On the Separability Problem of String Constraints

Parosh Aziz Abdulla
Uppsala University, Sweden
parosh@it.uu.se

Mohamed Faouzi Atig
Uppsala University, Sweden
mohamed_faouzi.atig@it.uu.se

Vrunda Dave
IIT Bombay, India
vrunda@cse.iitb.ac.in

Shankara Narayanan Krishna
IIT Bombay, India
krishnas@cse.iitb.ac.in

Abstract

We address the separability problem for straight-line string constraints. The separability problem for languages of a class C by a class S asks: given two languages A and B in C, does there exist a language I in S separating A and B (i.e., I is a superset of A and disjoint from B)? The separability of string constraints is the same as the fundamental problem of interpolation for string constraints. We first show that regular separability of straight line string constraints is undecidable. Our second result is the decidability of the separability problem for straight-line string constraints by piece-wise testable languages, though the precise complexity is open. In our third result, we consider the positive fragment of piece-wise testable languages as a separator, and obtain an ExpSpace algorithm for the separability of a useful class of straight-line string constraints, and a Pspace-hardness result.

2012 ACM Subject Classification Security and privacy → Logic and verification; Theory of computation → Verification by model checking

Keywords and phrases string constraints, separability, interpolants

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.16

Related Version A full version of the paper is available at `https://arxiv.org/abs/2005.09489`.

1 Introduction

The *string* data type is widely used in almost all modern programming and scripting languages. Many of the well-known security vulnerabilities such as SQL injections and cross-site scripting attacks are often caused by an improper handling of strings. The detection of such vulnerabilities is usually reduced to the satisfiability of a formula which is then solved by SMT solvers (e.g., [39, 40, 46, 30]). Therefore, string constraints solving has received considerable attention in recent years (e.g. [13, 12, 27, 46, 47, 42, 28, 26, 1, 30, 10, 25]) and this has led to the development of many efficient string solvers such as HAMPI [27], Z3-str3 [9], CVC4 [28, 29, 38], S3P [42, 43], Trau [1, 2, 5], SLOTH [25] and OSTRICH [14].

In spite of these advances, most of these tools do not provide any completeness guarantees. The foundational question regarding the decidability of string solving for a large class of string constraints has several challenges to be overcome. A major difficulty is that any reasonably expressive class of string constraints is either undecidable, or has its decidability status open for several years [20, 21, 22]. In fact, the satisfiability problem is undecidable even for the class of string constraints with concatenation (useful to model assignments in the

program) and transduction (useful to model sanitisation and replacement operations) [14]. A direction of research is to find meaningful and expressive subclasses of string constraints for which the satisfiability problem is decidable (e.g., [3, 5, 21, 30, 25, 12]). An interesting subclass, that has been studied extensively, is that of straight-line (SL) string constraints (e.g., [25, 14, 30, 25, 12]). The SL fragment was introduced by Barceló and Lin in [30]. Roughly, an SL constraint models the feasibility of a path of a string-manipulating program that can be generated by symbolic execution. The *satisfiability* of the SL fragment was shown to be EXPSPACE-complete in [30] and forms the basis of many of the tools above [25, 12].

In this paper, we focus on the fundamental problem of *interpolation/separability* for the SL fragment of string constraints. An *interpolant* for a pair of formulas A, B is a formula over their common vocabulary that is implied by A and is inconsistent with B. The Craig-Lyndon interpolation technique is very well-known in mathematical logic. McMillan [32] in his pioneering work, has also recognized interpolation as an efficient method for automated construction of abstractions of systems. Interpolation based algorithms have been developed for a number of problems in program verification [32, 33, 34]. Interpolation procedures have been implemented by many solvers for the theories most commonly used in program verification like linear arithmetic, uninterpreted functions with equality and some combination of such theories. In most of these algorithms, the interpolants were simple. The interpolation technique can also be used to check the unsatisfiability. In fact, the existence of an interpolant for formulas A and B implies the unsatisfiability of $A \wedge B$.

The notion of *separators* in formal language theory is the counterpart of interpolants in logic. The separability problem for languages of a class $\mathcal{C}$ by a class $\mathcal{S}$ asks: given two languages $I, E \in \mathcal{C}$, does there exist a language $S \in \mathcal{S}$ separating I and E? That is, $I \subseteq S$ and $S \cap E = \emptyset$. The language S is called the separator of I, E. Separability is a classical problem of fundamental interest in theoretical computer science, and has recently received a lot of attention. For instance, regular separability has been studied for one-counter automata [16], Parikh automata [15], and well-structured transition systems [17]. In the following, we use the terms interpolant or separator of two SL string constraints to mean the same thing, since the solutions of a string constraint can be interpreted as a language.

In this paper, we first show that any string constraint ϕ can be written as the conjunction of two SL string constraints A and B. Therefore, the interpolation problem for the pair A and B can be used to check the unsatisfiability of the string constraint ϕ. (Recall that the satisfiability problem for general string constraints is undecidable [14].)

Then, we consider the regular separability problem for SL string constraints. We show that this problem is undecidable (Theorem 2) by a reduction from the halting problem of Turing Machines. The main technical difficulty here is to ensure that the encoding of a sequence of configurations of a Turing machine results in SL string constraints.

Due to this undecidability, we focus on the separability problem of SL string constraints by piece-wise testable languages (PTL). A PTL is a finite Boolean combination of special regular languages called *piece languages* of the form $\Sigma^* a_1 \Sigma^* a_2 \dots \Sigma^* a_n \Sigma^*$, where all $a_j \in \Sigma$. PTL is a well-studied class of languages in the context of the separability problem (e.g. [36, 18, 19]). Furthermore, among the various separator classes considered in the literature, the class of piecewise testable languages (PTL) seems to be the most tractable: PTL-separability of regular languages is in PTIME [36, 18]. To decide the PTL-separability of SL string constraints, we first encode the solutions of an SL string constraint as the language of an Ordered Multi-Pushdown Automaton (OMPA) (Section 4.1). Then, we show that the PTL-separability of SL constraints can be reduced to the PTL-separability of OMPAs. To show the decidability of the latter problem, we first prove that the language of an OMPA: (1)

```
str old = real_escape_string(oldIn);
str new1 = real_escape_string(newIn1);
str new2 = real_escape_string(newIn2);
str pass = database_query("SELECT password FROM users WHERE userID=" userID);
if (old == pass AND new1 == new2 AND new1 != old )
    if (newIn1==newIn2 AND newIn1 != oldIn)
       str query = "UPDATE users SET password=" + new1 + "WHERE userID=" + userID;
       database_query(query);
```

Figure 1 A pseudo PHP code for changing password.

is a full trio [23] and (2) has a semilinear Parikh image. Using (1), we obtain the equivalence of the PTL separability problem and the *diagonal problem* for OMPAs from [19], where the equivalence has been shown to hold for full trios. Next, the decidability of PTL-separability problem for OMPAs is obtained from the decidability of the diagonal problem for OMPAs: the latter is obtained using (2) and [19] where the decidability of the diagonal problem has been shown for languages having a semilinear Parikh image. As a corollary of these results, we obtain the decidability of the PTL-separability problem for SL string constraints and OMPAs; however the exact complexity is still an open question. In fact, it is an open problem in the case of OMPAs with one stack (i.e., Context-Free Languages (CFLs)) [19].

Given the complexity question, we propose the class of positive piecewise testable languages (PosPTL) as separators. PosPTL is obtained as a negation-free Boolean combination of piece languages. As a first result (Theorem 12) we show that deciding PosPTL-separability for any language class has a very elegant proof: it suffices to check if the upward (downward) closure of one of the languages is disjoint from the other language. Using this result, we prove the PSPACE-completeness of the PosPTL-separability for CFLs, thereby progressing on the complexity front with respect to a problem which is open in the case of PTL-separability for CFLs. Then, we focus on a class of SL string constraints where the variables used in outputs of the transducers are independent of each other. This class contains SL string constraints with functional transducers (computing partial functions, by associating at most one output with each input). We prove the decidability and EXPSPACE membership for the PosPTL-separability of this class by first encoding the solutions of string constraints as outputs of two way transducers (2NFT), and then proving the decidability of PosPTL-separability for 2NFT.

Due to lack of space, missing proofs of all results can be found in the full version [4].

As a practical motivation of PosPTL (and PTL), consider the following pseudo-PHP code in Figure 1 obtained as a variation of the code at [31]. In this code, a user is prompted to change his password by entering the new password twice. In this code, database_query represents the function which executes the query given as its parameter. We use '+' operator as concatenation of strings and variables (variables are represented using blue color). The user inputs the old password oldIn and the new password twice : newIn1 and newIn2. These are sanitized and assigned to old, new1 and new2 respectively. The old sanitized password is compared with the value pass from the database to authenticate the user, and also with the new sanitized password to check that a different password has been chosen, and finally, the sanitized new passwords entered twice are checked to be the same.

Sanitization ensures that there are no SQL injections. To ensure the absence of SQL attacks, we require that the query query does not belong to a regular language Bad of bad patterns over some finite alphabet Σ (i.e., the program is safe). This safety condition can be expressed as the unsatisfiability of the following formula φ given by

$\texttt{new1} = \texttt{T}(\texttt{newIn1}) \wedge \texttt{new2} = \texttt{T}(\texttt{newIn2}) \wedge \texttt{old} = \texttt{T}(\texttt{oldIn}) \wedge \texttt{new1} = \texttt{new2} \wedge \texttt{pass} = \texttt{old} \wedge$
$\texttt{old} \neq \texttt{new1} \wedge \texttt{newIn1} = \texttt{newIn2} \wedge \texttt{query} = \texttt{u} \cdot \texttt{new1} \cdot \texttt{v} \cdot \texttt{userID} \wedge \texttt{query} \in \texttt{Bad}.$

Note that the check $\texttt{new1} = \texttt{new2}$ has to be done by the server to ensure the sanitized new passwords entered twice are same; however, the check $\texttt{newIn1} = \texttt{newIn2}$ is not redundant, since it can happen that post sanitization, the passwords may agree, but not before. The sanitization on lines 1, 2 and 3 is represented by the transducer $\texttt{T}$ and $\texttt{u}, \texttt{v}$ are the constant strings from line 7. It is easy to see that the program given here is safe iff the formula φ is unsatisfiable. Observe that the formula φ is not in the straight line fragment [30] since variable $\texttt{new1}$ has two assignments. Further, it also has a non-benign chain making it fall out of the fragment of string programs handled in [5]. However the formula φ can be rewritten as a conjunction of the two formula φ_1 and φ_2 in straight-line form where

$\varphi_1 : \texttt{new1} = \texttt{T}(\texttt{newIn1}) \wedge \texttt{old} = \texttt{T}(\texttt{oldIn}) \wedge \texttt{pass} = \texttt{old} \wedge \texttt{query} = \texttt{u} \cdot \texttt{new1} \cdot \texttt{v} \cdot \texttt{userID} \wedge \texttt{query} \in \texttt{Bad}$

$\varphi_2 : \texttt{new2} = \texttt{T}(\texttt{newIn2}) \wedge \texttt{new1} = \texttt{new2} \wedge \texttt{old} \neq \texttt{new1} \wedge \texttt{newIn1} = \texttt{newIn2}.$

It is easy to see that the program is safe if solution sets of φ_1 and φ_2 are separable by some PosPTL set, in that case, we can say that there is no solution which is common to φ_1 and φ_2 and thus $\varphi = \varphi_1 \wedge \varphi_2$ is unsatisfiable.

Related work. The satisfiability problem for string constraints is an active research area and there is a lot of progress in the last decade (e.g., [37, 27, 30, 12, 14, 5, 21, 22, 3, 45]). An interpolation based semi-decision procedure for string constraints has been proposed in [3]. As far as we know, this is the first time the separability problem has been studied in the context of string constraints.

2 Preliminaries

Notations. Let $[i, j]$ denote the set $\{i, \ldots, j\}$ for $i, j \in \mathbb{N}$. Let Σ be a finite alphabet. Σ^* denotes the set of all finite words over Σ and Σ^+ denotes $\Sigma^* \backslash \{\epsilon\}$ where ϵ is the empty word. We denote $\Sigma \cup \{\epsilon\}$ by Σ_ϵ. Let $u \in \Sigma^*$. We use u^R to denote the reverse of u. The length of the word u is denoted $|u|$ and the i^{th} symbol of u by $u[i]$. Given two words $u \in \Sigma^*$ and $v \in \Sigma^*$, we say that u is a subword of v (denoted $u \preceq v$) if there is a mapping $h : [1, |u|] \mapsto [1, |v|]$ such that (1) $u[i] = v[h(i)]$ for all $i \in [1, |u|]$, and (2) $h(i) < h(j)$ for all $i < j$.

(Multi-tape)-Automata. A *Finite State Automaton* (FSA) over an alphabet Σ is a tuple $\mathcal{A} = (Q, \Sigma, \delta, I, F)$, where Q is a finite set of *states*, $\delta \subseteq Q \times \Sigma_\epsilon \times Q$ is a set of *transitions*, and $I \subseteq Q$ (resp. $F \subseteq Q$) are the *initial* (resp. *accepting*) states. $\mathcal{A}$ accepts a word w iff there is a sequence $q_0 a_1 q_1 a_2 \cdots a_n q_n$ such that $(q_{i-1}, a_i, q_i) \in \delta$ for all $1 \leq i \leq n$, $q_0 \in I$, $q_n \in F$, and $w = a_1 \cdot \cdots \cdot a_n$. The *language* of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set all accepted words.

Given $n \in \mathbb{N}$, a *n-tape automaton* $\mathcal{T}$ is an automaton over the alphabet $(\Sigma_\epsilon)^n$. It *recognizes* the relation $\mathcal{R}(\mathcal{T}) \subseteq (\Sigma^*)^n$ that contains the n-tuple of words $(w_1, w_2, \ldots, w_n)$ for which there is a word $(a_{(1,1)}, a_{(2,1)}, \ldots, a_{(n,1)}) \cdots (a_{(1,m)}, a_{(2,m)}, \ldots, a_{(n,m)}) \in \mathcal{L}(\mathcal{T})$ with $w_i = a_{(i,1)} \cdots \cdots a_{(i,m)}$ for all $i \in \{1, \ldots, n\}$. A *transducer* is a 2-tape automaton.

Well-quasi orders. Given a (possibly infinite set) C, a quasi-order on C is a reflexive and transitive relation $\sqsubseteq \subseteq C \times C$. An infinite sequence $c_1, c_2, \ldots$ in C is said to be saturating if there exists indices $i < j$ s.t. $c_i \sqsubseteq c_j$. A quasi-order $\sqsubseteq$ is said to be a well-quasi order (wqo) on C if every infinite sequence in C is saturating. Observe that the subword ordering $\preceq$ between words u, v over a finite alphabet Σ is well-known to be a wqo on Σ^* [24].

Upward and Downward Closure: Given a wqo $\sqsubseteq$ on a set C, a set $U \subseteq C$ is said to be upward closed if for every $a \in U$ and $b \in C$, with $a \sqsubseteq b$, we have $b \in U$. The upward closure of a set $U \subseteq C$ is defined as $U{\uparrow} = \{b \in C \mid \exists a \in U, a \sqsubseteq b\}$. It is known that every

upward closed set U can be characterized by a finite *minor*. A minor $M \subseteq U$ is s.t. (i) for each $a \in U$, there is a $b \in M$ s.t. $b \sqsubseteq a$, and (ii) for all $a, b \in M$ s.t. $a \preceq b$, we have $a = b$. For an upward closed set U, let min be the function that returns the minor of U. Downward closures are defined analogously. The downward closure of a set $D \subseteq C$ is defined as $D{\downarrow} = \{b \in C \mid \exists a \in D, b \sqsubseteq a\}$. The notion of subword relation and thus upward and downward closures naturally extends to n-tuples of words. The subword relation here is component wise i.e. $(u_1, \ldots, u_n) \preceq_n (v_1, \ldots, v_n)$ iff $u_i \preceq v_i$ for all $i \in [1, n]$.

String Constraints. An atomic string constraint φ over an alphabet Σ and a set of string variables $\mathcal{X}$ is either: (1) a *membership constraint* of the form $x \in \mathcal{L}(\mathcal{A})$ where $x \in \mathcal{X}$ and $\mathcal{A}$ is a FSA (i.e., the evaluation of x is in the language of a FSA $\mathcal{A}$ over Σ), or (2) a *relational constraint* of the form $(t', t) \in \mathcal{R}(\mathcal{T})$ where t and t' are string terms (i.e., concatenation of variables in $\mathcal{X}$) and $\mathcal{T}$ is a transducer over Σ, and t and t' are related by a relation recognised by the transducer $\mathcal{T}$. $(t', t) \in \mathcal{R}(\mathcal{T})$ can also be written as $t' = \mathcal{T}(t)$, that is, $\mathcal{T}$ produces t' as the output on input t. For a given term t, $|t|$ denotes the number of variables appearing in t.

A string constraint Ψ is a conjunction of atomic string constraints. We define the semantics of string constraints using a mapping η, called *evaluation*, that assigns for each variable a word over Σ. The evaluation η can be extended in the straightforward manner to string terms as follows $\eta(t_1 \cdot t_2) = \eta(t_1) \cdot \eta(t_2)$. We extend also η to atomic constraints as follows: (1) $\eta(x \in \mathcal{L}(\mathcal{A})) = \top$ iff $\eta(x) \in \mathcal{L}(\mathcal{A})$, and (2) $\eta((t, t') \in \mathcal{R}(\mathcal{T})) = \top$ iff $(\eta(t), \eta(t')) \in \mathcal{R}(\mathcal{T})$. The truth value of Ψ for an evaluation η is defined in the standard manner. If $\eta(\Psi) = \top$ then η is a *solution* of Ψ, written $\eta \models \Psi$. The formula Ψ is *satisfiable* iff it has a solution.

A string constraint is said to be *Straight Line*[1] (SL) if it can be rewritten as $\Psi' \wedge \bigwedge_{i=1}^{k} \varphi_i$ where Ψ' is a conjunction of membership constraints, and $\varphi_1, \ldots, \varphi_k$ are relational constraints such that (1) there is a sequence of different string variables $x_1, x_2, \ldots, x_n$ with $n \geq k$, and (2) φ_i is of the form $(x_i, t_i) \in \mathcal{R}(\mathcal{T}_i)$ such that if a variable x_j is appearing in t_i then $j > i$. A string constraint in the SL form is called an SL formula. Observe that any string formula can be rewritten as a conjunction of two SL formulas (by using extra-variables).

▶ **Lemma 1.** *Given a string constraint Ψ, it is possible to construct two SL string constraints Ψ_1 and Ψ_2 such that Ψ is satisfiable iff $\Psi_1 \wedge \Psi_2$ is satisfiable.*

Let Ψ be a string constraint and $x_1, \ldots, x_n$ be the set of variables appearing in Ψ. We use $\mathcal{L}(\Psi)$ to denote the language of Ψ which consists of the set of n-tuple of words $(u_1, \ldots, u_n)$ such that there is an evaluation η with $\eta(\Psi) = \top$ and $\eta(x_i) = u_i$ for all $i \in [1, n]$.

The Separability Problem. Given two classes of languages $\mathcal{C}$ and $\mathcal{S}$, the separability problem for $\mathcal{C}$ by the separator class $\mathcal{S}$ is defined as follows: Given two languages I and E from the class $\mathcal{C}$, does there exist a separator $S \in \mathcal{S}$ such that $I \subseteq S$ and $E \cap S = \emptyset$.

3 Regular Separability of String Constraints.

Let Σ be an alphabet and k, n be two natural numbers. A set R of n-tuples of words over Σ is said to be regular (REG) iff there is a sequence of finite-state automata $\mathcal{A}_{(i,1)}, \ldots, \mathcal{A}_{(i,n)}$ for every $i \in [1, k]$ such that $R = \bigcup_{i=1}^{k} [\mathcal{L}(\mathcal{A}_{(i,1)}) \times \cdots \times \mathcal{L}(\mathcal{A}_{(i,n)})]$. The REG separability

[1] In [30], the authors consider Boolean combinations of membership constraints. Our results can be extended to handle such formulas. In [30], they also consider constraints of the form $x = t$. Such constraints can be encoded using our relational constraints.

problem for string constraints consists in checking for two given string constraints Ψ and Ψ' over the string variables $x_1, \ldots, x_n$ whether there is a regular set $R \subseteq (\Sigma^*)^n$ such that $\mathcal{L}(\Psi) \subseteq R$ and $R \cap \mathcal{L}(\Psi') = \emptyset$.

The regular separability problem is undecidable in general. This can be seen as an immediate corollary of the fact that the satisfiability problem of string constraints is undecidable [35, 14] even for a simple formula of the form $(x, x) \in \mathcal{R}(\mathcal{T})$ where $\mathcal{T}$ is a transducer and x is a string variable. To see why, consider Ψ to be $(x, x) \in \mathcal{R}(\mathcal{T})$ and Ψ' such that $\mathcal{L}(\Psi') = \Sigma^*$. It is easy to see that Ψ' and Ψ are separable by a regular set iff Ψ is unsatisfiable. In the following, we show a stronger result, namely that this undecidability still holds even for REG separability between two SL formulas.

▶ **Theorem 2.** *The* REG *separability problem is undecidable even for SL string constraints.*

4 PTL-Separability of String Constraints

Given the undecidability of REG separability, we focus on the separability problem using piece-wise testable languages (PTL). We show that the problem is in general undecidable and then we show its decidability in the case of SL formulas. The undecidability proof is exactly the same as in the case of the REG separability (since Σ^* is a PTL) while the decidability proof is done by reduction to its corresponding problem for the class of Ordered Multi Pushdown Automata (OMPA) [7, 11] (which we show its decidability). In the rest of this section, we first recall the definition of PTL and extend it to n-tuples of words. Then, we define the class of OMPAs and show the decidability of its separability problem by PTL. Finally, we show the decidability of the separability problem for SL formulas by PTL.

Piece-wise testable languages. Let Σ be an alphabet. A piece-language is a regular language of the form $\Sigma^* a_1 \Sigma^* a_2 \Sigma^* \ldots \Sigma^* a_k \Sigma^*$ where $a_1, a_2, \ldots, a_k \in \Sigma$. The class of piecewise testable languages (PTL) is defined as a finite Boolean combination of piece languages [41]. We can define PTL for an n-tuple alphabet with $n \in \mathbb{N}$, as follows: The class of PTL over n-tuple words (denoted n-PTL) is defined as the finite Boolean combination of languages of the form $(\Sigma^*)^n \mathbf{v}_1 (\Sigma^*)^n \cdots (\Sigma^*)^n \mathbf{v}_k (\Sigma^*)^n$ where $\mathbf{v}_i \in (\Sigma_\epsilon)^n$ for all $i \in [1, k]$.

Ordered Multi Pushdown Automata. Let Σ be a finite alphabet and $n \geq 1$ a natural number. Ordered multi-pushdown automata extend the model of pushdown automata with multiple stacks. An n-Ordered Multi Pushdown Automaton (OMPA or n-OMPA) is a tuple $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, Q_0, F)$ where (1) Q, Q_0 and F are finite sets of states, initial states and final states, respectively, (2) Γ is the stack alphabet and it contains the special symbol $\perp$, and (3) δ is the transition relation. OMPA are restricted in a sense that pop operations are only allowed from the first non-empty stack. A transition in δ is of the form $(q, \perp, \ldots, \perp, A_j, \epsilon, \ldots, \epsilon) \to^a (q', \gamma_1, \ldots, \gamma_n)$ where $A_j \in \Gamma_\epsilon$ represents the symbol that will be popped from the stack j on reading the input symbol $a \in \Sigma_\epsilon$, and $\gamma_i \in \Gamma^*$ represents the sequence of symbols which is going to be pushed on the stack i. The condition that $A_1 = \ldots = A_{j-1} = \perp$ (resp. $A_{j+1} = \ldots = A_n = \epsilon$) corresponds to the fact that the stacks $1, \ldots, j-1$ (resp. $j+1, \ldots, n$) are required to be empty (resp. inaccessible).

A configuration of $\mathcal{A}$ is of the form $(q, w, \alpha_1, \ldots, \alpha_n)$ where $q \in Q$, $w \in \Sigma^*$ and $\alpha_1, \ldots, \alpha_n \in (\Gamma \setminus \{\perp\})^* \cdot \{\perp\}$. The transition relation $\to$ between the set of configurations of $\mathcal{A}$ is defined as follows: Given two configurations $(q, w, \alpha_1, \ldots, \alpha_n)$ and $(q', w', \alpha'_1, \ldots, \alpha'_n)$, we have $(q, w, \alpha_1, \ldots, \alpha_n) \to (q', w', \alpha'_1, \ldots, \alpha'_n)$ iff there is a transition $(q, A_1, \ldots, A_n) \to^a (q', \gamma_1, \ldots, \gamma_n) \in \delta$ such that $w = aw'$ and $\alpha'_i = \gamma_i u_i$ where $\alpha_i = A_i u_i$ for all $i \in [1, n]$. We

use $\rightarrow^*$ to denote the transitive and reflexive closure of $\rightarrow$. A word $w \in \Sigma^*$ is accepted by $\mathcal{A}$ if there exists a sequence of configurations $c_1, \ldots, c_m$ such that: (1) c_1 is of the form $(q_0, w, \bot, \ldots, \bot)$, with $q_0 \in Q_0$, (2) c_m is of the form $(q_f, \epsilon, \bot, \ldots, \bot)$, with $q_f \in F$, and (3) $c_i \rightarrow c_{i+1}$ for all $i \in [1, m-1]$. The language of $\mathcal{A}$ (denoted by $\mathcal{L}(\mathcal{A})$) is defined as the set of words accepted by $\mathcal{A}$. The languages accepted by OMPA are referred to as OMPL.

In the following, we show that the separability problem for OMPL by PTL is decidable. As a first step, we show that the class of OMPL forms a *full trio* [23, 19]. We first recall the definition of a full-trio. Let L be a language over an alphabet A, and let $B \subseteq A$. The B-projection of a word $w \in A^*$ is the longest scattered subword containing only symbols from B. For example, if $A = \{a, b, c\}$, $B = \{b, c\}$, then the B-projection of $w = ababac$ is bbc. The B-upward closure of L is the set of all words that can be obtained by taking a word in L and padding it with symbols from B. For example, if $L = \{w\}$ for w as above, then the B-upward closure of L is the set $B^*aB^*bB^*aB^*bB^*aB^*cB^*$. A class of languages $\mathcal{C}$ is a full trio if it is effectively closed under (1) B-projection for every finite alphabet B, (2) B-upward closure for every finite alphabet B, and (3) intersection with regular languages.

▶ **Lemma 3.** *The class of OMPLs forms a full trio.*

To connect the PTL separability problem of SL string constraints to that of OMPL, we first use lemma 4. Lemma 4 states that the PTL separability problem for OMPL is equivalent to the *diagonal problem* for OMPL. We recall the diagonal problem [19]. Fix a class of languages $\mathcal{C}$ as above and a language $L \in \mathcal{C}$ over alphabet $\Sigma = \{a_1, \ldots, a_n\}$. Assume an ordering $a_1 < \cdots < a_n$ on Σ. For $a \in \Sigma$ and $w \in L$, let $\#_a(w)$ denote the number of occurrences of a in w. The *Parikh image* of w is the n-tuple $(\#_{a_1}(w), \ldots, \#_{a_n}(w))$. The *Parikh image* of L is the set of all Parikh images of words in L. An n-tuple $(m_1, \ldots, m_n) \in \mathbb{N}^n$ is dominated by another n-tuple $(d_1, \ldots, d_n)$ iff $m_i \leq d_i$ for all $1 \leq i \leq n$. The *diagonal problem* for $\mathcal{C}$ is the decision problem, which, given as input, a language L from $\mathcal{C}$ asks whether each n-tuple $(m, \ldots, m) \in \mathbb{N}^n$ is dominated by some Parikh image of L.

▶ **Lemma 4.** *The* PTL*-separability and diagonal problems are equivalent for OMPLs.*

Proof. This equivalence has been shown for full trios in [19] and by Lemma 3, OMPLs form a full trio. ◀

▶ **Lemma 5.** *Each language L in OMPL has a semilinear Parikh image and its representation can be effectively computable.*

▶ **Theorem 6.** *Given two OMPAs $\mathcal{A}_1$ and $\mathcal{A}_2$, checking whether there is a* PTL *L such that $\mathcal{L}(\mathcal{A}_1) \subseteq L$ and $L \cap \mathcal{L}(\mathcal{A}_2) = \emptyset$ is decidable.*

Proof. The proof follows from Lemmas 5, 4 and [19], from where we know that the diagonal problem is decidable for classes of languages having effectively semilinear Parikh images. ◀

▶ Remark 7. For the case of 1−OMPA, the PTL separability problem is already known to be decidable [19] but its complexity is still an open problem.

4.1 From SL formula to OMPA

In the following, we show that the n-PTL separability problem for SL formulas can be reduced to the PTL separability problem for OMPLs. To that aim, we proceed as follows: First, we show how to encode an n-tuple of words ($\in (\Sigma^*)^n$) as a word over $(\Sigma \cup \{\#\})^*$. Then, we show how to encode the set of solutions of an atomic relational constraint $(x, t) \in \mathcal{R}(\mathcal{T})$ using

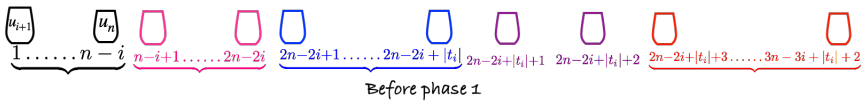
the stacks of an OMPA. Finally, we construct an OMPA that accepts exactly the language of a given SL formula Ψ. This construction will make use of the constructed OMPAs that encode the set of atomic relational constraints appearing in Ψ. Let Σ be an alphabet.

Encoding an n-tuple of words. Let n be a natural number. We assume w.l.o.g. that the special symbol $\#$ does not belong to Σ. We define the function `Encode` that maps any n-tuple word $\mathbf{w} = (w_1, \ldots, w_n) \in (\Sigma^*)^n$ to the word $w_1 \# w_2 \# \cdots \# w_n$.

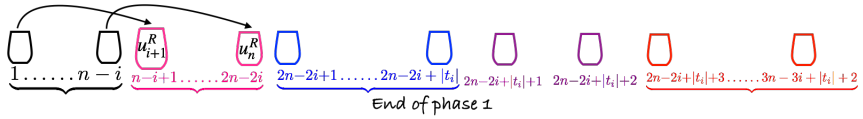
From SL atomic relational constraints to OMPAs. Let $x_1, x_2, \ldots, x_n$ be a sequence of string variables. Let P_i be a relational constraint of the form $(x_i, t_i) \in \mathcal{R}(\mathcal{T}_i)$ such that if a variable x_j is appearing in the term t_i, then $j > i$. In the following, we show that we can construct an OMPA $\mathcal{A}_i$ with $(3n+|t_i|+2-3i)$ stacks such that if $\mathcal{A}_i$ starts with a configuration where the first $(n-i)$ stacks contain, respectively, the evaluations $\eta(x_{i+1}), \ldots, \eta(x_n)$ (and all the other stacks are empty), then it can compute an evaluation $\eta(x_i)$ of the variable x_i such that: (1) $(\eta(x_i), \eta(t_i)) \in \mathcal{R}(\mathcal{T}_i)$ and the evaluations $\eta(x_i), \ldots, \eta(x_n)$ are stored in the last $n-i+1$ stacks of $\mathcal{A}_i$. Such an OMPA $\mathcal{A}_i$ will be used as a gadget when constructing the OMPA $\mathcal{A}$ that accepts exactly the language of a given SL formula Ψ.

▶ **Lemma 8.** *We can construct an OMPA $\mathcal{A}_i = (Q_i, \Sigma, \{\bot\} \cup \Sigma, \delta_i, \{q_i^{init}\}, \{q_i^{final}\})$ with $(3n + |t_i| + 2 - 3i)$-stacks such that for every $u_i, \ldots, u_n \in \Sigma^*$, we have $(q_i^{init}, \epsilon, u_{i+1}\bot, \ldots, u_n\bot, \bot, \ldots, \bot) \to^* (q_i^{final}, \epsilon, \bot, \ldots, \bot, u_i\bot, u_{i+1}\bot, \ldots, u_n\bot)$ iff $(\eta(x_i), \eta(t_i)) \in \mathcal{R}(\mathcal{T}_i)$ with $\eta(x_j) = u_j$ for all $j \in [i, n]$.*

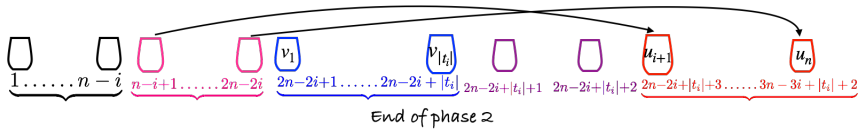
Proof. In the proof, we omit the input ϵ from the OMPA configurations, and only write the state, and stack contents. Let us assume that the string term t_i is of the form $y_1 y_2 \cdots y_{|t_i|}$. Observe that $y_j \in \{x_{i+1}, \ldots, x_n\}$. The OMPA $\mathcal{A}_i$ proceeds in phases starting from the configuration $(q_i^{init}, u_{i+1}\bot, \ldots, u_n\bot, \bot, \ldots, \bot)$. To begin, stacks 1 to $n-i$ contain $u_{i+1}, \ldots, u_n$, the evaluations of $x_{i+1}, \ldots, x_n$, and all other stacks are empty. The computation proceeds in 4 phases. The stacks indexed $1, \ldots, n-i$ and $n-i+1, \ldots, 2n-2i$ will be used in the first phase below. The second phase uses stacks indexed $n-i+1, \ldots, 2n-2i$ and $2n-2i+1, \ldots, 2n-2i+|t_i|$ along with the last $n-i$ stacks indexed $2n-2i+|t_i|+3$ to $3n-3i+|t_i|+2$. In the third phase, stacks indexed $2n-2i+1, \ldots, 2n-2i+|t_i|, 2n-2i+|t_i|+1$ are used. In the last phase, stacks indexed $2n-2i+|t_i|+1$ and $2n-2i+|t_i|+2$ are used. At the end of the 4 phases, stacks indexed $2n-2i+|t_i|+2, \ldots, 3n-3i+|t_i|+2$ hold the evaluations of $x_i, x_{i+1}, \ldots, x_n$, and all other stacks are empty.



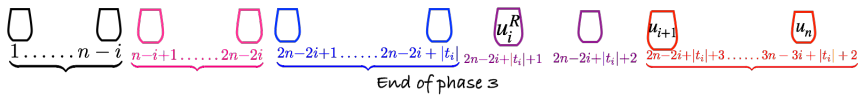
Phase 1. The OMPA $\mathcal{A}_i$ pops the symbols, one by one, from the first $(n-i)$-stacks $1, \ldots, n-i$ and pushes them into the stacks from index $(n-i+1)$ to $(2n-2i)$, respectively. At the end of this phase, the new configuration of the OMPA $\mathcal{A}_i$ is $(q_i^{init}, \bot, \ldots, \bot, u_{i+1}^R\bot, \ldots, u_n^R\bot, \bot, \ldots, \bot)$. That is, stacks $n-i+1, \ldots, 2n-2i$ have $u_{i+1}^R, \ldots, u_n^R$, while all other stacks are empty.



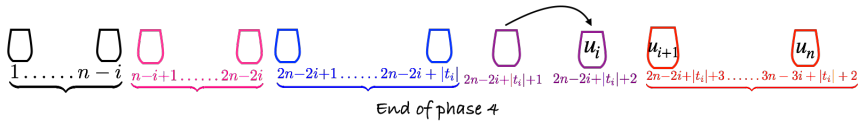
Phase 2. We do two things. (1) the contents of the $n-i$ stacks $n-i+1,\ldots,2n-2i$ are moved (in reverse) into the $n-i$ stacks $2n-2i+|t_i|+3,\ldots,3n-3i+|t_i|+2$. This results in the stacks $2n-2i+|t_i|+3,\ldots,3n-3i+|t_i|+2$ containing $u_{i+1},\ldots,u_n$. (2) If y_j appearing in t_i is the variable $x_{i+\ell}$, then the content of stack $n-i+\ell$ (with $n-i+1\leq n-i+\ell\leq 2n-2i$) is also moved (in reverse) to stack $2n-2i+j$, $1\leq j\leq |t_i|$. This results in stack $2n-2i+j$ containing $u_{i+\ell}$. Thus, at the end of (1), (2), the stacks $n-i+1,\ldots,2n-2i$ are empty, the stack $2n-2i+|t_i|+\ell+2$ contains $u_{i+\ell}$, the evaluation of $x_{i+\ell}$ for $\ell\geq 1$, while stack $2n-2i+k$ for $1\leq k\leq |t_i|$ contains u_{i+m} if $y_k=x_{i+m}$. The two stacks $2n-2i+|t_i|+1$ and $2n-2i+|t_i|+2$ are empty at the end of this phase. Stack contents of $2n-2i+k$, $1\leq k\leq |t_i|$ are referred to as v_k in the figure.



Phase 3. The OMPA $\mathcal{A}_i$ mimics the transducer $\mathcal{T}_i$. The current state of $\mathcal{A}_i$ is the same as the current state of the simulated transducer. Each transition of $\mathcal{T}_i$ of the form $(q,(a,b),q')$ is simulated by (1) moving the state of $\mathcal{A}_i$ from q to q', (2) pushing the symbol a into the stack $(2n-2i+|t_i|+1)$, and (3) popping the symbol b from the first non-empty stack having an index between $2n-2i+1$ to $2n-2i+|t_i|$. Recall that the stacks $2n-2i+1$ to $2n-2i+|t_i|$ contain the evaluations of $y_1,\ldots,y_{|t_i|}$, for $(\eta(x_i),\eta(y_1).\eta(y_2).\ldots\eta(y_{|t_i|}))\in\mathcal{R}(\mathcal{T}_i)$. When the current state of $\mathcal{A}_i$ is in a final state of $\mathcal{T}_i$ and the stacks from index $2n-2i+1$ to $2n-2i+|t_i|$ are empty, then we know that $\eta(y_1)\ldots\eta(y_{|t_i|})$ is indeed related by $\mathcal{T}_i$ on $\eta(x_i)$. Then $\mathcal{A}_i$ changes its state to q_i^{final}. Observe that, in case $\eta(y_1)\ldots\eta(y_{|t_i|})$ does not belong to the domain of $\mathcal{T}_i$, then the outgoing transition is not defined.



The Last Phase. At the end of the third phase, the current configuration of $\mathcal{A}_i$ is $(q_i^{final},\bot,\ldots,\bot,u_i^R,\bot,u_{i+1}\bot,\ldots,u_n\bot)$ such that $(u_i,v_1\cdots v_{|t_i|})\in\mathcal{R}(\mathcal{T}_i)$: that is, the last $n-i$ stacks $2n-2i+|t_i|+3,\ldots,3n-3i+|t_i|+2$ contain $u_{i+1},\ldots,u_n$, and stack $2n-2i+|t_i|+1$ contains the reverse of u_i. Then, $\mathcal{A}_i$ pops, one-by-one, the symbols from the $(2n-2i+|t_i|+1)$-th stack and pushes them, in the reverse order, into the stack $(2n-2i+|t_i|+2)$. Thus, the new configuration of $\mathcal{A}_i$ is of the form $(q_i^{final},\bot,\ldots,\bot,u_i\bot,u_{i+1}\bot,\ldots,u_n\bot)$ such that $(u_i,v_1\cdots v_{|t_i|})\in\mathcal{R}(\mathcal{T}_i)$ where $v_j=u_\ell$ if $y_j=x_\ell$ for $1\leq j\leq |t_i|$.



◀

From SL formula to OMPAs. In the following, we first construct an OMPA that accepts the encoding of the set of solutions of an SL formula.

▶ **Lemma 9.** *Given an SL formula Ψ, with $x_1,\ldots,x_n$ as its set of variables, it is possible to construct an OMPA $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A})=\texttt{Encode}(\mathcal{L}(\Psi))$.*

Proof. Let us assume that Ψ is of the form $\bigwedge_{i=1}^{n} x_i \in \mathcal{L}(\mathcal{A}_i) \wedge \bigwedge_{i=1}^{k} \varphi_i$ where $\varphi_1, \ldots, \varphi_k$ are relational constraints such that φ_i is of the form $(x_i, t_i) \in \mathcal{R}(\mathcal{T}_i)$. The OMPA $\mathcal{A}$ will have $(n - k + \sum_{i=1}^{k}(2n - 2i + 2 + |t_i|))$ stacks. $\mathcal{A}$ first guesses an evaluation for the variables $x_{k+1}, \ldots, x_n$ in the first $n - k$ stacks and then starts simulating the OMPA $\mathcal{A}_k$ (see Lemma 8 for the definition of $\mathcal{A}_k$) in order to compute a possible evaluation of the variable x_k such that the relational constraint $(x_k, t_k) \in \mathcal{R}(\mathcal{T}_k)$ holds for that evaluation. After this step, the stacks from index $(2n-2k+|t_k|+2)$ to $(3n-3k+|t_k|+2)$ contain the evaluation of the string variables $x_k, \ldots, x_n$, and all remaining stacks are empty. Now $\mathcal{A}$ can start the simulation of the OMPA $\mathcal{A}_{k-1}$ (Lemma 8) in order to compute a possible evaluation of the variable x_{k-1} such that $(x_k, t_k) \in \mathcal{R}(\mathcal{T}_k) \wedge (x_{k-1}, t_{k-1}) \in \mathcal{R}(\mathcal{T}_{k-1})$ holds for that evaluation. At the start of the simulation of $\mathcal{A}_{k-1}$ by $\mathcal{A}$, the $n - k + 1$ stacks (indexed $(2n - 2k + |t_k| + 2)$ to $(3n-3k+|t_k|+2)$) contain the evaluations of $x_k, \ldots, x_n$, and the next $2n-2(k-1)+|t_{k-1}|+2$ stacks are used to simulate phases 2-4 of $\mathcal{A}_{k-1}$. At the end of this, the $n - k + 2$ stacks backwards from the stack indexed $(3n - 3k + |t_k| + 2) + 2n - 2(k - 1) + |t_{k-1}| + 2$ contain the evaluations of $x_{k-1}, \ldots, x_n$. Now, $\mathcal{A}$ simulates $\mathcal{A}_{k-2}, \ldots, \mathcal{A}_n$ in the same way. At the end of this simulation phase, the last n-stacks of $\mathcal{A}$ contain an evaluation of the string variables $x_1, \ldots, x_n$ that satisfies $\bigwedge_{i=1}^{k} \varphi_i$. Let us assume that the current configuration of $\mathcal{A}$ at the end of this is of the form $(q^{final}, \bot, \ldots, \bot, u_1\bot, u_2\bot, \ldots, u_n\bot)$. Then, $\mathcal{A}$ starts popping, one-by-one, from the n-th stack from the last and outputs the read stack symbol $\in \Sigma$ while ensuring that the evaluation u_1 of x_1 belongs to $\mathcal{L}(\mathcal{A}_1)$. When the n-th stack from the last is empty, $\mathcal{A}$ outputs the special symbol $\#$. Then, $\mathcal{A}$ does the same for the i-th stack from last, with $i \in [1, n-1]$, which contains the evaluation of x_{i+1}. If $\mathcal{A}$ succeeds to empty all stacks, then this means that the evaluation η which associates to the variable x_i, the word u_i for all $i \in [1, n]$ satisfies $\bigwedge_{i=1}^{n} x_i \in \mathcal{L}(\mathcal{A}_i)$. Hence, $u_1\#u_2\#\cdots\#u_n$ is accepted by $\mathcal{A}$ iff $\eta \models \Psi$. ◀

The following lemma shows that the PTL-separability problem for SL formulas can be reduced to the PTL-separability problem for OMPLs.

▶ **Lemma 10.** *Let Ψ_1 and Ψ_2 be two SL formulae with $x_1, \ldots, x_n$ as their set of variables. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two OMPAs such that $\mathcal{L}(\mathcal{A}_1)$=`Encode`$(\mathcal{L}(\Psi_1))$ and $\mathcal{L}(\mathcal{A}_2)$=`Encode`$(\mathcal{L}(\Psi_2))$. Ψ_1, Ψ_2 are n-PTL separable iff $\mathcal{A}_1, \mathcal{A}_2$ are PTL-separable.*

As an immediate corollary of Theorem 6, Lemma 10, we obtain our main result:

▶ **Theorem 11.** *The n-PTL separability problem of SL formulae is decidable.*

5 PosPTL-Separability of String Constraints

In this section, we address the separability problem for string constraints by a sub-class of PTL, called positive piece-wise testable languages (PosPTL). A language is in PosPTL iff it is defined as a finite positive Boolean combination (i.e., union and intersection but no complementation) of piece-languages. Given a natural number $n \in \mathbb{N}$, this definition can naturally be extended to n-tuples of words in the straightforward manner (as in the case of PTL) to obtain the class of n-PosPTL. In the following, we first provide a necessary and sufficient condition for the n-PosPTL separability problem of any two languages. Following result follows easily from PosPTL being upward closed.

▶ **Theorem 12.** *Two languages I and E are n-PosPTL separable iff $I{\uparrow} \cap E = \emptyset$ iff $I \cap E{\downarrow} = \emptyset$.*

The rest of this section is structured as follows: First, we show that the PosPTL separability is decidable for OMPLs; in the particular case of CFLs, this problem is PSPACE-complete. Then, we use the encoding of SL formulas to OMPAs (as defined in Section 4), and show that n-PosPTL separability of SL formulas reduces to the PosPTL-separability of corresponding OMPLs. Finally, we consider the PosPTL-separability problem for a subclass of SL formulas, called *right sided* SL formulas. We show that the PosPTL separability problem for this subclass is PSPACE-HARD and is in EXPSPACE.

5.1 PosPTL-Separability of SL formulas

First, we show that the PosPTL separability for OMPLs is decidable.

▶ **Theorem 13.** PosPTL *separability of OMPLs is decidable.*

Proof. Consider $\mathcal{C}$ to be the class of OMPLs in Theorem 12. Let I and E be two languages belonging to $\mathcal{C}$ as stated in Theorem 12. Then, the set $\mathsf{min}(I\uparrow)$ is effectively computable as an immediate consequence of the *Generalized Valk-Jantzen* construction [6]. The main idea behind this construction is to start with an empty minor set M (so to begin, $M \subseteq I$) and keep adding new words $w \in I$ to M if w is not already in $M\uparrow$. Before adding a new word, we need to test that $I \cap \overline{M\uparrow} \neq \emptyset$ (the complement of $M\uparrow$ intersects with I). This test is decidable since (i) OMPLs are closed under intersection with regular languages and (ii) the emptiness problem for OMPA is decidable [7]. At each step, we remove all the non-minimal words from M (since M is finite). The algorithm terminates due to the Higman's Lemma [24] (the minor of an upward closed set is finite). When the algorithm terminates, $I \subseteq M\uparrow$ and thus $I\uparrow \subseteq M\uparrow$. By construction, $M \subseteq I$ and $M\uparrow \subseteq I\uparrow$. Thus, $M\uparrow = I\uparrow$. Since M is a minor set, we have $\mathsf{min}(I\uparrow) = M$. Using (i) and (ii), we obtain the decidability of checking the emptiness of $I\uparrow \cap E$, and thus PosPTL separability of OMPL is decidable. ◀

As mentioned in section 4, the complexity of PTL-separability for 1-OMPL is open; however, we show that the PosPTL separability problem for 1-OMPL is PSPACE-COMPLETE.

▶ **Theorem 14.** *The* PosPTL*-separability for CFLs is* PSPACE*-*COMPLETE*.*

For the decidability of the n-PosPTL separability of SL formulas, we use the encoding of SL formulas to OMPAs (as defined in section 4), and show that the n-PosPTL separability of SL formulas reduces to the PosPTL separability of their corresponding OMPLs. The decidability of the n-PosPTL separability of SL formulas follows from Theorem 13.

▶ **Lemma 15.** *Given two SL formulas Ψ and Ψ', with $x_1, \ldots, x_n$ as their set of variables. Let $\mathcal{A}$ and $\mathcal{A}'$ be two OMPAs such that $\mathcal{L}(\mathcal{A}) = \texttt{Encode}(\mathcal{L}(\Psi))$ and $\mathcal{L}(\mathcal{A}') = \texttt{Encode}(\mathcal{L}(\Psi'))$. Then, Ψ and Ψ' are separable by an n-*PosPTL *iff $\mathcal{A}$ and $\mathcal{A}'$ are separable by a* PosPTL*.*

As an immediate corollary of Lemma 13 and 15, we obtain the following theorem:

▶ **Theorem 16.** *The n-*PosPTL *separability problem of SL formulae is decidable.*

5.2 PosPTL-Separability of Right-sided SL formula

Unfortunately, the proof of Theorem 16 does not allow us to extract any complexity result. Therefore, we consider in this subsection a useful fragment of SL formulas, called *right-sided* SL formulas. Roughly speaking, an SL formula Ψ is *right-sided* iff any variable appearing on the right-side of a relational constraint can not appear on the left-side of any relational constraint. Let us formalize the notion of right-sided SL formulas. Let us assume an SL

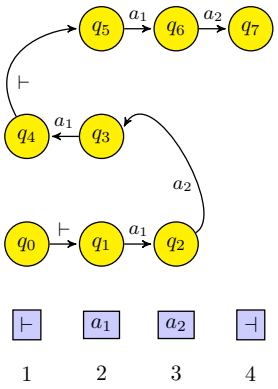
formula Ψ of the form $\bigwedge_{i=1}^{n} x_i \in \mathcal{L}(\mathcal{A}_i) \wedge \bigwedge_{i=1}^{k} (x_i, t_i) \in \mathcal{R}(\mathcal{T}_i)$ with $x_1, \ldots, x_n$ as set of variables. Then, Ψ is said to be *right-sided* if none of the variables $x_1, \ldots, x_k$ appear in any of $t_1, \ldots, t_k$. We call $x_{k+1}, \ldots, x_n$ (resp. $x_1, \ldots, x_k$) *independent* (resp. *dependent*) variables. Observe that the class of SL formulas with functional transducers can be rewritten as right-sided SL formulas (detailed proof can be found at [4]). A transducer $\mathcal{T}$ is functional if for every word w, there is at most one word w' such that $(w', w) \in \mathcal{R}(\mathcal{T})$ ($\mathcal{T}$ computes a function). An example of a functional transducer is the one implementing the identity relational constraint (allowing to express the equality $x = t$).

In the following, we show that the PosPTL-separability problem for right-sided SL formulas is in EXPSPACE. To show this result, we will reduce the PosPTL-separability problem for right-sided SL formulas to its corresponding problem for two-way transducers.

Two way transducers. Let Σ be a finite input alphabet and let $\vdash, \dashv$ be two special symbols not in Σ. We assume that every input string $w \in \Sigma^*$ is presented as $\vdash w \dashv$, where $\vdash, \dashv$ serve as left and right delimiters that appear nowhere else in w. We write $\Sigma_{\vdash\dashv} = \Sigma \cup \{\vdash, \dashv\}$. A two-way automaton $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ has a finite set of states Q, subsets $I, F \subseteq Q$ of initial and final states and a transition relation $\delta \subseteq Q \times \Sigma_{\vdash\dashv} \times Q \times \{-1, 1\}$. The -1 represents that the reading head moves to left after taking the transition while a 1 represents that it moves to right. The reading head cannot move left when it is on $\vdash$, and cannot move right when it is on $\dashv$. A configuration of $\mathcal{A}$ on reading $w' = \vdash w \dashv$ is represented by (q, i) where $q \in Q$ and i is a position in the input, $1 \leq i \leq |w| + 2$, which will be read in state q. An initial configuration is of the form $(q_0, 1)$ with $q_0 \in I$ and the reading head on $\vdash$. If $w' = w_1 a w_2$ and the current configuration is $(q, |w_1| + 1)$, and $(q, a, q', -1) \in \delta$, then there is a transition from the configuration $(q, |w_1| + 1)$ to $(q', |w_1|)$ (hence $a \neq \vdash$). Likewise, if $(q, a, q', 1) \in \delta$, we obtain a transition from $(q, |w_1| + 1)$ to $(q', |w_1| + 2)$. A run of $\mathcal{A}$ on reading $\vdash w \dashv$ is a sequence of transitions; it is accepting if it starts in an initial configuration and ends in a configuration of the form $(q, |w| + 2)$ with $q \in F$ and the reading head on $\dashv$. The language of $\mathcal{A}$ (denoted $\mathcal{L}(\mathcal{A})$) is the set of all words $w \in \Sigma^*$ s.t. $\mathcal{A}$ has an accepting run on $\vdash w \dashv$.

We extend the definition of a two-way automaton $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ into a two-way transducer (2NFT) $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, I, F)$ where Γ is a finite output alphabet. The transition relation is defined as a *finite* subset $\delta \subseteq Q \times \Sigma_{\vdash\dashv} \times Q \times \Gamma^* \times \{-1, 1\}$. The output produced on each transition is appended to the right of the output produced so far. $\mathcal{A}$ defines a relation $\mathcal{R}(\mathcal{A}) = \{(w, u) \mid w \text{ is the output produced on an accepting run of } u\}$. The acceptance condition is the same as in two-way automata. Sometimes, we use the macro-notation $(p, a, q, \alpha, 0)$ to denote a sequence of consecutive transitions (p, a, s, α, d) and (s, b, q, ϵ, d') in δ with $d + d' = 0$, $b \in \Sigma_{\vdash\dashv}$ and s is an extra intermediary state of $\mathcal{A}$ that is not used anywhere else (and that we omit from the set of states of $\mathcal{A}$).

PosPTL separability of 2NFT. In the following, we study the PosPTL separability of 2NFT. We define here the notion of *visiting sequences* (similar to *crossing sequences* of 2NFT [8]), which will be used in the proof of Lemma 17. Let $w = \vdash a_1 \ldots a_n \dashv$ be an input word and let ρ be a run of the 2NFT on w. A visiting sequence at a position x of a word w, in a run ρ of w captures the states visited in order in the run, each time the reading head is on position x, along with the information pertaining to the direction of the outgoing transition from that state. For example, in run ρ, if position x is visited for the first time in state q, and the outgoing transition chosen in ρ from q during that visit had direction $+1$, then q^+ will be the first entry in the visiting sequence. For a run ρ, the visiting sequence at a position x is defined as the tuple $\rho|x = (q_1^{d_1}, q_2^{d_2}, \ldots, q_h^{d_h})$ of states that have, in order, visited position x



in ρ, and whose outgoing transitions had direction $d_1, \ldots, d_h$. In the example, the visiting sequence at position 2 is (q_1^+, q_3^-, q_5^+), while those at 1 and 3 respectively are (q_0^+, q_4^+) and (q_2^-, q_6^+).

▶ **Lemma 17.** *Given a* 2NFT $\mathcal{T}$*, if* (v, u) *is in* $\mathsf{min}(\mathcal{R}(\mathcal{T})\uparrow)$ *then* $|u|$ *and* $|v|$ *are of at most exponential length in the size of* $\mathcal{T}$*.*

Proof. In the following, we show that, if $(v, u) \in \mathsf{min}(\mathcal{R}(\mathcal{T})\uparrow)$, then $|u| \leq \mathsf{in}_{\max} = \sum_{i=1}^{|Q|} ((2|Q|)^i \cdot |\Sigma|)$ and $|v| \leq \mathsf{out}_{\max} = \sum_{i=1}^{|Q|}((2|Q|)^i \cdot |\Sigma| \cdot |Q| \cdot \gamma_{\max})$ where $\gamma_{\max}$ represents the maximum length of an output on any transition in $\mathcal{T}$. To show this result, we need to define normalized runs as follows: A run is *normalized* if it visits each state at most once on each position x. In the following, we show that (v, u) can be generated by a normalized run ρ. Assume that (v, u) is accepted by a run ρ' which is not normalized. Then, we will have, in ρ', two visits to some position x of the word in the same state q. After the first visit to position x in state q, the transducer has explored some positions till its second visit to position x in state q. This part does not produce any output since (v, u) is a minimal word. We can delete this explored part of the run in between, obtaining again, an accepting run, which reads u while producing v. For example, in the figure if we have $q_6 = q_2$, then we have another run without visiting positions 1, 2 for a second time. Observe that repeating this procedure will lead to a normalized run ρ accepting (v, u). The length of visiting sequences in a normalized run is $\leq |Q|$ and hence the number of visiting sequences is at most exponential in $|Q|$, precisely it is $\leq \sum_{i=1}^{|Q|} (2|Q|)^i$.

Suppose $|u| > \mathsf{in}_{\max}$. Then there exists a visiting sequence which is repeated on reading the same input symbol in the accepting run of u, at positions $i \neq j$. By deleting the part between the ith and $(j-1)$th position, we again obtain an accepting run over a word u', which is a strict subword of u, and whose output v' is also a subword (may not be strict) of v, a contradiction to $(v, u) \in \mathsf{min}(\mathcal{R}(\mathcal{T})\uparrow)$. Now suppose $|u| \leq \mathsf{in}_{\max}$ but $|v| > \mathsf{out}_{\max}$. We saw that in the normalized run, each visiting sequence has length at most $|Q|$. Then, since $|u| \leq \mathsf{in}_{\max}$, on reading each position of u, at most $(|Q|)\gamma_{\max}$ symbols can be produced. Hence, we have $|v| \leq (|Q|) \cdot \gamma_{\max} \cdot |u|$. ◀

From Theorem 12 and Lemma 17, the following result holds:

▶ **Lemma 18.** *The 2-*PosPTL *separability problem for* 2NFT *is in* ExpSpace*.*

Proof. Using Theorem 12, we know that $\mathcal{R}(\mathcal{T}_1)\uparrow \cap \mathcal{R}(\mathcal{T}_2) = \emptyset$ iff $\mathcal{T}_1$ and $\mathcal{T}_2$ are 2-PosPTL separable. Here is an NExpSpace algorithm.

(1) Guess some (v, u) s.t. the lengths of v, u are at most as given by the proof of Lemma 17.
(2) Check if $(v, u) \in \mathcal{R}(\mathcal{T}_1)$. If yes, then do (3). Else exit.
(3) Check if $(v, u) \uparrow \cap \mathcal{R}(\mathcal{T}_2) \neq \emptyset$.

The guessed word (v, u) has exponential length in the size of $\mathcal{T}_1$. To check if $(v, u) \in \mathcal{R}(\mathcal{T}_1)$, we construct another transducer $\mathcal{T}_1'$ that first checks that its input word is u, then it comes back to $\vdash$ and starts simulating $\mathcal{T}_1$, while also keeping track, longer and longer prefixes of v. We then compare those prefixes with the output produced by $\mathcal{T}_1$. This gives rise to exponentially many states (maintaining prefixes of u and v) and we finish when $\mathcal{T}_1$ enters an accepting state, and at the same time, the produced word is v. Since $\mathcal{R}(\mathcal{T}_1') = \{(v, u)\} \cap \mathcal{R}(\mathcal{T}_1)$ by construction, checking if $(v, u) \in \mathcal{R}(\mathcal{T}_1)$ can be reduced to the emptiness problem of $\mathcal{T}_1'$. After this, we check the emptiness of $(v, u) \uparrow \cap \mathcal{R}(\mathcal{T}_2)$. This is done as follows. First, construct automata $\mathcal{A}_u, \mathcal{A}_v$ accepting languages $\{u\}\uparrow$ and $\{v\}\uparrow$ respectively. The number of states of $\mathcal{A}_u, \mathcal{A}_v$ are exponential in the number of states of $\mathcal{T}_1$, since the lengths of u, v have this bound. Then, we construct a transducer $\mathcal{T}_2'$ such that $\mathcal{R}(\mathcal{T}_2') = \{(v, u)\} \uparrow \cap \mathcal{R}(\mathcal{T}_2)$ in a similar manner as $\mathcal{T}_1'$. $\mathcal{T}_2'$ reads the input word while simulating $\mathcal{A}_u$. On entering an accepting state of $\mathcal{A}_u$, it comes back to $\vdash$. Then it simulates $\mathcal{T}_2$, and, on the outputs produced, simulates $\mathcal{A}_v$. If $\mathcal{A}_v$ enters an accepting state at the same time $\mathcal{T}_2$ accepts, then we are done. The state space of $\mathcal{T}_2'$ is exponential in the states of $\mathcal{T}_1$ and linear in the states of $\mathcal{T}_2$. Since $\mathcal{R}(\mathcal{T}_2') = \{(v, u)\} \uparrow \cap \mathcal{R}(\mathcal{T}_2)$, checking the emptiness of $(v, u) \uparrow \cap \mathcal{R}(\mathcal{T}_2)$ can be reduced to checking the emptiness problem of $\mathcal{T}_2'$. The emptiness problem for 2NFT is known to be PSPACE-COMPLETE [44]. Thus, in our case, the emptiness of $\mathcal{T}_1'$ and $\mathcal{T}_2'$ can be achieved in space exponential in $\mathcal{T}_1$. Since we can handle the second and third steps in exponential space, we obtain an NEXPSPACE algorithm. By Savitch's Theorem, we obtain the EXPSPACE complexity. ◀

From Right-sided SL formulas to 2NFT. Hereafter, we show how to encode the set of solutions of a right-sided SL formula using 2NFT. Let Σ be an alphabet and $\# \notin \Sigma$.

▶ **Lemma 19.** *Let Ψ be a right-sided SL formula over Σ, with $x_1, x_2, \ldots, x_n$ as its set of variables. Then, it is possible to construct, in polynomial time, a* 2NFT *$\mathcal{A}_\Psi$ such that $\mathcal{R}(\mathcal{A}_\Psi) = \{(u_1\#u_2\#\cdots\#u_n, w_1\#w_2\#\ldots\#w_n) | u_1\#u_2\#\cdots\#u_n \in \mathtt{Encode}(\mathcal{L}(\Psi))$ and $w_i = u_i$ if x_i is an independent variable $\}$.*

Proof. Let us assume that Ψ is of the form $\bigwedge_{i=1}^{n} y_i \in \mathcal{L}(\mathcal{A}_i) \wedge \bigwedge_{i=1}^{k} (y_i, t_i) \in \mathcal{R}(\mathcal{T}_i)$ with $y_1, \ldots, y_n$ is a permutation of $x_1, \ldots, x_n$. Let $\pi : [1, n] \to [1, n]$ be the mapping that associates to each index $i \in [1, n]$, the index $j \in [1, n]$ s.t. $x_i = y_j$ (or $x_i = y_{\pi(i)}$). We construct $\mathcal{A}_\Psi$ as follows: $\mathcal{A}_\Psi$ reads n words over Σ separated by $\#$ as input. We explain hereafter the working of $\mathcal{A}_\Psi$ when it produces the assignment for x_1 (the other variables are handled in similar manner).
• Assume that x_1 is a dependent variable. Let $\varphi_{\pi(1)} = (y_{\pi(1)}, t_{\pi(1)}) \in \mathcal{R}(\mathcal{T}_{\pi(1)})$, with $t_{\pi(1)} = x_{i_1} x_{i_2} \ldots x_{i_c}$ and $x_{i_j} \in \{y_{k+1}, y_{k+2}, \ldots, y_n\}$ for all j. First, $\mathcal{A}_\Psi$ reads x_{i_1} i.e. the first variable in $t_{\pi(1)}$. To read x_{i_1}, it skips $(i_1 - 1)$ many blocks separated by #s of the input, and comes to w_{i_1}. On the first symbol of w_{i_1}, $\mathcal{A}_\Psi$ starts mimicking transitions of $\mathcal{T}_{\pi(1)}$ from its initial state, while producing the same output as $\mathcal{T}_{\pi(1)}$. On the same output, $\mathcal{A}_\Psi$ mimics the transitions of $\mathcal{A}_{\pi(1)}$ starting from the initial state to check the membership constraint of $y_{\pi(1)}$. This can be done by a product construction between $\mathcal{A}_{\pi(1)}$ and $\mathcal{T}_{\pi(1)}$. For instance, $\mathcal{A}$ will have a transition $((p, q), a, (p', q'), b, 1)$ (resp. $((p, q), a, (p', q'), b, 0)$), if there are transitions $(p, (b, a), p')$ (resp. $(p, (b, \epsilon), p')$) in $\mathcal{T}_{\pi(1)}$ and (q, b, q') in $\mathcal{A}_{\pi(1)}$. If it reaches # or $\dashv$ in the input, it remembers the current states of $\mathcal{T}_{\pi(1)}$ and $\mathcal{A}_{\pi(1)}$, say (p_1, q_1) in its control state. Next, $\mathcal{A}_\Psi$ reads x_{i_2} in the input. To read x_{i_2}, $\mathcal{A}_\Psi$ moves to $\vdash$ and then changes direction.

As before it reaches x_{i_2} by skipping $(i_2 - 1)$ many #s, and starts reading the input (the first symbol of w_{i_2}) from the state (p_1, q_1) stored in the finite control. Transitions are similar to explained above. This procedure is repeated to read $x_{i_3} \dots x_{i_c}$. After reading x_{i_c}, if the next state contains the pair (p_c, q_c), where p_c (resp. q_c) is a final state of $\mathcal{T}_{\pi(1)}$ (resp. $\mathcal{A}_{\pi(1)}$), we can say that the output produced till now satisfies $\varphi_{\pi(1)}$ and $y_{\pi(1)} \in \mathcal{L}(\mathcal{A}_{\pi(1)})$. Observe that this procedure requires $|t_{\pi(1)}|$ reversals, and thus we need at most $|t_{\pi(1)}|$ copies of transducer $\mathcal{T}_{\pi(1)}$. Thus the number of states required are at most polynomial.

• Assume now that x_1 is an independent variable, then $\mathcal{A}_\Psi$ needs to read x_1. We need a single pass of the input which verifies if the first block corresponding to value of x_1 in input indeed satisfies its corresponding membership constraint. During this pass $\mathcal{A}_\Psi$ mimics transitions of $\mathcal{A}_{\pi(1)}$ starting from its initial states, and outputs the same letter as input.

The above procedure is repeated for all variables from x_2 to x_n. After each pass, $\mathcal{A}_\Psi$ moves to $\vdash$ and then changes direction. Irrespective of whether x_i is dependent or not, while going from x_i to x_{i+1}, $i \in [1, n-1]$, $\mathcal{A}_\Psi$ outputs a # as separator. From the description above, it can be seen that if x_i is independent, then its evaluation u_i given as the ith block of the input is equal to the output w_i, and if x_i is a dependent variable, then the output block w_i is the output of $\mathcal{T}_{\pi(i)}$. It is clear from the construction that $\mathcal{A}_\Psi$ requires at most polynomial states in input. More details can be found at [4]. ◀

Notice that the above construction of 2NFT relies on the right-sidedness: if a variable x_i appears in the output of $\mathcal{T}_i$ and also in the input of $\mathcal{T}_k$ for some k, then we will have to store the produced evaluation of x_i in order to use it later on when processing $\mathcal{T}_k$. However, there is no way to store the produced evaluation of x_i or compare it with its input evaluation. Next, we show that the PosPTL separability problem for right-sided formulas can be reduced to its corresponding problem for 2NFT.

▶ **Lemma 20.** *Let Ψ_1 and Ψ_2 be two right-sided SL formula, with $x_1, \dots, x_n$ as their set of variables. Let $\mathcal{A}_{\Psi_1}$ and $\mathcal{A}_{\Psi_2}$ be the two* 2NFT*s encoding, respectively, the set of solutions of Ψ_1 and Ψ_2 (as described in Lemma 19). Then, the two formulae Ψ_1 and Ψ_2 are separable by n-*PosPTL *iff $\mathcal{R}(\mathcal{A}_{\Psi_1})$ and $\mathcal{R}(\mathcal{A}_{\Psi_2})$ are separable by a 2-*PosPTL*.*

Proof. Let $\mathcal{R}(\mathcal{A}_{\Psi_1})$ and $\mathcal{R}(\mathcal{A}_{\Psi_2})$ be separable by a 2-PosPTL L. By definition, L is a Boolean combination (except complementation) of piece languages of words over the two tuple alphabet $(\Sigma \cup \{\#\})^2$. We can assume w.l.o.g. that L is the union of piece languages. This is possible since the intersection of two piece languages can be rewritten as a union of piece languages. Consider $L' = L \cap (R \times R)$, where R is a regular language consisting of words having exactly $(n-1)$ #s. We claim that L' can be rewritten as the union of languages of the form $[L_1 \# L_2 \# \dots \# L_n] \times [R_1 \# R_2 \# \dots \# R_n]$ where the L_is and R_is are piece languages over Σ, and that L' is also a separator of $\mathcal{R}(\mathcal{A}_{\Psi_1})$ and $\mathcal{R}(\mathcal{A}_{\Psi_2})$.

We prove this claim inductively. As a base case consider L to be a piece language $((\Sigma \cup \{\#\})^*)^2 (a_1, b_1)((\Sigma \cup \{\#\})^*)^2 \dots (a_m, b_m)((\Sigma \cup \{\#\})^*)^2$. Let S be a finite set containing only the *minimal* words of the form (w, w') such that $a_1 a_2 \dots a_m \preceq w$, $b_1 b_2 \dots b_m \preceq w'$, and the symbol # appears exactly $(n-1)$-times in w and w'. Thus $L \cap (R \times R) = \bigcup_{(a'_1 \dots a'_k, b'_1 \dots b'_\ell) \in S} \left[\Sigma^* a'_1 \Sigma^* a'_2 \dots a'_k \Sigma^*\right] \times \left[\Sigma^* b'_1 \Sigma^* b'_2 \dots b'_\ell \Sigma^*\right]$. So $L \cap (R \times R)$ is the union of piece languages of the form $[L_1 \# L_2 \# \dots \# L_n] \times [R_1 \# R_2 \# \dots \# R_n]$ where the L_is and R_is are piece languages over Σ. Now assume that L is of the form $L_1 \cup L_2$. It is easy to see that $L \cap (R \times R)$ is equivalent to $(L_1 \cap (R \times R)) \cup (L_2 \cap (R \times R))$. Thus we can use our induction hypothesis to show that $L \cap (R \times R)$ is the union of languages of the form $[L_1 \# L_2 \# \dots \# L_n] \times [R_1 \# R_2 \# \dots \# R_n]$ where L_i and R_is are piece languages over Σ.

Next we prove that L' is a separator of $\mathcal{R}(\mathcal{A}_{\Psi_1})$ and $\mathcal{R}(\mathcal{A}_{\Psi_2})$. Indeed if $(v,u) \in \mathcal{R}(\mathcal{A}_{\Psi_1})$, then $(v,u) \in R \times R$, by definition of $\mathcal{R}(\mathcal{A}_{\Psi_1})$. Since L is a separator, we have $(v,u) \in L$ and hence $(v,u) \in L'$. Suppose $(v,u) \in \mathcal{R}(\mathcal{A}_{\Psi_2}) \cap L'$, then $(v,u) \in L \cap \mathcal{R}(\mathcal{A}_{\Psi_1})$ since $(v,u) \in L'$, and $L' \subseteq L$, which is a contradiction with the assumption that L is a separator.

Now we are in a condition to provide n-PosPTL separator for $\mathcal{L}(\Psi_1)$ and $\mathcal{L}(\Psi_2)$, using L'. Given a language of the form $[L_1\#L_2\#\ldots\#L_n] \times [R_1\#R_2\#\ldots\#R_n]$ where L_i and R_is are piece languages, we associate to it an n-PosPTL equivalent to $((L_1 \cap R_1) \times (L_2 \cap R_2) \times \ldots \times (L_n \cap R_n))$: the idea is to generate the n dimensions in the n-PosPTL from the n #-separated blocks in two dimensions. This definition is extended in the straightforward manner to union of piece languages. Let K be the n-PosPTL associated to L'. K is indeed a separator of $\mathcal{L}(\Psi_1)$ and $\mathcal{L}(\Psi_2)$: Suppose $\mathbf{v} = (w_1,\ldots,w_n) \in \mathcal{L}(\Psi_1)$, then $(w_1\#\ldots\#w_n, w_1\#\ldots\#w_n) \in \mathcal{R}(\mathcal{A}_{\Psi_1})$ (from the definition of $\mathcal{A}_{\Psi_1}$). Since L' is a separator, $(w_1\#\ldots\#w_n, w_1\#\ldots\#w_n) \in L'$. By construction of K, $(w_1,\ldots,w_n) \in K$. Assume by contradiction $\mathbf{v} = (w_1,\ldots,w_n) \in \mathcal{L}(\Psi_2) \cap K$, then $(w_1\#\ldots\#w_n, w_1\#\ldots\#w_n) \in L'$. Since $L' \cap \mathcal{R}(\mathcal{A}_{\Psi_2}) = \emptyset$, then $(w_1\#\ldots\#w_n, w_1\#\ldots\#w_n) \notin \mathcal{R}(\mathcal{A}_{\Psi_2})$. By definition of $\mathcal{A}_{\Psi_2}$, if $(w_1,\ldots,w_n) \in \mathcal{L}(\Psi_2)$, then $(w_1\#\ldots\#w_n, w_1\#\ldots w_n) \in \mathcal{R}(\mathcal{A}_{\Psi_2})$. Hence contradiction.

For the other direction of the proof, assume the n-PosPTL S is a separator of $\mathcal{L}(\Psi_1)$ and $\mathcal{L}(\Psi_2)$. Then S can be rewritten as the union of $(L_1 \times L_2 \times \ldots \times L_n)$ where L_is are piece languages. Replace each n-piece language $(L_1 \times L_2 \times \ldots \times L_n)$ of S with the 2-piece language $(L_1'\#L_2'\#\ldots\#L_n') \times ((\Sigma \cup \{\#\})^*\#\ldots\#(\Sigma \cup \{\#\})^*)$, where $L_1' = (\Sigma \cup \{\#\})^* a_1 (\Sigma \cup \{\#\})^* \ldots a_n (\Sigma \cup \{\#\})^*$ if $L_1 = \Sigma^* a_1 \Sigma^* \ldots a_n \Sigma^*$. Denote the union of such languages by S'. It is a 2-PosPTL over $(\Sigma \cup \{\#\})$. We show that S' is a 2-PosPTL separator of $\mathcal{R}(\mathcal{A}_{\Psi_1})$ and $\mathcal{R}(\mathcal{A}_{\Psi_2})$. Let $\mathbf{(v,\ u)} = (v_1\#\ldots\#v_n, u_1\#\ldots\#u_n) \in \mathcal{R}(\mathcal{A}_{\Psi_1})$, then $(v_1,\ldots,v_n) \in \mathcal{L}(\Psi_1)$, and thus $(v_1,\ldots,v_n) \in S$ (since S is a separator). This implies that $\mathbf{(v,\ u)} \in S'$ by its construction. Suppose $\mathbf{(v,\ u)} = (v_1\#\ldots\#v_n, u_1\#\ldots\#u_n) \in \mathcal{R}(\mathcal{A}_{\Psi_2}) \cap S'$, then $(v_1, v_2, \ldots, v_n) \in \mathcal{L}(\Psi_2)$. Also $(v_1, v_2, \ldots, v_n) \in S$ by construction of S'. This leads to a contradiction that S is separator of $\mathcal{L}(\Psi_1)$ and $\mathcal{L}(\Psi_2)$. So S' is a 2-PosPTL separator of $\mathcal{R}(\mathcal{A}_{\Psi_1})$ and $\mathcal{R}(\mathcal{A}_{\Psi_2})$. ◀

▶ **Theorem 21.** *The n-PosPTL separability of right-sided SL formulas is in Expspace and is Pspace-Hard.*

Proof. Given two right-sided SL formulas Ψ_1 and Ψ_2, one can construct corresponding two way transducers $\mathcal{A}_{\Psi_1}$ and $\mathcal{A}_{\Psi_2}$ with polynomial states, as mentioned in Lemma 19. Thanks to Lemma 20, the n-PosPTL separability reduces to 2-PosPTL separability of $\mathcal{R}(\mathcal{A}_{\Psi_1})$ and $\mathcal{R}(\mathcal{A}_{\Psi_2})$. The 2-PosPTL separability of 2NFTs is in Expspace (Lemma 18). Hence n-PosPTL separability of SL formulae is also in Expspace. For the Pspace-Hard lower bound, we reduce the emptiness of k-NFA intersection to PosPTL separability of right sided SL. Let $\mathcal{A}_1,\ldots,\mathcal{A}_k$ be k-NFA. We want to decide if $\bigcap_{i=1}^k \mathcal{A}_i = \emptyset$. Let Ψ_1 be $\bigwedge_{i=1}^k x_i = x \wedge \bigwedge_{i=1}^k (x_i \in \mathcal{A}_i)$, and Ψ_2 be $x \in \Sigma^* \wedge \bigwedge_{i=1}^k (x_i \in \Sigma^*)$. Ψ_1 and Ψ_2 are PosPTL separable iff $\bigcap_{i=1}^k \mathcal{A}_i = \emptyset$. ◀

References

1 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Bui Phi Diep, Lukás Holík, Ahmed Rezine, and Philipp Rümmer. Flatten and conquer: a framework for efficient analysis of string constraints. In *PLDI*. ACM, 2017.

2 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Bui Phi Diep, Lukás Holík, Ahmed Rezine, and Philipp Rümmer. Trau: SMT solver for string constraints. In *FMCAD*. IEEE, 2018.

3 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Lukás Holík, Ahmed Rezine, Philipp Rümmer, and Jari Stenman. String constraints for verification. In *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.

4 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Vrunda Dave, and Shankara Narayanan Krishna. On the separability problem of string constraints. *CoRR*, abs/2005.09489, 2020. `arXiv:2005.09489`.

5 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*, pages 277–293. Springer, 2019.

6 Parosh Aziz Abdulla and Richard Mayr. Priced timed petri nets. *Logical Methods in Computer Science*, 9(4), 2013. `doi:10.2168/LMCS-9(4:10)2013`.

7 Mohamed Faouzi Atig, Benedikt Bollig, and Peter Habermehl. Emptiness of Ordered Multi-Pushdown Automata is 2ETIME-Complete. *Int. J. Found. Comput. Sci.*, 28(8):945–976, 2017.

8 Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. Minimizing resources of sweeping and streaming string transducers. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 114:1–114:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

9 Murphy Berzish, Vijay Ganesh, and Yunhui Zheng. Z3str3: A string solver with theory-aware heuristics. In *2017 Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, October 2-6, 2017*, pages 55–59. IEEE, 2017.

10 Murphy Berzish, Yunhui Zheng, and Vijay Ganesh. Z3str3: A string solver with theory-aware branching. *CoRR*, abs/1704.07935, 2017. `arXiv:1704.07935`.

11 Luca Breveglieri, Alessandra Cherubini, Claudio Citrini, and Stefano Crespi-Reghizzi. Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996. `doi:10.1142/S0129054196000191`.

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

13 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), January 2019. `doi:10.1145/3290362`.

14 Taolue Chen, Matthew Hague, Anthony W. Lin, Philipp Rümmer, and Zhilin Wu. Decision procedures for path feasibility of string-manipulating programs with complex operations. *PACMPL*, 3(POPL):49:1–49:30, 2019.

15 Lorenzo Clemente, Wojciech Czerwinski, Slawomir Lasota, and Charles Paperman. Regular separability of parikh automata. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 117:1–117:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

16 Wojciech Czerwinski and Slawomir Lasota. Regular separability of one counter automata. *Logical Methods in Computer Science*, 15(2), 2019. URL: `https://lmcs.episciences.org/5563`, `doi:10.23638/LMCS-15(2:20)2019`.

17 Wojciech Czerwinski, Slawomir Lasota, Roland Meyer, Sebastian Muskalla, K. Narayan Kumar, and Prakash Saivasan. Regular separability of well-structured transition systems. In *29th International Conference on Concurrency Theory, CONCUR 2018, September 4-7, 2018, Beijing, China*, volume 118 of *LIPIcs*, pages 35:1–35:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

18 Wojciech Czerwinski, Wim Martens, and Tomás Masopust. Efficient separability of regular languages by subsequences and suffixes. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, volume 7966 of *Lecture Notes in Computer Science*, pages 150–161. Springer, 2013.

19 Wojciech Czerwinski, Wim Martens, Lorijn van Rooijen, Marc Zeitoun, and Georg Zetzsche. A characterization for decidable separability by piecewise testable languages. *Discrete Mathematics & Theoretical Computer Science*, 19(4), 2017.

20 Vijay Ganesh and Murphy Berzish. Undecidability of a theory of strings, linear arithmetic over length, and string-number conversion. *CoRR*, abs/1605.09442, 2016. `arXiv:1605.09442`.

21 Vijay Ganesh, Mia Minnes, Armando Solar-Lezama, and Martin C. Rinard. Word equations with length constraints: What's decidable? In *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*, pages 209–226. Springer, 2012.

22 Vijay Ganesh, Mia Minnes, Armando Solar-Lezama, and Martin C. Rinard. (Un)Decidability results for word equations with length and regular expression constraints. *CoRR*, abs/1306.6054, 2013. `arXiv:1306.6054`.

23 Seymour Ginsburg and Sheila A. Greibach. Abstract families of languages. In *8th Annual Symposium on Switching and Automata Theory, Austin, Texas, USA, October 18-20, 1967*. IEEE Computer Society, 1967.

24 G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc. (3)*, 2(7), 1952.

25 Lukás Holík, Petr Janku, Anthony W. Lin, Philipp Rümmer, and Tomás Vojnar. String constraints with concatenation and transducers solved efficiently. *PACMPL*, 2(POPL):4:1–4:32, 2018. `doi:10.1145/3158092`.

26 Scott Kausler and Elena Sherman. Evaluation of string constraint solvers in the context of symbolic execution. In *ASE '14*. ACM, 2014.

27 Adam Kiezun, Vijay Ganesh, Philip J. Guo, Pieter Hooimeijer, and Michael D. Ernst. HAMPI: a solver for string constraints. In *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis, ISSTA 2009, Chicago, IL, USA, July 19-23, 2009*, pages 105–116. ACM, 2009.

28 Tianyi Liang, Andrew Reynolds, Cesare Tinelli, Clark Barrett, and Morgan Deters. A DPLL(T) theory solver for a theory of strings and regular expressions. In *CAV'14*, volume 8559 of *LNCS*. Springer, 2014.

29 Tianyi Liang, Andrew Reynolds, Nestan Tsiskaridze, Cesare Tinelli, Clark W. Barrett, and Morgan Deters. An efficient SMT solver for string constraints. *Formal Methods in System Design*, 48(3):206–234, 2016. `doi:10.1007/s10703-016-0247-6`.

30 Anthony Widjaja Lin and Pablo 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 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 123–136. ACM, 2016.

31 MakePHPSites. Php tutorials.`https://makephpsites.com/php-tutorials/user-management-tools/changing-passwords.php`, 2015.

32 Kenneth L. McMillan. Interpolation and SAT-based model checking. In *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*, volume 2725 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2003.

33 Kenneth L. McMillan. An interpolating theorem prover. In *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings*, volume 2988 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2004.

34 Kenneth L. McMillan. Lazy abstraction with interpolants. In *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4144 of *Lecture Notes in Computer Science*, pages 123–136. Springer, 2006.

35 Christophe Morvan. On rational graphs. In *Foundations of Software Science and Computation Structures*, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

36 Thomas Place, Lorijn van Rooijen, and Marc Zeitoun. Separating regular languages by piecewise testable and unambiguous languages. In *Mathematical Foundations of Computer Science 2013 - 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings*, volume 8087 of *Lecture Notes in Computer Science*, pages 729–740. Springer, 2013.

37 Wojciech Plandowski. An efficient algorithm for solving word equations. In *Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing*, STOC '06, pages 467–476, New York, NY, USA, 2006. ACM. doi:10.1145/1132516.1132584.

38 Andrew Reynolds, Maverick Woo, Clark W. Barrett, David Brumley, Tianyi Liang, and Cesare Tinelli. Scaling up DPLL(T) string solvers using context-dependent simplification. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, volume 10427 of *Lecture Notes in Computer Science*, pages 453–474. Springer, 2017.

39 Prateek Saxena, Devdatta Akhawe, Steve Hanna, Feng Mao, Stephen McCamant, and Dawn Song. A symbolic execution framework for javascript. In *31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berleley/Oakland, California, USA*, pages 513–528. IEEE Computer Society, 2010.

40 Prateek Saxena, Steve Hanna, Pongsin Poosankam, and Dawn Song. FLAX: systematic discovery of client-side validation vulnerabilities in rich web applications. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2010, San Diego, California, USA, 28th February - 3rd March 2010*. The Internet Society, 2010.

41 Imre Simon. Piecewise testable events. In *Automata Theory and Formal Languages, 2nd GI Conference, Kaiserslautern, May 20-23, 1975*, volume 33 of *Lecture Notes in Computer Science*, pages 214–222. Springer, 1975.

42 Minh-Thai Trinh, Duc-Hiep Chu, and Joxan Jaffar. S3: A symbolic string solver for vulnerability detection in web applications. In *CCS'14*. ACM, 2014.

43 Minh-Thai Trinh, Duc-Hiep Chu, and Joxan Jaffar. Progressive reasoning over recursively-defined strings. In *CAV'16*, volume 9779 of *LNCS*. Springer, 2016.

44 Moshe Y. Vardi. A note on the reduction of two-way automata to one-way automata. *Inf. Process. Lett.*, 30(5):261–264, 1989. doi:10.1016/0020-0190(89)90205-6.

45 Hung-En Wang, Tzung-Lin Tsai, Chun-Han Lin, Fang Yu, and Jie-Hong R. Jiang. String analysis via automata manipulation with logic circuit representation. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I*, volume 9779 of *Lecture Notes in Computer Science*, pages 241–260. Springer, 2016.

46 Fang Yu, Muath Alkhalaf, and Tevfik Bultan. Stranger: An automata-based string analysis tool for PHP. In *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6015 of *Lecture Notes in Computer Science*, pages 154–157. Springer, 2010.

47 Yunhui Zheng, Xiangyu Zhang, and Vijay Ganesh. Z3-str: A Z3-based string solver for web application analysis. In *ESEC/FSE'13*. ACM, 2013.