On the Satisfiability of Straight line fragments over subword ordering

Soumodev Mal

Supervisors:
C. Aiswarya, CMI
Prakash Saivasan, IMSc

A thesis submitted in partial fulfilment
of the requirements for the degree
Master of Science
in
Computer Science

Chennai Mathematical Institute
June 2021

Abstract

We study the satisfiability of a subclass of string constraints. A string constraint impose constraints over variables that takes string values. Given a string constraint the satisfiability problem asks, if there is an assignment (for the variables over which it is defined) that satisfy the constraints. The subclass that we are looking at has 1) regular membership constraints (where the variable assignments must belong to specified regular languages) and 2) Subword constraints that relate variables by a subword relation. In addition, the variables are ordered and a variable can be related by a subword relation to a concatenation of variables having higher order than it. We call this constraint straight line subword ordered string(SL-SOS) constraints. We show that satisfiability problem for SL-SOS constraint is decidable and is in NEXPTIME. We also show that it is NP-hard. We prove that SL-SOS constraints can define non-regular tuples.

Acknowledgement

I am highly indebted to my supervisors Prakash Sir and Aiswarya mam for their constant support and all the fruitful discussions we had throughout till the completion of this thesis. All that I have learnt from them both academic and personal will help me immensely in my future career. Their scrupulous reviews on the report not only helped refine and structure it but also helped me learn how to write academic reports. I cannot thank them enough for their invaluable guidance and the time they devoted even during this raging pandemic.

I would like to express my deepest gratitude to all the professors at CMI for their excellent courses. They helped me to intensify my interests in Theoretical Computer Science. I would like to specially thank our faculty advisors Prof. B. Srivathsan for helping me with my presentation skills and for being there whenever I wanted some help and Prof. Prajakta Nimbhorkar for all the conversations and advices.

I would like to thank all my friends for their love and support. I am grateful to CMI for this opportunity to learn from the best and the invaluable friends I have made there.

Last but not the least, No words can express my gratitude towards my parents and my brother. Their constant encouragement, love and unfailing support is what made everything possible.

Contents

1 Introduction **1**

2 Preliminaries **6**

3 Satisfiability of SL-SOS Constraints **9**

3.1 Satisfiability of an SL-SOS constraint is NP-Hard 9

3.2 Satisfiability for SL-SOS constraints is decidable in NEXP-time 12

3.2.1 An improvement to the proof of Theorem 3.2: 15

4 SL-SOS Constraints need not be regular **17**

4.1 The Intersection (non)emptiness problem for SL-SOS constraints 20

5 Conclusion and Discussion **21**

Bibliography **22**

6 Appendix **25**

1 Introduction

Strings can be found everywhere throughout and beyond computer science [17, 25]. They are a fundamental datatype in almost every modern scripting and programming languages. String operations occur frequently in varying fields like model checking, database applications [18, 8, 7, 1] and so on.

String theories are extremely useful in the verification of string-manipulating programs [29] and analysis of security vulnerabilities of scripting languages. Many well-known security vulnerabilities like SQL injection occur due to mishandling of strings [27]. The detection of such issues are reduced to solving satisfiability of a string constraint [20]. String contraints impose constraints over variables that take strings as assignments. Some of the typical examples of string constraints are over string length, substring, string matching, (in)equality etc [28]. String constraint solving is put into limelight in recent years and there are many efficient string solvers that are currently available such as HAMPI [20], z3-str3 [5] and many more. Inspite of such advances, most of these solvers don't provide any completeness guarantees rather they work for specific instances. They are possibly non-terminating and may suffer from the performance issues [16]. The study of string constraints has to balance between expressiveness and complexity.

A direction of research is to find meaningful and expressive subclasses of string constraints for which satisfiability problem is decidable [1]. *The satisfiability/decidability problem for string constraints* asks if there is an assignment to the string variables satisfying the given constraints.

One of the major issues is that most of the expressive classes of string constraints are either undecidable or their decidability is left as an open question [13, 14, 15]. In the past, people have found logical extensions on string constraints such as over transducers [23] or word equations (string logic with concatenation operation) [24] etc. for which the satisfiability is decidable. Although, the satisfiability problem for a full class of string constraint combining transducer, concatenation, and length constraints is undecidable in general [26]. Even without transducer constraints, the decidability of analyzing a combination of concatenation and length constraints is still an open problem [6, 14]. Recent studies proposed that there are some sub-classes of these string logics (that are undecidable in general) for which the satisfiability problem is decidable [7].

When straightforward analysis is undecidable, an approach that has gained momentum in the recent past is of (regular) separability. This has been quite well studied for many classes of infinite state systems [21, 19, 11, 9]. It will be interesting to see if given two string constraints whether these can be regularly separable. Here, by regular we mean that, given that there are k variables over which the string constraint is defined, an assignment for

each of the k variable comes from an automaton over an extended alphabet of k dimension, i.e., a letter from the extended alphabet is a k-tuple of letters. An accepting word for the automaton will be a k-tuple of words where the i^{th} word in the tuple gives the assignment for the i^{th} variable for all $i \in \{1, \ldots, k\}$. If the regular separator can be effectively computed then many assertion checks can be automated, thanks to the decidability results on regular languages in the related works.

Our work: In this thesis, we follow this line of research and attempt to look into these two motivating questions for a particular class of string constraints. Our version of the string constraint (defined over a set of variables and alphabets) contains membership constraints (where the variable assignments must belong to specified regular languages) and subword ordered constraints. In addition, the variables are ordered and a variable can be related by a subword relation to a concatenation of variables having higher order than it. We call this restricted fragment as *Straight line subword ordered string (SL-SOS) constraints*.

In our first result, we show that checking the satisfiability for this subclass of string constraint is decidable. We get the upper bound for the satisfiability problem to be in NEXPTIME. We also prove that it is NP-hard. The proof of the former follows by establishing a bound on the length of the strings to be examined, based on the ordering of the variables. We proved that a variable can be bounded by atmost an exponential size to the input. Hence by guessing an assignment of appropriate size we obtain an NEXPTIME upper bound. We refined the procedure further by getting the minimal length for each string variables by looking at the depth[1] of each variable with respect to the subword ordered constraints. The proof of the latter (i.e.,NP-hardness for the defined class of string constraint) is a reduction from 3-SAT to the subword ordered string constraint with straight line restriction as we defined.

A collorary to the satisfiability problem for SL-SOS constraints is checking the intersection (non)emptiness problem for SL-SOS constraints is easily decidable. Since we can effectively get an SL-SOS constraint for the intersection of two given SL-SOS constraints. The intersection is trivial to get for the membership constraints following that regular languages are closed under intersection. So we just take the intersection of the automaton defined in the two constraints for every variable. Now, for the straight line variant of the subword ordered (SOSL) constraint we allow multiple constraints to have the same L.H.S. variable in contrast to the straight line definition in [23] where they don't allow multiple constraints. This allows us to simply take the union of the SOSL constraints. Hence, we can get the

[1]Depth of a variable with respect to the straight line variant of the subword ordering can be seen inductively as follows:

- A variable has depth 0 if it doesn't occur in the R.H.S. of any straight line constraint.
- A variable has depth i if it occurs in the R.H.S. of an straight line constraint whose L.H.S. variable is of depth $i-1$

intersection for two SL-SOS constraints. Hence, checking the intersection (non)emptiness is just to checking the satisfiability of the intersection of the two SL-SOS constraints. This result is in contrast with that of general SL constraints, for which decidability of intersection non-emptiness is not trivial where they used alternating automata as a symbolic representation for SL fragments.

Our second result is a step towards checking regular separability for SL-SOS constraints. We prove that the string constraint that we define may not regular and can define non-regular tuples. We proved that it might not be regular by giving a proof by contradiction using pumping lemma for the regular language over such extended alphabet. The fact that the constraint may not be regular also means that the separability problem is not immediate. Hence, it is still meaningful to look at the regular separability of SL-SOS constraints. The regular separability problem is not trivial because if they were regular then we would have directly concluded that the SL-SOS constraints are regularly separable since the regular separability problem would have just reduced to only checking intersection non-emptiness. Thus one of the *future motive* of our work is to answer the regular separability problem for these constraints and is now left as an open problem.

Related work: The straight line constraint is one of the subclasses of string constraints that have been studied quite extensively. The satisfiability of general string constraint is shown to be undecidable. This is because checking satisfiability of a transducer that realizes identity (i.e., a transducer T such that $T(s) = s$ for any string s) is already undecidable by a simple reduction from the Post Correspondence Problem (PCP) as proved in [26]. For this reason, an acyclicity restriction [3, 4] like the Straight line variant (which is shown [23] to be reasonable since it is typically satisfied by constraints that are generated by symbolic execution.) is often imposed to obtain decidability. Under this straight line restriction, the satisfiability was shown to be in EXPSPACE[23].

A Comparison of SL-SOS constraints with SL fragments: Our definition for straight line is more general than the straight line restriction[2] of the string constraints introduced by Lin and Barcelo in [23]. They define *the string constraint as the conjunction of membership constraints and relational constraints* where a relational constraint is a 2-tuple representing the input and output of a transducer. They say that a string constraint is straight-line if it is a conjunction of straight-line relational constraints (where the input is over a single variable and there is an ordering on the variables such that the output variables are of higher order than the input variable) along with membership constraints (membership constraint associates each variable with a regular language from which they can be assigned values) and there is atmost one constraint for each variable that occurs in the L.H.S. In our case, we allow multiple constraints for same variable occuring in the L.H.S.

[2]Similar notions that appear in the literature of string constraints include acyclicity [1] and solved form [14]

We see that the satisfiability problem is decidable for both the straight line string constraint defined in [23] and SL-SOS constraints that we proposed. But the latter is in NEXPTIME whereas the former has the complexity in EXPSPACE. This is overall a better result for SL-SOS constraint compared to that of the classical straight line string constraint.

The *regular separability* has been studied quite extensively in the past. The idea of separability is to decide whether two given languages are not only disjoint, but whether there exists a finite, easily verifiable, certificate for disjointness (and thus for safety). It has long been known that separability of context-free languages is undecidable [10] already for very simple classes of regular languages. This stifled the hope that separability would be decidable for any interesting classes of infinite-state systems and classes of separators. However, the Piecewise testable language turned out to have excellent decidability properties [12]. It was shown recently that for a wide range of language classes, it is decidable whether two given languages are separable by a piecewise testable language (PTL) [10, 2]. Even PTL separability is decidable for classical straight line string constraint as defined by Lin and Barcelo [2].

However, there hasn't been much studies on separability of string constraints in particular. The only paper that we know of to have studied the separability of straight line string constraints is [2] where they show that regular separability is undecidable for string constraints as well as straight line string constraints. Hence, they took a weaker separator that is Piece wise testable languages (PTL) and checked PTL separability. A PTL is a finite Boolean combination of special regular languages called piece languages of the form $\Sigma^* a_1 \Sigma^* a_2 \ldots \Sigma^* a_n \Sigma^*$, where all $a_j \in \Sigma$. For PTL separability of string constraint it was still undecidable but they got decidability for PTL separability of SL string constraints. However the complexity for the latter is still open. Then, they looked at Positive PTL separability of SL string constraints. PosPTL is obtained as a negation-free Boolean combination of piece languages. They proved the PSPACE completeness for the decidability of PosPTL separability of SL string constraints.

One of the future motive of the work is to answer the regular separability problem for SL-SOS constraints. The regular separability problem for two languages asks if there is a regular language that separates the two languages, i.e., if one of the language is a subset of the regular language and the other is disjoint from it. The fact that the constraint may not be regular also means that the separability problem is not as trivial. Because if they were regular then we would be able to directly conclude whether the SL-SOS constraint is regularly separable or not since the regular separability problem would have just reduced to only checking intersection non-emptiness. That is, if the intersection of the constraints is empty then we can simply take one of the constraint as the regular separator while the other will be clearly disjoint from the separator. If the intersection weren't empty then there won't be a separator anyway by definition and thus the constraint would not be regularly separable.

Thesis Outline: We have organized our work in three sections:

- In Section 2, we define the string constraint with subword ordering along with its straight line variant. The proof is done using pumping lemma over an instance of SL-SOS constraint that is not regular and this directly concludes that SOS constraint is also not regular since the straight line version is essentially a restricted version of the SOS constraint.

- In Section 4, We look at whether the straight line subword ordered string constraint is regular or not. It turns out that even straight line subword ordered string(SL-SOS) constraint may not be regular.

- Then in Section 3, we study the satisfiability of SL-SOS constraint and show that it is indeed decidable and prove it to be NP-hard in Subsection 3.1. The proof is a reduction from 3-SAT to the satisfiability of SL-SOS constraint.
  Then, we get an upper bound of NEXPTIME for SL-SOS constraint described in Subsection 3.2.
  Then, we gave a slightly better version of the NEXPTIME proof (described in the Subsection 3.2) in the although it doesn't change the overall complexity. Here, we checked the depth of each variable in terms of their occurrence in the R.H.S. of the list of constraints. That is, the depth of a variable is 0 if there is a variable that is not constrained (doesn't occur in the R.H.S. of any constraint) and the depth is i if there is a constraint where the L.H.S. variable is of depth $i-1$ and the former variable occurs in its R.H.S.

Straight Line Subword Ordered String Constraints

2 Preliminaries

Definition 2.1. Subwords: We say that y is a *subword* of x, when y is a word that is formed by dropping arbitrary many letters from the word x or in other words, the word y is embedded in the word x.
Formally, Given two words $u \in \Sigma^*$ and $v \in \Sigma^*$, u is a subword of v (denoted by $u \preceq v$) if $\exists$ a mapping $h : [1, |u|] \mapsto [1, |v|]$ [1] such that

1. $u[i] = v[h(i)], \forall\ i \in [1, |u|]$
2. $h(i) < h(j), \forall i < j$

For example, $\epsilon \preceq bab \preceq abacb$, $cab \not\preceq abacb$.

Definition 2.2. A **Subword Ordered String Constraint** Ψ can be defined as a 4-tuple $(\chi, \Sigma, (A_x)_{x\in\chi}, \mathcal{C})$ where

- χ is a finite non-empty set of variables
- Σ is a finite non-empty set, the input alphabet
- (Membership constraint) For each $x \in \chi$, A_x is a DFA describing the set of words that x can be mapped to
- $\mathcal{C}$ is a finite set of subword ordering of the form: $t \preceq t'$ where t, t' are string terms (i.e. concatenation of variables over χ).

An assignment η assigns a word over Σ to each variable $x \in \chi$. We say η satisfies Ψ, denoted $\eta \vDash \Psi$, if

[1] $[i, j]$ denotes the set $\{i, ..., j\}$ for $i, j \in \mathbb{N}$

1. $\eta(x) \in \mathcal{L}(\mathcal{A}_x)$ for all $x \in \chi$
2. For each constraint $t \preceq t'$ in $\mathcal{C}$ where $t = x_1 x_2 \dots x_n$ and $t' = y_1 y_2 \dots y_m$, we have, $\eta(x_1)\eta(x_2)\dots\eta(x_n) \preceq \eta(y_1)\eta(y_2)\dots\eta(y_m)$.

We define **Models**(Ψ) to be the set of all assignments that satisfy the subword ordered string constraint Ψ.

Definition 2.3. A **Straight Line Subword Ordered String(SL-SOS) constraints** $(\Psi, <)$ is a subword ordered string constraint $\Psi = (\chi, \Sigma, (A_x)_{x\in\chi}, \mathcal{C})$ together with an ordering $<$ on the variables in χ such that, for each $t \preceq t' \in \mathcal{C}$, $t = x$ for some $x \in \chi$ and $t' \in \{y \mid x < y\}^*$. We call this finite set of subword ordering with straight line restriction $\mathcal{C}$ as subword ordered straight line (SOSL) constraints.

Definition 2.4. We define **Models**$(\Psi, <)$ to be the set of all satisfying assignments for an SL-SOS constraint $(\Psi, <) = (\Sigma, \chi, (A_x)_{x\in\chi}, \mathcal{C})$.

Remark 1. Our definition of straight line constraint is more general than the one defined in [2, 23]. The paper [23] defines the straight line constraints as follows: Given the set of variables χ such that $|\chi| \geq m$ and alphabet Σ, a relational constraint of the form $\wedge_{i=1}^{m} x_i = P_i$ is straight line such that: 1) $x_1, \dots, x_m$ are different variables and is a subset of χ, 2) Each P_i uses only source variables in Ψ or variables from $\{x_1, \dots, x_{i-1}\}$. In other words, a variables must appear in the L.H.S. of atmost one constraints or there cannot be two constraints that have the same variable in its left hand side. In our definition, we allow multiple constraints with the same variable in their L.H.S. This relaxation will aid to a straightforward decidability of intersection (non)emptiness problem for SL-SOS constraints [See Section 4.1].

Example 1. We take a look at an example for SL-SOS constraints and see some satisfiable and unsatisfiable assignments.
Consider the SL-SOS constraint $(\Psi, <) = (\chi, \Sigma, (A_x)_{x\in\chi}, \mathcal{C})$ where

- $\chi = \{x, y, z\}$ with the ordering $x < y < z$
- $\Sigma = \{a, b\}$
- A_x, A_y and A_z is defined in Figure 1
- $\mathcal{C} = \{x \preceq yz, y \preceq z\}$

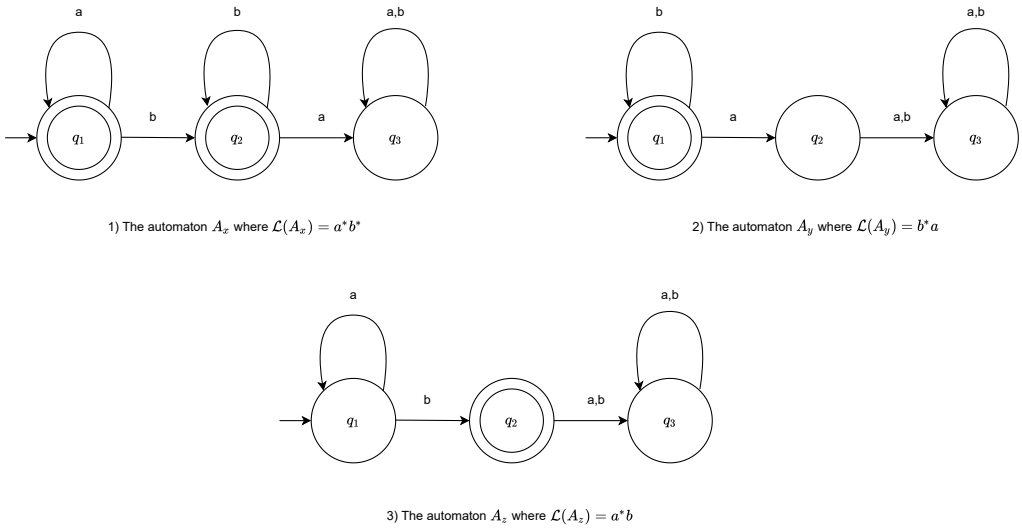

Figure 1: The membership constraint for each variable in χ

Note: $\mathcal{C}$ cannot have constraints of the form $xy \preceq z$ or $y \preceq x$ due to the straight line restriction on the subword ordering

Now, let us look at some satisfiable and unsatisfiable assignments.

- $(x = ab, y = a, z = ab) \vDash (\Psi, <)$ as it satisfies both the membership and SOSL constraints
- $(x = ba, y = a, z = ab) \nvDash (\Psi, <)$ since it violates the membership constraint A_x, i.e., $ba \notin \mathcal{L}(A_x)$
- $(x = abb, y = a, z = ab) \nvDash (\Psi, <)$ since it violates the SOSL constraint $x \preceq yz$.

Definition 2.5. An SL-SOS constraint $(\Psi, <) = (\Sigma, \chi, (A_x)_{x\in\chi}, \mathcal{C})$ is said to be **regular** if there is a finite state automaton A over an extended alphabet $(\Sigma \cup \epsilon)^n$ where $n = |\chi|$ whose language $\mathcal{L}(A)$ exactly defines Models$(\Psi, <)$. That is, the automaton A accepts words[2] of the form $(w_1, \ldots, w_n)$ iff there is a satisfying assignment η for $(\Psi, <)$ such that $\eta(x_1) = w_1, \eta(x_2) = w_2 \ldots \eta(x_n) = w_n$.

[2]a word in A is over $(\Sigma^*)^n$, i.e., we take the concatenation of letters coordinate-wise.

3 Satisfiability of SL-SOS Constraints

Definition 3.1. (Satisfiability of SL-SOSC (SL-SOSC-SAT)):The satisfiability of SL-SOS Constraints asks if:
Input: An SL-SOS constraint $(\Psi, <)$ over a set of variables χ
Question: Does there exist an η such that $\eta \vDash (\Psi, <)$ where η is an assignment for the set of variables in χ?

3.1 Satisfiability of an SL-SOS constraint is NP-Hard

Theorem 3.1. Satisfiability of a given SL-SOS constraint (SL-SOSC-SAT) is NP-Hard.

Proof. To prove that Satisfiability of SL-SOS constraint is NP-hard, we give a reduction from 3-SAT. For the reduction, take a 3-SAT instance $\varphi = c_1 \wedge c_2 \wedge c_3 \wedge ... \wedge c_k$ over a set of variables $V = \{x_1, ..., x_n\}$. We define the set of literals to be $L = V \cup \{\overline{x} \mid x \in V\}$. Each clause c_i is of the form $\ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3}$ where $\forall j \in \{1, 2, 3\}$, $\ell_{i_j} \in L$.

We define an SL-SOSC-SAT instance $(\Psi, <) = (\chi, \Sigma, (A_x)_{x \in \chi}, \mathcal{C})$ corresponding to the 3-SAT instance φ as follows

- $\chi = \{v_1, v_2, ..., v_k, z\}$
- $\Sigma = \{0, 1\}$
- *Membership Constraints:*
  - We let $\mathcal{L}(A_z) = (0 + 1)^n$ where A_z is a DFA describing the assignments that z can take.
  - For each $i \in \{1, \ldots, k\}$, Let $\mathcal{L}(A_{v_i}) = e_{i_1} + e_{i_2} + e_{i_3}$ that will ensure that atleast one of the three literals in clause c_i evaluates to True where $e_{i_j} = (0+1)^{m-1}x(0+1)^{n-m}$ such that
    - $x = 0$ if $\ell_{i_j} = \overline{x}_m$ in the clause c_i in φ.
    - $x = 1$ if $\ell_{i_j} = x_m$ in the clause c_i in φ.

  Note: A satisfying assignment for a variable in χ will be exactly of a bit vector of size n where n is the number of variables in φ.

- *Subword Ordered SL constraints $\mathcal{C}$:*
  - For every variable v_i that was introduced for each clause c_i, we have the constraint $v_i \preceq z$. This will ensure that for each $v_i \in \chi$, $v_i = z$, as each variable in χ is assigned a bit vector of length exactly n.

The reduction from 3-SAT to SL-SOS constraint takes polynomial time by Lemma 3.1. The correctness of the construction follows from Lemma 3.2 and 3.3. □

Lemma 3.1. The reduction in Theorem 3.1 can be computed in polynomial time.

Proof. Suppose that the original 3-CNF formula φ has k clauses, each of which has three literals over n variables. Then we construct an SL-SOS constraint with $k+1$ variables (k clause variable and a variable z) over the alphabet $\{0,1\}$. For each clause variable, we have its membership constraint to be a conjunction of 3 expressions for each literal in the clause. Each such expression is of length n. For each clause variable v, we have k SOSL constraints of the form $v \preceq z$. Each constraint can be constructed in polynomial time, so overall this reduction can be computed in polynomial time, as required. □

Lemma 3.2. If there is a satisfiable 3-SAT formula φ over n variables and k clauses then there is an SL-SOS constraint $(\Psi, <)$ with k variables that can be assigned words of length exactly n that is satisfiable .

Proof. If the 3-SAT instance φ is satisfiable then there is a satisfying assignment σ for the variables x_i in V, i.e., $\sigma : x_i \to \{\mathsf{True}, \mathsf{False}\}$. For the formula φ to be satisfiable, there must be atleast one literal in each clause that complies with σ. By our construction of $(\Psi, <)$ defined over variables $\{v_1, v_2, ..., v_k, z\}$ and alphabet $\{0,1\}$, we claim that there is a satisfying assignment η for $(\Psi, <)$ such that $\eta(z) = \sigma$, i.e.,

$$\forall i \in \{1, \ldots, n\}, \eta(z[i]) = 1 \text{ if } \sigma(x_i) = \mathsf{True} \text{ and } \eta(z[i]) = 0 \text{ if } \sigma(x_i) = \mathsf{False}$$ [3]

The SOSL constraints are of the form $v_i \preceq z$ for each clause variable v_i. This ensures that $|v_i| \geq |z|$. Now, by the membership constraints, we have each $\eta(v_i)$ either of the form e_{i_1}, e_{i_2} or e_{i_3}. Each $e_{i_j} = (0+1)^{m-1}x(0+1)^{n-m}$ where $x = 1$ if there is a literal $\ell_{i_j} = x_m$ and $x = 0$ if $\ell i_j = \overline{x}_m$ in the clause c_i. This implies that the clause variables can be assigned words of length exactly n. Thus $|v_i| = |z|$ and $v_i \preceq z$ tells that $\eta(v_i) = \eta(z)$ for all clause variable v_i. Since, we know that σ is a satisfying assignment, we take $\eta(v_i)$ for one of the literals that complies with σ and match the other bits with z such that $v_i = z$. Thus, η is a satisfying assignment for $(\Psi, <)$. Hence, $(\Psi, <)$ is satisfiable. □

[3] $\eta(z[i])$ is the i^{th} bit of the word assigned to z w.r.t. assignment η

Lemma 3.3. If an SL-SOS constraint $(\Psi, <)$ is satisfiable with k variables that can be assigned words of length exactly n then there is a 3-SAT formula φ over n variables and k clauses.

Proof. If the SL-SOS constraint $(\Psi, <)$ is satisfiable then there is a satisfying evaluation η that gives an assignment to each variable $v_1, v_2, \ldots, v_k, z$. Now, by our construction, a satisfying assignment for $(\Psi, <)$ is possible if all the variables have been assigned the same n length word, i.e., $\eta(v_1) = \eta(v_2) = \cdots = \eta(v_k) = \eta(z) = u$ *(say)*. We claim that $\sigma = \eta(z)$ [4] is a satisfying assignment for the 3-SAT formula φ which has k clauses over n variables. To see this, we know that each variable v_i can be assigned bit vectors of the form e_{i_1}, e_{i_2} or e_{i_3} where e_{i_j} has it's m^{th} bit to be 0 if $\ell_{i_j} = \overline{x}_m$ in clause c_i of φ and 1 if $\ell_{i_j} = x_m$. In other words, e_{i_1}, e_{i_2} and e_{i_3} corresponds to the three literals in clause c_i in φ. Hence, each $\eta(v_i)$ has a fixed bit in its bit vector that corresponds to one of the literals in the clause c_i of φ. Thus, contributing to one bit of the bit vector u. Hence, we get a satisfying assignment for φ. Therefore, φ is satisfiable. □

[4] $\sigma = \eta(z) \Leftrightarrow \forall v_i \in V,\ \sigma(v_i) = \mathsf{True}$ if $\eta(z[i]) = 1$ and $\sigma(v_i) = \mathsf{False}$ if $\eta(z[i]) = 0$

3.2 Satisfiability for SL-SOS constraints is decidable in NEXP-time

Assume that the number of states for each finite state automata $(A_x)_{x\in\chi}$ is M for the SL-SOS constraint $(\Psi, <)$.

Lemma 3.4. If there is a satisfying assignment η for an SL-SOS constraint $(\Psi, <)$ where $\chi = \{x_1, \ldots, x_k\}$, then for every $\ell \in \{1, \ldots, k\}$, there is another satisfying assignment η_ℓ such that

$$\eta_\ell(x_i) = \begin{cases} \eta(x_i) & \text{if } i \neq \ell \\ w_\ell & \text{otherwise} \end{cases}$$

with $|w_\ell| \leq M \times \left(1 + \sum_{i=1}^{\ell-1} n_i.c_i\right) - 1$ where $n_i = |\eta(x_i)|$ and c_i is the number of SOSL constraints in $(\Psi, <)$ with x_i as the L.H.S.

Proof. Assuming that there is a satisfying assignment η for the SL-SOS constraint $(\Psi, <)$ over $\chi = \{x_1, x_2, \ldots, x_k\}$.
When $\ell = 1$, we will show that there exists a word w_1 such that $\eta_1(x_1) = w_1$ and $|w_1| \leq M \times (0 + 1) - 1 = M - 1$. Since x_1 doesn't occur in the R.H.S. of any SLSO constraint, it doesn't depend on any other variable for its assignment. To get a minimal word we see that if there is a word of size greater than $M - 1$ then there will be atleast one state that is repeated, by pigeonhole principle. Now, we can shrink the subsequence starting and ending with the same state to get a smaller accepting run in the automaton A_{x_1}. We can repeatedly shrink such portion of the run until there are no repeating state in the run for $\eta_1(x_1)$. If there are no repeating state then the length of w_1 can be atmost $M - 1$. Hence, for $\ell = 1$, the lemma holds.

Now, for $\ell = i,$ such that $i > 1$, let for all $j \in \{1, \ldots, i\}$ there be c_j many SOSL constraints in $(\Psi, <)$ with x_j as the L.H.S. and $n_j = |\eta(x_j)|$. In the worst case, x_i may occur in all the c_j constraints with x_j as the L.H.S. for all $j < i$ and $\eta(x_i)$ contains each $\eta(x_j)$ entirely with no overlapping of letters over the words that are embedded into $\eta_\ell(x_i)$. For e.g., $\mathcal{L}(A_x) \in \{baab\}, \mathcal{L}(A_{x'}) \in a^*ba^*, \mathcal{L}(A_{x_b}) \in \{b\}$, and $\mathcal{C} = \{x \preceq x'x_b, x \preceq x_bx'\}$ here, in the first constraint the a's in $\mathcal{L}(A_x)$ are embedded in the right hand side of b in the valuation of x' whereas in the second constraint it is embedded in the right hand side of b. From the above argument, w_ℓ will be of the form

$$u_1b_1u_2b_2 \ldots u_zb_zu_{z+1}$$

where we have $u_p \in \Sigma^*$ for all $p \in \{1, \ldots, (z + 1)\}$ and for all variables $x_j \leq x_i$, $\eta(x_j) \preceq b_1b_2b_3 \ldots b_z$ which implies that $z \leq c_i.n_1 + c_2.n_2 + \cdots + c_{i-1}.n_{i-1}$. Now by the similar argument used for $\ell = 1$, for all $p \in \{1, \ldots, (z + 1)\}$, we can shrink the word w_p in order to

get the length for w_p to be less than or equals $M-1$. Thus, by the form of w_ℓ, it's minimal length will be bounded by $(M-1)\times(1+\sum_{i=1}^{\ell-1} n_i.c_i)+(\sum_{i=1}^{\ell-1} n_i.c_i) = M\times\left(1+\sum_{i=1}^{\ell-1} n_i.c_i\right)-1$. $\square$

Theorem 3.2. Satisfiability of a given SL-SOS constraint is decidable and has NEXP-TIME upper bound.

Proof. Given an SL-SOS constraint $(\Psi,<)$ over $\chi=\{x_1,\ldots,x_k\}$ ordered as $x_1<x_2<\cdots<x_k$ and alphabet Σ, deciding the satisfiability of an SL-SOS constraint $(\Psi,<)$ can be done in NEXP time by non-deterministically guessing the valuation of the variables where the maximum bound on their length is exponential i.e., M^k.

We claim that if $(\Psi,<)$ has a satisfying assignment η then there exists a set of satisfying assignments $\{\hat{\eta}_1,\hat{\eta}_2,\ldots,\hat{\eta}_k\}$ such that for each $\ell\in\{1,\ldots,k\}$ and $j\leq\ell$, $\hat{\eta}_\ell(x_j)$ is upper bounded by $M\times\left(1+\sum_{i=1}^{j-1} n_i.c_i\right)-1$ where $n_i=|\eta(x_i)|$ and c_i is the number of SOSL constraints in $(\Psi,<)$ with x_i as the L.H.S.
We prove the above claim by induction on the existence of a satisfying assignment $\hat{\eta}_\ell$.

Base case: When $\ell=1$, we show that $\hat{\eta}_1$ is a satisfying assignment for $(\Psi,<)$.
This directly follows from Lemma 3.4, that is, if there is an assignment η for $(\Psi,<)$, then there is another satisfying assignment η_1 where $\eta_1(x_i)=\begin{cases}\eta(x_i) & \text{if } i\neq 1\\ w_1 & \text{otherwise}\end{cases}$
with $w_1\leq M\times(1+0)-1=M-1$. Hence, from the definition of $\hat{\eta}_1$ we can put the assignment $\hat{\eta}_1=\eta_1$.

Induction hypothesis: Let us assume the claim is true for $\hat{\eta}_m$, i.e., there is a satisfying assignment $\hat{\eta}_m$ for $(\Psi,<)$ where for all $j\leq m$, $\hat{\eta}_m(x_j)$ is bounded by $M\times\left(1+\sum_{i=1}^{j-1} n_i.c_i\right)-1$.

Inductive step: Now, we show that there is a satisfying assignment $\hat{\eta}_{m+1}$. From the hypothesis, we know that $\hat{\eta}_m$ is a satisfying assignment for $(\Psi,<)$. From Lemma 3.4, we take the satisfying assignment $\eta=\hat{\eta}_m$ and thus, there is another assignment $\hat{\eta}_{m+1}$ such that

$$\hat{\eta}_{m+1}(x_j)=\begin{cases}\hat{\eta}(x_j) & \text{if } j\neq m+1\\ w_{m+1} & \text{otherwise}\end{cases}$$

with $w_{m+1}\leq M\times\left(1+\sum_{i=1}^{m} n_i.c_i\right)-1$.
Hence, the assignment $\hat{\eta}_{m+1}$ satisfies $(\Psi,<)$ and for each $j\leq m+1$, $\hat{\eta}_{m+1}(x_j)$ is bounded by $M\times\left(1+\sum_{i=1}^{j-1} n_i.c_i\right)-1$.

From the above induction, we have an assignment $\hat{\eta}_k$ where for each variable $x_\ell\in\chi$, $\hat{\eta}_k(x_\ell)$

is bounded by $M \times \left(1 + \sum_{i=1}^{\ell-1} n_i.c_i\right)$. Now we can non-deterministically choose assignments for each variable from this bounded space if there is a satisfying assignment. Otherwise we have $(\Psi, <)$ unsatisfiable.

Now, we claim that for the assignment η^k, the length of the string assigned to the variable x_i, $|\eta^k(x_i)|$ is $\mathcal{O}(M^i)$. We show that by an induction on the bounds of the variables in order.

Base case: When $i = 1$,

From Lemma 3.4, it follows that $|\eta^1(x_1)| \leq M - 1$. Now, $\eta^2(x_1) = \eta^1(x_1), \eta^3(x_1) = \eta^2(x_1), \ldots, \eta^k(x_1) = \eta^{k-1}(x_1)$ implies that $|\eta^k(x_1)| \leq M - 1$ i.e., the assignment η^k for x_1 is bounded by $\mathcal{O}(M)$.

Inductive step: When $i = j + 1$,

From Lemma 3.4, we see that the maximum length of $|\eta^{j+1}(x_{j+1})|$ is $M \times (1 + |\eta^{j+1}(x_1)|.c_1 + \cdots + |\eta^{j+1}(x_j)|.c_j) - 1$. Thus, the maximum length of $|\eta^{j+1}(x_{j+1})|$ is $M \times (1 + \mathcal{O}(M^j)) - 1 = \mathcal{O}(M^{j+1})$. Now, $\eta^{j+2}(x_{j+1}) = \eta^{j+1}(x_{j+1}), \eta^{j+3}(x_{j+1}) = \eta^{j+2}(x_{j+1}), \ldots, \eta^k(x_{j+1}) = \eta^{k-1}(x_{j+1})$ implies that $|\eta^k(x_{j+1})| \leq M \times (1 + |\eta^{j+1}(x_1)|.c_1 + \cdots + |\eta^{j+1}(x_j)|.c_j) - 1$, i.e., the assignment η^k for x_{j+1} is bounded by $\mathcal{O}(M^{j+1})$.Now, from hypothesis, we have $|\eta^k(x_p)| = \mathcal{O}(M^p)$ for all $p \in \{1, \ldots, j\}$. Hence, for all $i \leq j + 1$, $|\eta^k(x_i)|$ is bounded by $\mathcal{O}(M^i)$

Hence, we have proved that the assignment η^k is such that $|\eta^k(x_i)|$ is bounded by $\mathcal{O}(M^i)$ for all variables $x_i \in \chi$. We can conclude that we have to non-deterministically guess a satisfying assignment for variables that are bounded exponentially w.r.t. the given SL-SOS constraint. Thus, it takes NEXP time to decide the satisfiability of SL-SOS constraints. □

3.2.1 An improvement to the proof of Theorem 3.2:

The bound that we calculated above for SL-SOS constraint $(\Psi, <)$ is assuming the worst case, where each $x_i \in \chi$ may occur in all the constraints with x_j as the L.H.S. for all $j < i$ and $\eta(x_i)$ covers each $\eta(x_j)$ entirely with no overlapping of letters that are embedded into $\eta_\ell(x_i)$. But the actual bound might be much less for a given SL-SOS constraint $(\Psi, <)$. In order to get a tighter bound we can do the following:

1. For each variable x_i in χ, we maintain a depth function $d(x_i)$ that will map the variables to the maximum depth in which the variable x_i occurs w.r.t. the SOSL constraints. Also, define $depend(x)$ to be a multiset [5] of variables x' occurring in the L.H.S. of SOSL constraints of the form $x' \preceq t$ where $x \in t$. Initially take $d(x)$ to be 0 and $depend(x)$ to be empty for all $x \in \chi$.
   Note: x_1 doesn't occur in the R.H.S. of any SOSL constraint, hence, it's depth number $d(x_1)$ is always 0.

2. For all $x \in \chi$ in order,
   Initialize max to -1. where max stores the maximum depth for x
   For all the SOSL constraints $\mathcal{C}_i \in \mathcal{C}$ that have x' as the L.H.S. where $x' < x$,
   If x occurs in the R.H.S. of $\mathcal{C}_i$,
   Update $depend(x) = depend(x) \cup \{x'\}$
   If $d(x') \geq max$, then Update max with $d(x')$
   Update $d(x) = max + 1$

It is easy to see that the algorithm terminates as it iterates over all the variables and stops when all the variables are processed accordingly. Now, we claim that for a variable x in χ with depth $d(x)$, there is an assignment η such that the $\eta(x)$ is bounded by $\mathcal{O}(M^{d(x)-1})$. We prove the claim by induction on the depth of the variable.

Base case: For all $d(x) = 0$, since these variables do not occur in the R.H.S. of any SOSL constraint and it doesn't depend on any other variable for its assignment. So there are no letters that have to be embedded in the assignment for the variables of depth 0. Hence, we can just take their assignment to be from the set of minimal words of their respective membership constraints as defined in 6.3 where we repeatedly shrink the loops in the run till there are no repeating state in the run for $\eta(x)$. This implies that $\eta(x)$ is bounded by $M - 1$ since the longest run will visit all the M states in the automata atmost once.

[5] The function *depend* maps a variable to a multiset because there might be constraints of the form $x \preceq t$ and $x \preceq t'$ where there is a variable x' in both t and t' such that the same word x is embedded in different position in x' for t and t'.
For e.g., $\mathcal{L}(A_x) \in \{baab\}, \mathcal{L}(A_{x'}) \in a^*ba^*, \mathcal{L}(A_{x_b}) \in \{b\}$, and $\mathcal{C} = \{x \preceq x'x_b, x \preceq x_bx'\}$ here, in the first constraint the a's in $\mathcal{L}(A_x)$ are embedded in the right hand side of b in the valuation of x' whereas in the second constraint it is embedded in the right hand side of b.

Induction hypothesis: Assume that for all variables x of depth $d(x)$ less than or equals n, $\eta(x)$ is bounded by $\mathcal{O}(M^{d(x)-1})$.
Inductive step: For variables x' with depth $d(x') = n+1$, we can calculate the bound for $\eta(x')$ to be less than $(M-1) \times \left(1 + \sum_{x'' \in\ depend(x)} |\eta(x'')|\right) + \left(\sum_{x'' \in\ depend(x)} |\eta(x'')|\right)$
$= \times \left(1 + \sum_{x'' \in\ depend(x)} |\eta(x'')|\right) - 1$ which follows a similar argument of Lemma 3.4 where we explicitly added the maximal bounds of the variables (of lower order than x') occurring in the L.H.S.of a constraint but here, we consider only the variables in $depend(x')$.

Now, all the variables of $depend(x')$ will be of lower depth ,i.e., strictly less than $n+1$, if there were a variable $\geq n+1$ then the depth of x' would be greater than $n+1$(contradiction).There will be atleast one variables of depth 0 to n. Now, from our hypothesis, we substitute the bounds for these variables to get $M \times (1 + \mathcal{O}(M^n)) - 1 = \mathcal{O}(M^{n+1})$.

Note: Instead of guessing the variables of depth 0 we can evaluate their Minimal words as defined in Definition 6.3 and iterate through all the words in the set of minimal words Min(A_x) for all x that has depth 0. Even though the complexity will still be NEXPTIME *since we have to anyways guess all the other variables that have depth greater than 0 but this will be useful for cases where the minimal words are very less and the variables with depth greater than 0 is also less.*

4 SL-SOS Constraints need not be regular

In this section, we prove that the SL-SOS constraint need not be regular. This is an important result with respect to the regular separability of the SL-SOS constraint. If SL-SOS constraints were regular then trivially they are also regularly separable. Since we can just take the regular separator to be either of the language that we are required to separate which implies that we just need to check if they are disjoint. By proving that SL-SOS constraints may not be regular, it will be interesting to see if they are regular separability problem.

Theorem 4.1. Straight Line Subword Ordered String Constraints may not be regular.

Proof. Consider an SL-SOS Constraints $(\Psi, <) = (\Sigma, \chi, (\mathcal{A}_x)_{x\in\chi}, \mathcal{C})$ where

- $\Sigma = \{a, b\}$
- $\chi = \{x, y, z\}$
- $\mathcal{L}(A_x) = a^*b^*,\ \mathcal{L}(A_y) = a^*,\ \mathcal{L}(A_z) = (ba)^*$
- $\mathcal{C} = \{x \preceq yz,\ y \preceq z\}$

A satisfying assignment η for Ψ will be of the form:

$$x \in a^i b^j,\ y \in a^k,\ z \in (ba)^l$$

where the following conditions must hold

1. $i + j \leq k + l$
2. $k \leq l$
3. $j \leq l$

We prove that Models$(\Psi, <)$ is not regular by contradiction. Suppose Models$(\Psi, <)$ is regular. Then from definition 2.5, there is an automaton $\mathcal{A}$ (say with n states) that accepts Models$(\Psi, <)$ where the alphabet is a triple (a_x, a_y, a_z) over $(\Sigma \cup \epsilon)^3$.
Now, We assign the pumping length to be n (the number of states in A) given by the pumping lemma.
Take the string $w = (a^n b^n, a^n, (ba)^n) \in$ Models$(\Psi, <)$. Because w is a member of A and w has length more than n, the pumping lemma guarantees that w can be split into three pieces, $w = pqr$ such that $|y| \geq n$ and y can be split further into three pieces tuv so that

$u \neq \epsilon$ and for all $i \geq 0$, the string $ptu^ivq \in A$. Now, we look at the following cases to show that the above claim is impossible:

Consider $p = q = \epsilon$. [Note:We ignore t and v for convenience.]

case 1. When u is of the form $(a^f, a^g, (ba)^h)$ where $f, g, h > 0$
Pumping down u, we get, $ptvr$ of the form $(a^{n-f}b^n, a^{n-g}, (ba)^{n-h})$. This violates condition 3 since $n \not\leq n - h$. Hence, $ptvr \notin \text{Models}(\Psi)$

case 2. When u is of the form $(b^f, a^g, (ba)^h)$ where $f, g, h > 0$ and $n \geq f$.

case 2.1. When $f < h$.
Pumping down u, we get, $ptvr$ of the form $(a^n b^{n-f}, a^{n-g}, (ba)^{n-h})$ This clearly violates condition 3, i.e., $n - f \not\leq n - h$, hence, $ptvr \notin \text{Models}(\Psi)$

case 2.2. When $f > h$.
Pumping u twice, we get, ptu^2vr of the form $(a^n b^{n+f}, a^{n+g}, (ba)^{n+h})$. This will clearly violate condition 3 since $n + f \not\leq n + h$. Hence, $ptu^2vr \notin \text{Models}(\Psi)$

case 2.3. When $f = h$.
Pumping down u, we get, $ptvr$ of the form $(a^n b^{n-f}, a^{n-g}, (ba)^{n-h})$. This will violate the condition 1 because $(n) + (n - f) \not\leq (n - g) + (n - h) \implies 2n - f \not\leq 2n - f - g$. Thus, $ptvr \notin \text{Models}(\Psi)$

Let us take $x', y', z' \neq \epsilon$ to be valid substrings of w_x, w_y, w_z respectively.

case 3. When u is of the form $(a^f b^g, y', z')$
Pumping u twice, we get, ptu^2vr of the form $(a^{n-f}a^f b^g a^f b^g b^{n-g}, y'', z'')$. Since w_x is not of the form a^*b^*, $ptu^2vr \notin \text{Models}(\Psi)$

case 4. When u is of the form (x', y', a) *or* (x', y', b)
Pumping u twice, we get, ptu^2vr of the form $(x'', y'', (ba)^{z_1}b^2a(ba)^{z_2})$ or
$(x'', y'', (ba)^{z_1}ba^2(ba)^{z_2})$ where $z_1 + z_2 + 1 = n$. Since w_z is not of the form $(ba)^*$,hence, $ptu^2vr \notin \text{Models}(\Psi)$

case 5. When u is of the form (ϵ, y', z') where $y' = a^g, z' = (ba)^h$
Pumping down u, we get, $ptvr$ of the form $(a^n b^n, a^{n-g}, (ba)^{n-h})$ Now, this violates condition 3 because $n \not\leq n - h$. Hence, $ptvr \notin \text{Models}(\Psi)$.
Note: $z' \in \{a, b\}$ was covered in case 4

case 6. When u is of the form (x', ϵ, z') or (ϵ, ϵ, z') where x is not of the form $a^f b^g$.
Pumping down u, we get, $ptvr$ of the form $(a^{f_1}b^{f_2}, a^n, (ba)^{n-h})$ where $f_1, f_2 \leq n$. This violates condition 2 since $n \not\leq n - h$. Hence, $ptvr \notin \text{Models}(\Psi)$
Note: $x' = a^f b^g$ was covered in case 3

case 7. When u is of the form (x', y', ϵ) or (ϵ, y', ϵ)
Pumping u twice, we get, ptu^2vr of the form $(x'', a^{n+g}, (ba)^n$. Since $n + g \not\leq n$, hence it violates condition 2. So $ptu^2vr \notin$ Models(Ψ)

case 8. When u is of the form (x', ϵ, ϵ).
Pumping u twice, we get, ptu^2vr of the form $(x'', a^n, (ba)^n)$ where

- If $x' = a^f$ then $x'' = a^{f+n}b^n$
- If $x' = b^f$ then $x'' = a^n b^{f+n}$

In any of the above cases, condition 1 is violated as $2n + f \not\leq 2n$. Hence, $ptu^2vr \notin \mathcal{L}(\Psi)$.
Note: $x' = a^f b^g$ is covered in case 3.

Since a contradiction is unavoidable if we make the assumption that A is regular, hence A is infact not regular. Thus, by the above result, we see that a language in SL-SOS constraint may not be regular. □

4.1 The Intersection (non)emptiness problem for SL-SOS constraints

Although we saw that SL-SOS constraints may not be regular, it is still interesting to note that the intersection emptiness problem for this constraints is easily decidable.

Definition 4.1. The **intersection (non)emptiness problem** for SL-SOS constraints asks if:
Input: Two SL-SOS constraints $(\Psi_1, <), (\Psi_2, <)$ over the same set of variables χ, same alphabet Σ and with the same ordering $<$
Question: Is the intersection between the two SL-SOS constraints (non)empty, i.e., if $\text{Models}(\Psi_1, <) \cap \text{Models}(\Psi_1, <) = \emptyset$?

Theorem 4.2. The intersection (non)emptiness problem for SL-SOS constraints is decidable.

Proof. Given two SL-SOS constraints defined over the same vocabulary[6] say, $(\Psi_1, <) = (\chi, \Sigma, (A_x)_{x\in\chi}, \mathcal{C}_1)$ and $(\Psi_2, <) = (\chi, \Sigma, (B_x)_{x\in\chi}, \mathcal{C}_2)$, the intersection of these two constraints is an SL-SOS constraint given as follows: $(\Psi_{12}, <) = (\chi, \Sigma, (C_x)_{x\in\chi}, \mathcal{C}_{12})$ where $(C_x)_{x\in\chi} = (A_x \cap B_x)_{x\in\chi}$ and $\mathcal{C}_{12} = (C)_1 \cup (C)_2$. Since, regular languages are closed under intersection, we get a single automata for each variable in Ψ_{12}. Since we are using a relaxed version of straight line restriction (as defined by [23]) where we can have multiple constraints for the same L.H.S. variable, we just take the union of all the SOSL constraint. Thus, Ψ_{12} is indeed an SL-SOS constraint and it defines the intersection of Ψ_1 and Ψ_2 since any assignment for Ψ_{12} must satisfy both the membership constraint and the SOSL constraints for Ψ_1 and Ψ_2 by the construction. Now, checking the intersection emptiness is reduced to checking the satisfiability of the SL-SOS constraint Ψ_{12} which we know is decidable in NEXPTIME. □

[6] By same vocabulary we mean that the constraints are defined over the same alphabet and set of variables along with the same ordering on the variables

5 Conclusion and Discussion

In this thesis, we introduced a new modified version of the straight line string constraint introduced first in [23] with subword ordering. We first looked at the satisfiability problem of the SL-SOS constraints. We showed that it is decidable in NEXPTIME. We also proved that the satisfiability problem of SL-SOS constraints is NP-hard. This result is is better than the satisfiability of SL string constraints in [23] that is in EXPSPACE. Then we studied that the SL-SOS constraints need not be regular. This also implies that SOS constraints may not be regular as well. The result influenced our future aim of checking the regular separability of the constraint since if it were regular then solving regular separability would be trivial. Lastly, we saw that the decidability of intersection (non)emptiness problem is quite straightforward for SL-SOS constraint. In contrast, getting a constraint for the intersection of for the general Straight line fragment defined by [23] is not immediate where there cannot be multiple constraints for the same variable. This implies that we can't just take directly the union of the straight line constraints.

Future Work:

- To **reduce the gap between the complexity result** in deciding the satisfiability of the SL-SOS constraints - Since the upper bound and lower bound that we got for the satisfiability of SL-SOS constraint is quite wide, a future work would be to get better bounds that would shrink the gap between the upper and the lower bound.
- To check if we can decide the **satisfiability problem for Subword ordered string constraints** - Since, the satisfiability of string constraints over concatenation and length constraints is a long standing open problem and subword ordered string constraints are closely related to those constraints, it will be interesting to come up with the satisfiability result for the latter since the result could help in solving the former.
- To look if the **SL-SOS constraint is regularly separable** - Since from section 4 we got that SL-SOS constraints may not be regular, one can try to check if it is decidable whether an SL-SOS constraint is regularly separable or not.
- To have **tool implementation for the satisfiability problem** for the SL-SOS constraints - Even for some undecidable constraints in general there are tools that work for specific instances for those constraints [20, 5]. Our result that the satisfiability of SL-SOS constraints is decidable and has NEXPTIME upper bound, motivates the need for a tool implementation for the same.
- There are **possibilities to work on theories related to presburger arithmetic** and our SL-SOS constraint. One can think of an algorithm to translate satisfiability of SL-SOS constraint to satisfiability of a presburger formula or other expressible logic for SL-SOS constraints.

Bibliography

[1] P. A. Abdulla, M. F. Atig, Y.-F. Chen, L. Holík, A. Rezine, P. Rümmer, and J. Stenman. String constraints for verification. In A. Biere and R. Bloem, editors, *Computer Aided Verification*, pages 150–166, Cham, 2014. Springer International Publishing.

[2] P. A. Abdulla, M. F. Atig, V. Dave, and S. N. Krishna. On the Separability Problem of String Constraints. In I. Konnov and L. Kovács, editors, *31st International Conference on Concurrency Theory (CONCUR 2020)*, volume 171 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:19, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[3] P. Barceló, D. Figueira, and L. Libkin. Graph Logics with Rational Relations. *Logical Methods in Computer Science*, 9(3), 2013.

[4] W. Bekker and V. Goranko. Symbolic model checking of tense logics on rational kripke models. In M. Archibald, V. Brattka, V. Goranko, and B. Löwe, editors, *Infinity in Logic and Computation*, pages 2–20, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[5] M. Berzish, V. Ganesh, and Y. Zheng. Z3str3: A string solver with theory-aware heuristics. In *2017 Formal Methods in Computer Aided Design (FMCAD)*, pages 55–59, Oct 2017.

[6] J. R. Büchi and S. Senger. *Definability in the Existential Theory of Concatenation and Undecidable Extensions of this Theory*, pages 671–683. Springer New York, New York, NY, 1990.

[7] T. Chen, Y. Chen, M. Hague, A. W. Lin, and Z. Wu. What is decidable about string constraints with the replaceall function. *Proc. ACM Program. Lang.*, 2(POPL), Dec. 2017.

[8] T. Chen, M. Hague, A. W. Lin, P. Rümmer, and Z. Wu. Decision procedures for path feasibility of string-manipulating programs with complex operations. *Proc. ACM Program. Lang.*, 3(POPL), Jan. 2019.

[9] L. Clemente, W. Czerwiński, S. Lasota, and C. Paperman. Regular Separability of Parikh Automata. In *The 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017))*, Varsovie, Poland, 2017.

[10] W. Czerwiński, W. Martens, L. van Rooijen, and M. Zeitoun. A note on decidable separability by piecewise testable languages. In A. Kosowski and I. Walukiewicz, editors, *Fundamentals of Computation Theory*, pages 173–185, Cham, 2015. Springer International Publishing.

[11] W. Czerwiński and S. Lasota. Regular separability of one counter automata. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, 2017.

[12] W. Czerwiński, W. Martens, L. van Rooijen, M. Zeitoun, and G. Zetzsche. A Characterization for Decidable Separability by Piecewise Testable Languages. *Discrete Mathematics & Theoretical Computer Science*, Vol. 19 no. 4, FCT '15, Dec. 2017.

[13] V. Ganesh and M. Berzish. Undecidability of a theory of strings, linear arithmetic over length, and string-number conversion. *CoRR*, abs/1605.09442, 2016.

[14] V. Ganesh, M. Minnes, A. Solar-Lezama, and M. Rinard. Word equations with length constraints: What's decidable? In A. Biere, A. Nahir, and T. Vos, editors, *Hardware and Software: Verification and Testing*, pages 209–226, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[15] V. Ganesh, M. Minnes, A. Solar-Lezama, and M. C. Rinard. (un)decidability results for word equations with length and regular expression constraints. *CoRR*, abs/1306.6054, 2013.

[16] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Dpll(t): Fast decision procedures. In R. Alur and D. A. Peled, editors, *Computer Aided Verification*, pages 175–188, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[17] G. H. Gonnet. Some string matching problems from bioinformatics which still need better solutions. *Journal of Discrete Algorithms*, 2(1):3–15, 2004. The 9th International Symposium on String Processing and Information Retrieval.

[18] H. Hojjat, P. Rümmer, and A. Shamakhi. On strings in software model checking. In A. W. Lin, editor, *Programming Languages and Systems*, pages 19–30, Cham, 2019. Springer International Publishing.

[19] H. B. Hunt. On the decidability of grammar problems. *J. ACM*, 29(2):429–447, Apr. 1982.

[20] A. Kiezun, V. Ganesh, P. J. Guo, P. Hooimeijer, and M. D. Ernst. Hampi: A solver for string constraints. In *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis*, ISSTA '09, page 105–116, New York, NY, USA, 2009. Association for Computing Machinery.

[21] E. Kopczyński. Invisible pushdown languages. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16, page 867–872, New York, NY, USA, 2016. Association for Computing Machinery.

[22] D. Kozen. Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 254–266, 1977.

[23] A. W. Lin and P. Barceló. String solving with word equations and transducers: Towards a logic for analysing mutation xss. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '16, page 123–136, New York, NY, USA, 2016. Association for Computing Machinery.

[24] G. S. Makanin. The Problem of Solvability of Equations in a Free Semigroup. *Mathematics of the USSR-Sbornik*, 32(2):129–198, feb 1977.

[25] F. Min and X. Wu. A comparative study of pattern matching algorithms on sequences. In H. Sakai, M. K. Chakraborty, A. E. Hassanien, D. Slkezak, and W. Zhu, editors, *Rough Sets, Fuzzy Sets, Data Mining and Granular Computing*, pages 510–517, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[26] C. Morvan. On rational graphs. In J. Tiuryn, editor, *Foundations of Software Science and Computation Structures*, pages 252–266, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[27] X. Ping-Chen. Sql injection attack and guard technical research. *Procedia Engineering*, 15:4131–4135, 2011. CEIS 2011.

[28] J. D. Scott, P. Flener, and J. Pearson. Constraint solving on bounded string variables. In L. Michel, editor, *Integration of AI and OR Techniques in Constraint Programming*, pages 375–392, Cham, 2015. Springer International Publishing.

[29] F. Yu, T. Bultan, and B. Hardekopf. String abstractions for string verification. In A. Groce and M. Musuvathi, editors, *Model Checking Software*, pages 20–37, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

6 Appendix

Definition 6.1. (Finite Automata intersection problem): Let $A_1, A_2, ..., A_k$ be k DFAs with a common alphabet Σ and let $\mathcal{L}(A_i)$ be the language accepted by the automaton A_i. The problem INT is to determine if all the A_i's accept a common element of Σ^*.

$$INT = \{< A_1, ..., A_k > | \bigcap_{i=1}^{k} \mathcal{L}(A_i) \text{ is nonempty}\}$$

Lemma 6.1. INT is $\leq_{log}$ - complete for PSPACE.[22]

Lemma 6.2. Concatenation of k DFA's takes linear time.

Proof. Let there be k DFA namely $A_1, \ldots, A_k$ and we want the concatenation to be like $A_1.A_2. \ \ldots \ .A_k$.We want to get an automaton that accepts a word w such that $w = w_1 w_2 \ldots w_k$ where $w_1 \in \mathcal{L}(A_1), \ldots, w_k \in \mathcal{L}(A_k)$ we define the automaton $A = (Q, \Sigma, \delta, s, F)$ such that Q is the union over all the states in k DFAs $A_1, \ldots, A_k$, the alphabet Σ is union over all the alphabets defined for $A_1, \ldots, A_k$, s is the start state for A_1, F is the final states of A_k. Now for the transition function, we keep all the transitions defined in every DFA and add the edges from each final state of A_i to the start state of A_{i+1} on an ϵ for $i \in \{1, \ldots, k-1\}$. This will clearly take linear time w.r.t. the input. □

Definition 6.2. (Minimal automaton): Given a DFA A, $MinFA(A)$ can be constructed by taking the union over all the minimal accepting runs in A, where a minimal accepting run is a run from start state to an accepting state in which no state is repeated.

Definition 6.3. (Minimal words): Given a DFA A, $Min(A)$ is a finite set of all the minimal accepting runs in A, where a minimal accepting run is a run from start state to an accepting state in which no state is repeated. $Min(A)$ is finite since A is a finite state automata and the length of any minimal word is bounded by one less than the number of states in A.

Lemma 6.3. Computing Min(A) takes polynomial time and polynomial space

Proof. We show a procedure to compute Min(A) in Algorithm 1.
Given a DFA $A = (Q, \Sigma, \delta, s, F)$, it takes

$$\approx \mathcal{O}((|\delta| * |Q| * |\Sigma|) + (|Q| * |\Sigma|)) = \mathcal{O}(|\delta| * |Q| * |\Sigma|)$$

to compute its Min which is polynomial to the description of the input automaton.
Note:The length of a word of a minimal automaton $Min(A)$ is strictly less than the number of states in A since no state can repeat. Since from each state there can be atmost $|\Sigma|$ many transitions, the total number of transitions is bounded by $|\Sigma| * |Q|$. □

Algorithm 1 To compute Min(A)

procedure COMPUTE-MIN($A = (Q, \Sigma, \delta, s, F)$) ▷ A is a DFA
    Initialize Queue Q, Linked List *Path*, A rooted Tree T where each node has three data fields $< state, path, word >$, a pointer curr=NULL, Min = $\emptyset$
    Create-Node(node)
    Path.head = s
    node.state = s, node.path = Path
    if s $\in$ F **then**
        node.word = ϵ
    else
        node.word = $\emptyset$
    T.root = node
    ENQUEUE(Q,curr)
    while Q is not empty **do**
        curr = DEQUEUE(Q)
        Path = curr.path
        Word = curr.word
        for each q such that curr.state $\xrightarrow{a}$ q, $\forall a \in \Sigma$ **do**
            if q is not an element of Path **then**
                Create-Node(new-node)
                new-node.state = q
                new-node.path = Path
                new-node.path.add-at-end(q)
                new-node.word = curr.word
                new-node.word.append(a)
                curr.child = node'
                ENQUEUE(Q,q)
    Initialize visited[node]=0 for each node in T, a stack s to empty, Min = $\emptyset$
    s.push(T.root)
    while s is not empty **do** ▷ to get the minimal accepting words
        curr-node = s.pop()
        if curr-node.child is empty **then**
            continue
        if curr-node.state $\in$ F **then**
            Min = Min $\cup$ curr-node.word
        while $\exists$ node n $\in$ T s.t. visited[n]=0 **do**
            if curr-node.child = n **then**
                s.push(n)
    return Min