtree rooted at n_2 by the subtree rooted at n_1 to get a smaller parse tree of a smaller word in the language. Furthermore, this smaller word will be a subword of w_1. Consider a shrinking of the parse tree of w_1 which cannot be shrunk any further. This tree has size at most 2^{p^3}, and hence its yield $\widehat{w_1}$ satisfies Condition 2. Setting x_1 to $\widehat{w_1}$ will also satisfy Condition 1. Further, note that there are no output variables corresponding to x_1. Hence we get $\widehat{\sigma}_1$: $\widehat{\sigma}_1(x) = \begin{cases} \widehat{w_1} & \text{if } x = x_1 \\ \widehat{\sigma}_0(x) & \text{otherwise.} \end{cases}$

Inductive Step. Now, for the inductive case, assume we have constructed $\widehat{\sigma}_{i-1}$. We will describe how to obtain $\widehat{\sigma}_i$. Let G_i be the context-free grammar in Chomsky Normal Form for $\mathsf{Mem}(x_i)$ with p^3 non-terminals. We will basically do a "conservative" pumping down of $w_i = \sigma_{i-1}(x_i)$, which ensures that the constraints are still satisfied, which we explain below.

Challenges. In order to bound the length of $\widehat{\sigma}_i(x_i)$ we may consider subwords $w'_i \preceq w_i$, so that the constraints in which x_i appear on the left continue to be satisfied. In addition, such a subword w'_i must not only satisfy the membership constraint ($w'_i \in \mathsf{Mem}(x_i)$) but also admit the specified transductions – that is, for all $\ell \in [\mathsf{multiplicity}(x_i)]$, we must have $(w'_i, u'_\ell) \in R(T_\ell^{x_i})$ for some u'_ℓ. Furthermore, a mere existence of such a u'_ℓ is not sufficient – consider the constraint $(x_j, Y) = \mathsf{constraint}(x_i, \ell)$ and let $\widehat{\sigma}_{i-1}(x_j) = w_j$ and $\widehat{\sigma}_{i-1}(Y) = U$ with $\widehat{\sigma}_{i-1}(o_\ell^{x_i}) = u_\ell$ (recall, $\widehat{\sigma}(Y)$ is defined in Equation 1)). Since $\widehat{\sigma}_{i-1}$ is satisfying we know that $w_j \preceq \mathsf{Shuffle}(U)$. However, it need not be the case that $w_j \preceq \mathsf{Shuffle}(U \setminus \{u_\ell\} \cup \{u'_\ell\})$. Hence we need to find a suitable u'_ℓ such that $w_j \preceq U \setminus \{u_\ell\} \cup \{u'_\ell\}$. One way to ensure this, is by insisting that u'_ℓ provides the same "witnessing subword" that u_ℓ provided. We formalise this notion of "witnessing subword" below.

We give two equivalent definitions for $w \preceq \mathsf{Shuffle}(U)$.

▷ Claim 19. Let w be a word and $U = \{\!\{u'_1, u'_2 \dots u'_n\}\!\}$ be a multiset of words. The following statements are equivalent.

1. There exists w': 1) $w' \in \mathsf{Shuffle}(U)$ and 2) $w \preceq w'$.
2. There exist $u'_1, u'_2 \dots u'_n$: 1) $w \in \mathsf{Shuffle}(\{\!\{u'_1, u'_2 \dots u'_n\}\!\})$ and 2) $u'_i \preceq u_i$.

We refer to $v_1, \dots v_n$ as the *witnessing subwords* of $u_1, \dots u_n$ for $w \preceq \mathsf{Shuffle}(U)$. Thus, for the inductive case, we need to find good w'_i, $u'_1, \dots u'_{\mathsf{multiplicity}(x_i)}$ such that

1. $|w'_i| \leq B_i$
2. for each ℓ, $|u'_\ell| \leq 2 \cdot t \cdot B_i$,
3. $w'_i \in \mathsf{Mem}(x_i)$,
4. $(w'_i, u'_\ell) \in R(T_\ell^{x_i})$ for each ℓ, and
5. $v_\ell \preceq u'_\ell$ where v_ℓ is a witnessing subword of $u_\ell = \widehat{\sigma}_{i-1}(o_\ell^{x_i})$ for the relation $\widehat{\sigma}_{i-1}(y) \preceq \mathsf{Shuffle}(\widehat{\sigma}_{i-1}(Y))$ letting $(y, Y) = \mathsf{constraint}(x_i, \ell)$. (Note that v_ℓ exists because $\widehat{\sigma}_{i-1}$ is satisfiable by induction hypothesis.)

Block decomposition. Towards the above goal, let us consider w_i and $\rho_1 \dots \rho_m$ where $m = \mathsf{multiplicity}(x_i)$ and for $\ell : 1 \leq \ell \leq m$, ρ_ℓ is an accepting run of the transducer $T_\ell^{x_i}$ on the input w_i producing $u_\ell = \widehat{\sigma}_{i-1}(o_\ell^{x_i})$. This is depicted in Figure 5. We factorize each ρ_ℓ into blocks. The number of blocks is exactly $n+1$, where $n = |w_i|$. For $j > 0$, the jth block contains the transition on the jth letter of w_i, followed by all the trailing ϵ-input transitions. The very first block (block 0), contains the leading ϵ-input transitions if present. Now, we decompose the output of ρ_ℓ according to the blocks. That is, $u_\ell = u_\ell^0 u_\ell^1 \cdot u_\ell^2 \dots u_\ell^n$. Next we want to identify the subword of w_i (and subruns of ρ_ℓ) that needs to be preserved.

Identifying crucial blocks and positions. Consider v_ℓ, the witnessing subword of u_ℓ for $\widehat{\sigma}_{i-1}(y) \preceq \mathsf{Shuffle}(\widehat{\sigma}_{i-1}(Y))$ where $(y, Y) = \mathsf{constraint}(x_i, \ell)$. Fix an embedding of v_ℓ in u_ℓ. If this embedding is incident on the factor u_ℓ^j for $j > 0$, we will mark the the jth block as well as the jth letter of w_i as *crucial*. Since $|\widehat{\sigma}_{i-1}(y)| \leq B_{i-1}$ (by induction hypothesis), the number of crucial blocks in ρ_ℓ is at most B_{i-1}. Hence the number of crucial positions in w_i is at most $m \times B_{i-1}$ where $m = \mathsf{multiplicity}(x_i)$. Notice that if we shrink w_i to a subword that 1) preserves the crucial positions, 2) preserves membership in $\mathsf{Mem}(x_i)$ and 3) yields subruns of ρ_ℓ that preserves the crucial blocks, then the satisfiability would be preserved. Our next aim is to obtain such a shrinking, which is sufficiently small to also satisfy the length requirements.

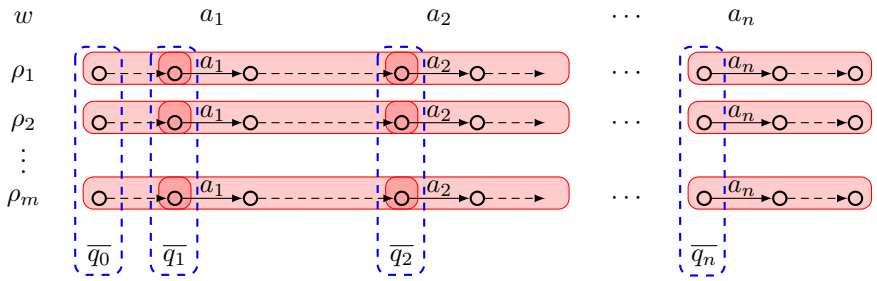

Figure 5 The figure describes the block decomposition of the runs of transducers. The very first block includes the sequence of ϵ-transitions (if any) before the first letter of the input is read. Every other block includes a transition on input letter followed by a sequence of ϵ transitions. Blocks here are represented as overlapping rectangles coloured in red. The transition on input is represented as a solid arrow and a sequence of ϵ transitions is represented as a dashed arrows.

Annotated parse trees. Consider a grammar G_i for $\mathsf{Mem}(x_i)$ in Chomsky Normal Form and a parse tree of w_i in G_i. Annotate the nodes of this parse-tree by pairs of m-tuple of states. The m-tuple of states $\overline{q_j}$ correspond to the states of the transducers at the boundary between $(j-1)$th block and jth block. A node is annotated with $\langle \overline{q_{j-1}}, \overline{q_{j'}} \rangle$ if the yield of the subtree rooted at that node generates the factor of w_i from jth letter to j'th letter (for some $j' \geq j$). Notice that some of the leaves are marked as crucial. We will mark an internal node as crucial if it is the least common ancestor of two crucial nodes.

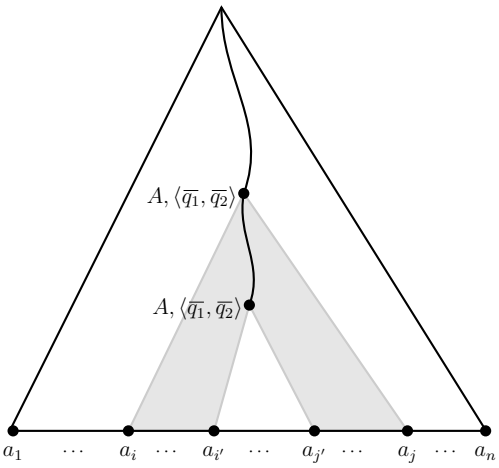

Figure 6 The figure illustrates the annotations of a nodes and pumping down in a parse tree.

Shrinking the parse tree. Now, if there are two nodes n_1 and n_2 in this tree such that 1) both have the same annotated non-terminal, 2) n_1 is an ancestor of n_2, 3) there are no crucial nodes in the path from n_1 to n_2, then we replace the subtree rooted at n_1 with the subtree rooted at n_2 (pumping down). This is illustrated in Figure 6. We repeat this until no more pumping down is possible. The yield of this shrunk parse tree is the required word w_i'. Let us analyse the size of w_i'. Any path without a crucial node is of length at most p^3t^{2m}. Hence the *skeleton* of the parse tree that contains all the crucial nodes and the paths from them to the root will be of size at most $2n_C \times p^3t^{2m}$, where n_C is the number of crucial positions of w_i.

Any sub tree rooted at any of the nodes of the skeleton is of size at most $2^{p^3t^{2m}}$. Hence the total size of the tree is at most $2n_C \times p^3t^{2m} \times 2^{p^3t^{2m}}$. Since $n_C \leq mB_{i-1}$, we have $\mathsf{len}(w'_i) \leq B_i$.

Shrinking the transducer runs. Note that, since the shrinking preserves the annotations, shrinking the ρ_ℓs appropriately gives us an accepting subrun ρ'_ℓ that preserves the crucial blocks. The number of blocks in ρ'_ℓ is at most B_i. Now, we need to shrink the size of each block as well, in order to satisfy $\mathsf{len}(u'_\ell) \leq 2ctB_i$. For this, consider the witnessing subword of u_ℓ. Note that it is still embedded in $\mathsf{out}(\rho'_\ell)$. If this embedding is incident on the output of a transition we will mark this transition as crucial. Further all the transitions that read a letter from w'_i are also crucial. Note that the number of crucial transitions is at most $B_i + B_{i-1}$. Now, let us shrink the run ρ'_ℓ without losing crucial transitions to get ρ''_ℓ. The number of transitions in ρ''_ℓ is at most $t \times (B_i + B_{i-1} + 1)$ where t is the number of states. Let $u'_\ell = \mathsf{out}(\rho''_\ell)$. Then it is easy to see that $\mathsf{len}(u'_\ell) \leq 2ctB_i$.

Finally, we can give the required $\widehat{\sigma}_i$. Below, x_j comes from $\mathcal{V}$ and o^x_ℓ comes from O.

$$\widehat{\sigma}_i(x_j) = \begin{cases} w'_i & \text{if } j = i \\ \widehat{\sigma}_{i-1}(x_j) & \text{if } j \neq i \end{cases} \qquad \widehat{\sigma}_i(o^x_\ell) = \begin{cases} u'_\ell & \text{if } x = x_i \\ \widehat{\sigma}_{i-1}(o^x_\ell) & \text{otherwise} \end{cases}$$

This establishes the proof of Lemma 18. ◀

A proof of NExptime-completeness for the setting where only regular membership is allowed (proof of Theorem 6) can be found in the full version of this paper [10].

6 Discussions

6.1 Application: Regular abstractions and DFA sizes

For any language L, we define its Parikh image closure ($\Pi(L)$), downward closure ($L\downarrow$) and upward closure ($L\uparrow$) as follows. Let $\Sigma = \{a_1, \cdots a_n\}$. For $w \in \Sigma^*$, we let $p(w) = \langle \mathsf{len}(w)_{a_1}, \cdots, \mathsf{len}(w)_{a_n} \rangle$ denote the Parikh image of w, that is, it counts the occurrences of each letter from Σ. Here, by $\mathsf{len}(w)_a$, we mean the number of times a occurs in w.

$$\Pi(L) = \{v \in \Sigma^* \mid \exists w \in L, p(v) = p(w)\} \qquad L\downarrow = \{v \in \Sigma^* \mid \exists w \in L, v \preceq w\}$$
$$L\uparrow = \{v \in \Sigma^* \mid \exists w \in L, w \preceq v\}$$

Efficient computability of these regular abstractions of languages of infinite state systems is a relevant question for verification and automata theory [14, 12, 46, 13]. It is interesting to see if small automata representing these abstractions can be computed for succinctly given infinite state systems.

We address here the case where a large pushdown system is presented as a small pushdown system and a certain number of finite state automata (referred to as the smaller components). Here the language of the large pushdown system is same as the intersection of the languages of the smaller components. We argue that the lower bound on the size of the regular abstraction holds even when pushdown system is presented succinctly.

For any $n \in \mathbb{N}$, let $L(n)$ be the language over $\Sigma = \{0, 1, a\}$ that accepts the word $w = n_0 o_1 n_1 o_2 n_2 \cdots n_k$, where $n_0 = 0^n$, $n_k = 1^n$, $o_1, o_2, \cdots o_k = \mathsf{inc}$, then $\mathsf{len}(w)_{\mathsf{inc}} = 2^n$. From Section 4 we know that we can construct n DFAs $B_1, B_2, \ldots B_n$ such that $\bigcap_i L(B_i) = \{w\}$. Since any DFA recognising the closure of this language requires at least 2^n states, we have the following claim.

▷ Claim 20. Given n regular languages as n finite state automata, let L be the language obtained by intersecting the languages of these automata. Then, the minimal DFA for $\Pi(L)$, $L\downarrow$ and $L\uparrow$ can be of exponential size.

Consider the language given in Claim 15 i.e. $L(A) \cap L'_\ell$, since it can recognize words with exactly 2^{2^n} many symbols from Γ_2, we have the following claim.

▷ Claim 21. Given n regular languages as n finite state automata and a pushdown system, let L be the language got by intersecting the languages of these automata. Then, the minimal DFA for $\Pi(L)$, $L\downarrow$ and $L\uparrow$ can be of double exponential size.

6.2 Concatenation instead of Shuffle

In [9] it is shown that shuffle can express concatenation with a polynomial blow-up, but preserving acyclicity. It is interesting to see if the hardness holds in the presence of concatenation alone. Already, in the setting of [9] (no transductions), if acyclicity is not imposed, it is not known whether satisfiability of the regular string constraints is decidable if only concatenation is allowed instead of shuffle. In our setting (in the presence of tranducers), it turns out that satisfiability is undecidable. We show this by modifying the reduction in [9].

Let $\mathcal{P} = (\Sigma_1, \Sigma_2, f, g)$ be a given PCP instance, let $\mathcal{L}_u = (\{i \cdot f(i) \mid i \in \Sigma_1\})^*$, $\mathcal{L}_v = (\{i \cdot g(i) \mid i \in \Sigma_1\})^*$, $\mathcal{L}_i = \Sigma_1^+$ and $\mathcal{L}_s = \Sigma_2^*$. Notice that all these four languages are regular, let $\mathcal{A}_u, \mathcal{A}_v, \mathcal{A}_i$ and $\mathcal{A}_s$ be their corresponding NFA. Then the required string constraint is $(\Sigma, \mathcal{V}, \mathsf{Mem}, \mathsf{Rel})$, where $\mathcal{V} = \{u, v, i, s\}$, for any $x \in \mathcal{V}$, $\mathsf{Mem}(x) = \mathcal{A}_x$. Let $T_\Sigma = \{(a_1 \dots a_n, \Sigma^* a_1 \Sigma^* a_1 \dots \Sigma^* a_n \Sigma^*)\}$ be the transduction that arbitrarily inserts words from Σ. The set Rel is given by

$$
\begin{array}{llll}
i \preceq u & s \preceq u & u \preceq T_{\Sigma_1}(s) & u \preceq T_{\Sigma_2}(t) \\
i \preceq v & s \preceq v & v \preceq T_{\Sigma_1}(s) & v \preceq T_{\Sigma_2}(t)
\end{array}
$$

In the case of acyclic string constraints, one may wonder if the lower bounds hold in a setting where only concatenation is allowed instead of shuffle. This was not discussed in [9]. Infact, in our case, the lower bound holds even when only concatenation is allowed. Notice that, in our 2NExptime-hard reduction, all relational constraints have only one variable in the RHS. Hence, the 2NExptime-hard holds for acyclic pushdown string constraints with transducers, even when shuffle (or even concatenation) is disallowed.

In fact, it is not known whether the lower bounds in [9] hold for the variant with only concatenation instead of shuffle. It is also open whether satisfiability is decidable for the *regular* string constraints without acyclicity restriction when only concatenation is allowed.

7 Conclusions

In this paper, we considered string constraints in the presence of sub-word relation, shuffle operator (which subsumes concatenation [9]) and transducers. We studied this problem for two different kinds of membership constraints, namely regular and context free. We showed that in the case when only regular membership constraints are involved, the problem is NExptime-complete. Whereas, when context-free membership constraints are involved, the problem is 2NExptime-complete. Towards the hardness proof, we showed how to count exactly 2^n using n finite state automata each of size $\mathcal{O}(n)$. As a consequence of this result, we also obtained a lower bound for any regular representation of the upward closure, downward closure and Parikh image closure of the intersection of the language of n finite state automata.

Similarly, we showed that we can count exactly 2^{2^n} using a pushdown automaton and n finite state automata, each of size $\mathcal{O}(n)$. With this, we also obtained a lower bound for any regular representation of the upward closure, downward closure and Parikh image closure of the intersection language of a pushdown and n finite state automata.

References

1. A. Parosh Abdulla, F. Mohamed Atig, Yu-Fang Chen, Diep Phi Bui, Lukáš Holík, Ahmed Rezine, and Philipp Rummer. Trau : SMT solver for string constraints. In *Proceedings of the 18th Conference on Formal Methods in Computer-Aided Design*, pages 165–169. FMCAD Inc., 2019. doi:10.23919/FMCAD.2018.8602997.
2. Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Bui Phi Diep, Lukás Holík, Denghang Hu, Wei-Lun Tsai, Zhilin Wu, and Di-De Yen. Solving not-substring constraint with flat abstraction. In Hakjoo Oh, editor, *Programming Languages and Systems - 19th Asian Symposium, APLAS 2021, Chicago, IL, USA, October 17-18, 2021, Proceedings*, Lecture Notes in Computer Science. Springer, 2021. doi:10.1007/978-3-030-89051-3_17.
3. Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Bui Phi Diep, Lukáš Holík, Ahmed Rezine, and Philipp Rümmer. Flatten and conquer: A framework for efficient analysis of string constraints. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2017, pages 602–617, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3062341.3062384.
4. Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Bui Phi Diep, Lukáš Holík, Ahmed Rezine, and Philipp Rümmer. Flatten and conquer: A framework for efficient analysis of string constraints. *SIGPLAN Not.*, 52(6), 2017. doi:10.1145/3140587.3062384.
5. Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Lukás Holík, Ahmed Rezine, Philipp Rümmer, and Jari Stenman. String constraints for verification. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 150–166. Springer, 2014. doi:10.1007/978-3-319-08867-9_10.
6. Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Lukás Holík, Ahmed Rezine, Philipp Rümmer, and Jari Stenman. Norn: An SMT solver for string constraints. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, volume 9206 of *Lecture Notes in Computer Science*, pages 462–469. Springer, 2015. doi:10.1007/978-3-319-21690-4_29.
7. Parosh Aziz Abdulla, Mohamed Faouzi Atig, Vrunda Dave, and Shankara Narayanan Krishna. On the separability problem of string constraints. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CONCUR.2020.16.
8. Parosh Aziz Abdulla, Mohamed Faouzi Atig, Bui Phi Diep, Lukás Holík, and Petr Janku. Chain-free string constraints. In Yu-Fang Chen, Chih-Hong Cheng, and Javier Esparza, editors, *Automated Technology for Verification and Analysis - 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28-31, 2019, Proceedings*, volume 11781 of *Lecture Notes in Computer Science*. Springer, 2019. doi:10.1007/978-3-030-31784-3_16.
9. C. Aiswarya, Soumodev Mal, and Prakash Saivasan. On the satisfiability of context-free string constraints with subword-ordering. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*. ACM, 2022. doi:10.1145/3531130.3533329.
10. C Aiswarya, Soumodev Mal, and Prakash Saivasan. Satisfiability of context-free string constraints with subword-ordering and transducers, 2024. arXiv:2401.07996.

11 Roberto Amadini. A survey on string constraint solving. *ACM Comput. Surv.*, 55(2):16:1–16:38, 2023. doi:10.1145/3484198.

12 Mohamed Faouzi Atig, Ahmed Bouajjani, K. Narayan Kumar, and Prakash Saivasan. On bounded reachability analysis of shared memory systems. In Venkatesh Raman and S. P. Suresh, editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, volume 29 of *LIPIcs*, pages 611–623. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPICS.FSTTCS.2014.611.

13 Mohamed Faouzi Atig, Ahmed Bouajjani, and Tayssir Touili. On the reachability analysis of acyclic networks of pushdown systems. In Franck van Breugel and Marsha Chechik, editors, *Concurrency Theory, 19th International Conference, CONCUR 2008, Toronto, Canada, August 19-22, 2008. Proceedings*, volume 5201 of *Lecture Notes in Computer Science*. Springer, 2008. doi:10.1007/978-3-540-85361-9_29.

14 Mohamed Faouzi Atig, Dmitry Chistikov, Piotr Hofman, K. Narayan Kumar, Prakash Saivasan, and Georg Zetzsche. The complexity of regular abstractions of one-counter languages. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 207–216. ACM, 2016. doi:10.1145/2933575.2934561.

15 Mohamed Faouzi Atig, K. Narayan Kumar, and Prakash Saivasan. Adjacent ordered multi-pushdown systems. In Marie-Pierre Béal and Olivier Carton, editors, *Developments in Language Theory - 17th International Conference, DLT 2013, Marne-la-Vallée, France, June 18-21, 2013. Proceedings*, volume 7907 of *Lecture Notes in Computer Science*, pages 58–69. Springer, 2013. doi:10.1007/978-3-642-38771-5_7.

16 Murphy Berzish, Joel D. Day, Vijay Ganesh, Mitja Kulczynski, Florin Manea, Federico Mora, and Dirk Nowotka. Towards more efficient methods for solving regular-expression heavy string constraints. *Theor. Comput. Sci.*, 2023. doi:10.1016/j.tcs.2022.12.009.

17 Murphy Berzish, Vijay Ganesh, and Yunhui Zheng. Z3str3: A string solver with theory-aware heuristics. In *Proceedings of the 17th Conference on Formal Methods in Computer-Aided Design*, FMCAD '17, Austin, Texas, 2017. FMCAD Inc. doi:10.23919/FMCAD.2017.8102241.

18 J. Richard Büchi and Steven Senger. Definability in the existential theory of concatenation and undecidable extensions of this theory. *Math. Log. Q.*, 34:337–342, 1988.

19 Taolue Chen, Yan Chen, Matthew Hague, Anthony W. Lin, and Zhilin Wu. What is decidable about string constraints with the replaceall function. *Proc. ACM Program. Lang.*, 2(POPL):3:1–3:29, 2018. doi:10.1145/3158091.

20 Taolue Chen, Alejandro Flores-Lamas, Matthew Hague, Zhilei Han, Denghang Hu, Shuanglong Kan, Anthony W. Lin, Philipp Rümmer, and Zhilin Wu. Solving string constraints with regex-dependent functions through transducers with priorities and variables. *Proc. ACM Program. Lang.*, 6(POPL), 2022. doi:10.1145/3498707.

21 Taolue Chen, Alejandro Flores-Lamas, Matthew Hague, Zhilei Han, Denghang Hu, Shuanglong Kan, Anthony Widjaja Lin, Philipp Rümmer, and Zhilin Wu. Solving string constraints with regex-dependent functions through transducers with priorities and variables. *CoRR*, abs/2111.04298, 2021.

22 Taolue Chen, Matthew Hague, Jinlong He, Denghang Hu, Anthony Widjaja Lin, Philipp Rümmer, and Zhilin Wu. A decision procedure for path feasibility of string manipulating programs with integer data type. In Dang Van Hung and Oleg Sokolsky, editors, *Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings*, volume 12302 of *Lecture Notes in Computer Science*, pages 325–342. Springer, 2020. doi:10.1007/978-3-030-59152-6_18.

23 Taolue Chen, Matthew Hague, Anthony W. Lin, Philipp Rümmer, and Zhilin Wu. Decision procedures for path feasibility of string-manipulating programs with complex operations. *Proc. ACM Program. Lang.*, 3(POPL), 2019. doi:10.1145/3290362.

24 Peter Chini, Jonathan Kolberg, Andreas Krebs, Roland Meyer, and Prakash Saivasan. On the complexity of bounded context switching. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.27.

25 John Corcoran, William Frank, and Michael Maloney. String theory. *J. Symb. Log.*, 39(4):625–637, 1974. doi:10.2307/2272846.

26 Joel D. Day. Word equations in the context of string solving. In Volker Diekert and Mikhail V. Volkov, editors, *Developments in Language Theory - 26th International Conference, DLT 2022, Tampa, FL, USA, May 9-13, 2022, Proceedings*, Lecture Notes in Computer Science. Springer, 2022. doi:10.1007/978-3-031-05578-2_2.

27 Joel D. Day, Vijay Ganesh, Nathan Grewal, and Florin Manea. On the expressive power of string constraints. *Proc. ACM Program. Lang.*, 7(POPL):278–308, 2023. doi:10.1145/3571203.

28 Joel D. Day, Vijay Ganesh, Paul He, Florin Manea, and Dirk Nowotka. The satisfiability of word equations: Decidable and undecidable theories. In Igor Potapov and Pierre-Alain Reynier, editors, *Reachability Problems*, pages 15–29, Cham, 2018. Springer International Publishing.

29 Diego Figueira, Artur Jez, and Anthony W. Lin. Data path queries over embedded graph databases. In Leonid Libkin and Pablo Barceló, editors, *PODS '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, pages 189–201. ACM, 2022. doi:10.1145/3517804.3524159.

30 Vijay Ganesh, Mia Minnes, Armando Solar-Lezama, and Martin C. Rinard. Word equations with length constraints: What's decidable? In Armin Biere, Amir Nahir, and Tanja E. J. Vos, editors, *Hardware and Software: Verification and Testing - 8th International Haifa Verification Conference, HVC 2012, Haifa, Israel, November 6-8, 2012. Revised Selected Papers*, volume 7857 of *Lecture Notes in Computer Science*. Springer, 2012. doi:10.1007/978-3-642-39611-3_21.

31 Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990.

32 Matthew Hague, Anthony Widjaja Lin, and C.-H. Luke Ong. Detecting redundant CSS rules in HTML5 applications: a tree rewriting approach. In Jonathan Aldrich and Patrick Eugster, editors, *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2015, part of SPLASH 2015, Pittsburgh, PA, USA, October 25-30, 2015*, pages 1–19. ACM, 2015. doi:10.1145/2814270.2814288.

33 Hermes Hans. Semiotik. eine theorie der zeichengestalten als grundlage für untersuchungen von formalisierten sprachen. *Journal of Philosophy*, 36(13):356–357, 1939. doi:10.2307/2017267.

34 Lukás Holík, Petr Janku, Anthony W. Lin, Philipp Rümmer, and Tomás Vojnar. String constraints with concatenation and transducers solved efficiently. *Proc. ACM Program. Lang.*, 2(POPL), 2018. doi:10.1145/3158092.

35 Shuanglong Kan, Anthony Widjaja Lin, Philipp Rümmer, and Micha Schrader. Certistr: a certified string solver. In Andrei Popescu and Steve Zdancewic, editors, *CPP '22: 11th ACM SIGPLAN International Conference on Certified Programs and Proofs, Philadelphia, PA, USA, January 17 - 18, 2022*. ACM, 2022. doi:10.1145/3497775.3503691.

36 Adam Kiezun, Vijay Ganesh, Shay Artzi, Philip J. Guo, Pieter Hooimeijer, and Michael D. Ernst. Hampi: A solver for word equations over strings, regular expressions, and context-free grammars. *ACM Trans. Softw. Eng. Methodol.*, 21(4), February 2013.

37 Adam Kiezun, Philip J. Guo, Pieter Hooimeijer, Michael D. Ernst, and Vijay Ganesh. Theory and practice of string solvers (invited talk abstract). In Dongmei Zhang and Anders Møller, editors, *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2019, Beijing, China, July 15-19, 2019*. ACM, 2019. doi:10.1145/3293882.3338993.

38 Tianyi Liang, Andrew Reynolds, Nestan Tsiskaridze, Cesare Tinelli, Clark Barrett, and Morgan Deters. An efficient SMT solver for string constraints. *Form. Methods Syst. Des.*, 48(3), 2016. doi:10.1007/s10703-016-0247-6.

39 Anthony Widjaja Lin and Pablo Barceló. String solving with word equations and transducers: towards a logic for analysing mutation XSS. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 123–136. ACM, 2016. doi:10.1145/2837614.2837641.

40 G S Makanin. The problem of solvability of equations in a free smigroup. *Mathematics of the USSR-Sbornik*, 32(2), 1977. doi:10.1070/SM1977v032n02ABEH002376.

41 Yu. V. Matiyasevich. A connection between systems of words-and-lengths equations and Hilbert's tenth problem. *Studies in constructive mathematics and mathematical logic. Part II,Zap. Nauchn. Sem. LOMI*, 8, 1968.

42 Wojciech Plandowski. Satisfiability of word equations with constants is in PSPACE. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 495–500. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814622.

43 W. V. Quine. Concatenation as a basis for arithmetic. *The Journal of Symbolic Logic*, 11, 1946. doi:10.2307/2268308.

44 Alfred Tarski. Der wahrheitsbegriff in den formalisierten sprachen. *Studia Philosophica*, 1:261–405, 1935.

45 Jean van Heijenoort. *From Frege to Gödel : A Source Book in Mathematical Logic*. Harvard University Press, January 2002.

46 Georg Zetzsche. The complexity of downward closure comparisons. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 123:1–123:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.ICALP.2016.123.