Expressive Decidable String Constraints with Parametric Transducers: A Generic Approach

Abstract. Finite-state transducers have emerged as an expressive formalism for modelling string functions including those that are crucial for analysis of cross-site scripting vulnerabilities in HTML5 applications, e.g., sanitisers. Thanks to their good algorithmic properties, finite-state transducers have been incorporated into decidable theories over strings (with concatenation and regular expression pattern matching) in a way that is applicable for symbolic execution analysis of string-manipulating programs. Despite this, such a formalism lacks the expressive power for modelling many useful string functions; three notable examples being the replaceall function, string reverse, and string concatenation. In this paper we introduce *parametric transducers*, an expressive formalism for modelling string functions that strictly generalises the standard finite-state transducer model and captures the three aforementioned string functions, among others. We provide an expressive decidable string logic with parametric transducers as primitive operations, and a detailed complexity analysis. The logic subsumes existing decidable string theories for symbolic execution analysis of programs with strings. We show that string functions required for context-sensitive auto-sanitisation web templating can be modelled in our logic. Finally, we lift our results to the world of symbolic finite automata/transducers, which is useful for practical applications (e.g. to deal with large alphabets like UTF16).

1 Introduction

Strings are a fundamental data type in virtually all programming languages. Their generic nature can, however, lead to many subtle programming bugs, some with security consequences, e.g., cross-site scripting (XSS), which is among OWASP Top 10 Application Security Risks [53]. One effective automatic testing method for identifying subtle programming errors is based on *symbolic executions* [35] and combinations with dynamic analysis called *dynamic symbolic execution* [12, 13, 25, 49, 50]. See [14] for an excellent survey. Unlike purely random testing, which runs only *concrete* program executions on different inputs, the techniques of symbolic executions analyse *static* paths (also called symbolic executions) through the software system under test. Since such a path is simply a sequence of assignments and conditionals/assertions, it can be viewed as a constraint φ (over appropriate data domains) and a solver could be invoked on φ to check the *feasibility* of the static path, which can be used for generating inputs that lead to certain parts of the program or an erroneous behaviour.

A symbolic execution analysis relies on constraint solvers at its core. When the system under test is a program that uses string data type (including most programs written in scripting languages like JavaScript/PHP), these techniques crucially need constraint solvers over the string domain (a.k.a. *string solvers*). This in fact has been one main reason behind the development of the theory,

implementation, and symbolic execution applications of string solving in the past decade, e.g., see [1, 2, 5, 10, 15, 24, 28, 34, 38–40, 43, 47, 49, 51, 52, 56, 59–62].

Constraints from Symbolic Executions. As elegantly described in Björner *et al.* [10], constraints from symbolic execution on string-manipulating programs can be understood in terms of the problem of *path feasibility* over a bounded program S with neither loop nor branching. That is, S is a sequence of assignments and conditionals/assertions, i.e., generated by the grammar

$$S ::= \quad y := f(x_1, \ldots, x_r) \mid \mathbf{assert}(g(x_1, \ldots, x_r)) \mid S_1; S_2 \tag{1}$$

where $f : (\Sigma^*)^r \to \Sigma^*$ and $g : (\Sigma^*)^r \to \{0, 1\}$ are some partial string functions. The syntaxes for the functions f and g are now left undefined, but will be instantiated as one explores algorithmic issues. The following is a simple example of a symbolic execution S which uses string variables (x, y, and z's) and string constants (letters `a` and `b`), and the concatenation operator ($\cdot$):

$$z_1 := x \cdot \mathtt{ba} \cdot y; \quad z_2 := y \cdot \mathtt{ab} \cdot x; \quad \mathbf{assert}(z_1 == z_2) \tag{2}$$

The problem of *path feasibility/satisfiability*[1] asks whether, for a given program S, there exist *input* strings (e.g. x and y in Example in (2)) that can successfully take S to the end of the program while satisfying all the assertions. Example in (2) can be satisfied by assigning y (resp. x) to `b` (resp. the empty string).

Decidable string constraint languages. A central theme in string solving is to design an expressive string constraint language for symbolic executions of string-manipulating programs that admits *decidability*. Being the most basic string operation, concatenation is typically present in a string theory. Constraints in the *theory of concatenation* are symbolic executions with concatenation of string constants/variables in the assignment (f in (1)) and equality/disequality checks of variables in the assertions (g in (1)). Example in (2) is an instance of a constraint in this theory. The theory of concatenation was proven to be decidable in the seminal paper by Makanin [41] on *word equations*, followed by the extension by Büchi and Senger [11]. This decidability holds even when *regular constraints* (i.e. regular expression matching, e.g., $y \in (\mathtt{ab})^*$) — useful to specify the set of "bad" strings — are allowed as assertions [48].

Sanitisers (e.g. htmlescape and backslash-escape) and implicit browser transductions (e.g. innerHTML) are two types of string functions that are frequently used in client-side web applications. Many XSS vulnerabilities are caused by subtle interactions between these functions and other parts of the applications, e.g., see [18, 27, 33, 39, 58]. (One-way) *finite-state transducers* [18, 29, 39, 55, 58] are a good formal model of such string functions that satisfy some good closure and algorithmic properties. Finite transducers are just like finite-state automata but with an output tape. Unfortunately, adding finite-state transductions to the

[1] It is equivalent to satisfiability of string constraints in SMT framework [6, 21, 37]. Simply convert a symbolic execution S into a *Static Single Assignment* (SSA) form (i.e. use a new variable on l.h.s. of each assignment) and treat assignments as equality, e.g., formula for the above example is $z_1 = x + \mathtt{ba} + y \wedge z_2 = y + \mathtt{ab} + x \wedge z_1 = z_2$. Many decidable string logics can be more naturally phrased in terms of path feasibility.

theory of concatenation (in the assignment of (1)) results in undecidability [39]. More recently, a decidable logic with these additional string operations [39] has been obtained by imposing the *straight-line restriction*. This is done by *disallowing string equality checks in the assertions g* in (1). In other words, concatenation and finite-state transductions can both be used in assignments, while only regular constraints can be used in assertions. The string logic is still sufficiently expressive for many examples that arise in practice [28, 39].

Despite the expressive power of finite-state transducers, they cannot model some important string functions. In addition to string concatenation (which justifies combining transducers and concatenation in string constraints [39]), there are two other important examples. The first example is the replaceall functions $\mathsf{replaceAll}_p(sub, rep)$ where the subject *sub* and the replacement *rep* are string *variables*, i.e., replacing every matching of the pattern p in the subject *sub* by *rep*. In a recent paper [15], such replaceall functions were argued to be important for modelling string functions used in web templating like Mustache.js [57], and that straight-line string constraints with replaceall/concatenation in assignments were shown to be decidable. Secondly, the string reverse function cannot be expressed by a finite-state transducer. To admit such a function, we need to allow *two-way* finite-state transducers [9, 44], which so far have not been incorporated into a string constraint language. *Our proposal is to design an expressive decidable string constraint language that admits all the aforementioned string functions, which furthermore unifies the decidable string logics of [15, 39].*

Contributions. Rather than admitting many different classes of string functions in our constraint language, we introduce *parametric transducers*, an expressive formalism for modelling string (partial) functions $f : (\Sigma^*)^r \to \Sigma^*$ with multiple inputs $x_1, \ldots, x_r$. Parametric transducers strictly generalise the standard finite-state transducers (which take only one input) by allowing: (1) parameters $x_2, \ldots, x_r$ in the output (x_1, the first argument of f, is placed on the input tape), and (2) the two-way (left/right) head pointing to the input tape. Parametric transducers are capable of modelling the string functions as diverse as replaceall, string reverse, and concatenation, as well as finite-state transductions, thus *unifying them into one single formalism*. As a result of the expressibility, parametric transducers can model string functions required in *context-sensitive auto-sanitisation* in web templating systems [45], e.g., Google Closure [26] and Handlebars [32]. We provide a *generic* automata-theoretic decision procedure for the path feasibility problem of symbolic executions that use parametric transducers in assignments and regular constraints in assertions. Our constraint language is one of the *most expressive decidable* string languages for symbolic executions of programs with strings. We also provide a detailed complexity analysis of the generic decision procedure and its instantiations with different classes of transducers[2]. Among others, our generic decision procedure implies that *the path*

[2] *Worst-case complexity* results do not prohibit fast solvers *in practice*, e.g., SLOTH for the straight-line logic with finite transducers (EXPSPACE) [28], and MONA for Weak Second-Order Theory of Successors (NONELEMENTARY). Rather, they *inform* algorithmic techniques that could work in practice, e.g., symbolic techniques

feasibility problem is EXPSPACE*-complete for the straight-line string constraint language including all the aforementioned string functions, namely, (functional) one-way finite transducers, concatenation, replaceall, and reverse*, thus strictly generalising the EXPSPACE-completeness result for finite transducers and concatenation in [39], and for replaceall in [15]. Finally, we extend our results to symbolic automata/transducers [18, 20, 29, 55], which are crucial for practical applications requiring large alphabets (e.g. UTF16).

Organisation. After Preliminaries (§2), we define parametric transducers and our constraint language in §3. We provide a generic decision procedure in §4, which will be instantiated with different subclasses of parametric transducers in §5. We provide an extension of our results to the world of symbolic automata and transducers in §6. We conclude with related work and future work in §7. Missing details can be found in the appendix.

2 Preliminaries

General Notations. Let $\mathbb{Z}$ and $\mathbb{N}$ denote the set of integers and natural numbers respectively. For $i \leq j \in \mathbb{N}$, $[i, j] := \{i, i+1, \dots, j\}$ and $[i] := [1, i]$. For a vector $\vec{x} = (x_1, \cdots, x_n)$, let $|\vec{x}|$ denote the length of $\vec{x}$ (i.e., n) and $\vec{x}[i]$ denote x_i for each $i \in [n]$. For a set S, we use S^* (resp. S^+) to denote the set of all finite (resp. finite and nonempty) sequences over S. We use ϵ for the empty sequence. Given a function $f : A \to B$ and $X \subseteq B$, we use $f^{-1}(X)$ to define the pre-image of X under f, i.e., $\{a \in A : f(a) \in X\}$.

Automata. We review some background from automata theory; for more, see [30, 36]. Let Σ be a finite set (called *alphabet*). We usually define $\overline{\Sigma} := \Sigma \cup \{\triangleright, \triangleleft\}$, where $\triangleright, \triangleleft \notin \Sigma$ are tape end markers for 2-way automata/transducer models. A sequence over Σ is called a *string*. A *language* is set of strings over Σ. We will review regular languages and necessary models of finite-state automata below.

Definition 1 (Two-way finite-state automata). *A* (nondeterministic) two-way finite-state automaton *(2FA) over a finite alphabet Σ is a tuple $\mathcal{A} = (\Sigma, \triangleright, \triangleleft, Q, q_0, F, \delta)$ where Q is a finite set of states, $\triangleright$ (resp. $\triangleleft$) a left (resp. right) input tape end marker, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is a set of final states, and δ is the transition relation $\delta \subseteq Q \times \overline{\Sigma} \times \{-1, 0, 1\} \times Q$. Here, we assume that there are no transitions that take the head of the tape past the left/right end marker (i.e. $(p, \triangleright, -1, q), (p, \triangleleft, 1, q) \notin \delta$ for every $p, q \in Q$).*

A (nondeterministic one-way) finite-state automaton (FA) is a 2FA such that $\delta \subseteq Q \times \overline{\Sigma} \times \{1, 0\} \times Q$.

Whenever understood we will only tacitly mention Σ, $\triangleright$, and $\triangleleft$ in $\mathcal{A}$.

The notion of runs of 2FA on an input string is exactly the same as that of Turing machines on a read-only input tape. More precisely, for a string $w = a_1 \dots a_n$, a *run* of $\mathcal{A}$ on w is a sequence of pairs $(q_0, i_0), \dots, (q_m, i_m) \in Q \times [0, n+1]$ defined as follows. Let $a_0 = \triangleright$ and $a_{n+1} = \triangleleft$. The following conditions then have to be satisfied: $i_0 = 0$, and for every $j \in [0, m-1]$, we have $(q_j, a_{i_j}, dir, q_{j+1}) \in \delta$ and $i_{j+1} = i_j + dir$ for some $dir \in \{-1, 0, 1\}$.

The run is said to be *accepting* if $i_m = n + 1$ and $q_m \in F$. A string w is *accepted* by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on w. The set of strings accepted by $\mathcal{A}$ is denoted by $\mathcal{L}(\mathcal{A})$, a.k.a., the language *recognised* by $\mathcal{A}$. The *size* $|\mathcal{A}|$ of $\mathcal{A}$ is defined to be $|Q|$; this will be needed when we talk about computational complexity. For convenience, we will also refer to an FA without initial and final states, that is, a pair (Q, δ), as a *transition graph*.

2FA and FA recognise precisely the same class of languages, i.e., *regular languages*. The following result is standard and can be found in textbooks on automata theory (e.g. [30]).

Proposition 1. *Every 2FA $\mathcal{A}$ can be transformed in exponential time into an equivalent FA of size $2^{\mathcal{O}(|\mathcal{A}| \log |\mathcal{A}|)}$. Moreover, every FA can be transformed in polynomial time into an ε-free FA, that is, an FA where $\delta \subseteq Q \times \overline{\Sigma} \times \{1\} \times Q$.*

For latter reference, here we briefly mention the construction of FAs from 2FAs in [30]: For each 2FA $\mathcal{A} = (Q, q_0, F, \delta)$, each state of the equivalent FA is a vector of states from Q of *odd* length, say $(q_1, \dots, q_{2n+1})$, such that all the states of even (resp. odd) indices are mutually distinct.

In the rest of this paper, FAs refer to ε-free FAs where directions in transitions are omitted. Moreover, for simplicity of notations, in FAs, we omit the two end markers $\triangleright, \triangleleft$ and assume that $\delta \subseteq Q \times \Sigma \times \{1, 0\} \times Q$.

Operations of FAs. For an FA $\mathcal{A} = (Q, q_0, F, \delta)$, $q \in Q$ and $P \subseteq Q$, we use $\mathcal{A}(q, P)$ to denote the FA (Q, q, P, δ), that is, the FA obtained from $\mathcal{A}$ by changing the initial state and the set of final states to q and P respectively. We use $q \xrightarrow[\mathcal{A}]{w} q'$ to denote the fact that a string w is accepted by $\mathcal{A}(q, \{q'\})$.

Given two FAs $\mathcal{A}_1 = (Q_1, q_{0,1}, F_1, \delta_1)$ and $\mathcal{A}_2 = (Q_2, q_{0,2}, F_2, \delta_2)$, the *product* of $\mathcal{A}_1$ and $\mathcal{A}_2$, denoted by $\mathcal{A}_1 \times \mathcal{A}_2$, is defined as $(Q_1 \times Q_2, (q_{0,1}, q_{0,2}), F_1 \times F_2, \delta_1 \times \delta_2)$, where $\delta_1 \times \delta_2$ is the set of tuples $((q_1, q_2), a, (q_1', q_2'))$ such that $(q_1, a, q_1') \in \delta_1$ and $(q_2, a, q_2') \in \delta_2$. Evidently, we have $\mathcal{L}(\mathcal{A}_1 \times \mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$.

Computational complexity. We study the complexity of the path feasibility problem for programs with strings. We will deal with the complexity classes (e.g. see [30]): EXPSPACE (problems solvable in exponential space), n-EXPSPACE (problems solvable in n-fold exponential space).

3 Parametric Transducers and the Constraint Language

We have defined the framework of Björner *et al.* [10] of constraints from symbolic executions in the Introduction. For ease of reading, we provide the more specific grammar for symbolic executions S satisfying the *straight-line restriction* from Lin and Barcelo [39] that we already discussed in the Introduction (i.e. only regular constraints are allowed in an assertion):

$$S ::= \quad y := f(x_1, \dots, x_r) \mid \mathbf{assert}(x \in L) \mid S_1; S_2 \tag{3}$$

where $f : (\Sigma^*)^r \to \Sigma^*$ is a (partial) string function, and L is a regular language.

3.1 Non-parametric transducers

Definition 2 (Finite-State Transducers). *Let Σ be an alphabet. A* nondeterministic two-way finite transducer *(2FT) T over Σ is a tuple $(\Sigma, \triangleright, \triangleleft, Q, q_0, F, \delta)$, where δ is a finite subset of $Q \times \overline{\Sigma} \times \{-1, 0, 1\} \times Q \times \Sigma^*$, satisfying the syntactical constraints of 2FAs, and the additional constraint that the output must be ϵ when reading $\triangleright$ or $\triangleleft$. Formally, for each transition $(q, \triangleright, dir, q', w)$ or $(q, \triangleleft, dir, q', w)$ in δ, we have $w = \epsilon$. A (nondeterministic one-way) finite transducer (FT) over Σ is a 2FT such that $\delta \subseteq Q \times \overline{\Sigma} \times \{1, 0\} \times Q \times \Sigma^*$.*

The notion of runs of 2FTs on an input string can be seen as a generalisation of 2FAs by adding outputs. More precisely, given a string $w = a_1 \dots a_n$, a *run* of T on w is a sequence of tuples $(q_0, i_0, w'_0), \dots, (q_m, i_m, w'_m) \in Q \times [0, n+1] \times \Sigma^*$ such that, if $a_0 = \triangleright$ and $a_{n+1} = \triangleleft$, we have $i_0 = 0$, and for every $j \in [0, m-1]$, $(q_j, a_{i_j}, dir, q_{j+1}, w'_j) \in \delta$, $i_{j+1} = i_j + dir$ for some $dir \in \{-1, 0, 1\}$, and $w'_0 = w'_m = \varepsilon$. The run is said to be *accepting* if $i_m = n + 1$ and $q_m \in F$. When a run is accepting, $w'_0 \cdots w'_m$ is said to be the *output* of the run. Note that some of these w'_i's could be empty strings. A word w' is said to be an output of T on w if there is an accepting run of T on w with output w'. We use $\mathcal{T}(T)$ to denote the *transduction* defined by T, that is, the relation comprising the pairs (w, w') such that w' is an output of T on w. In this paper, we are mainly interested in *functional transducers*, i.e., transducers that define functions instead of relations. (For instance, deterministic transducers are always functional.)

To take into consideration the outputs of transitions, we define the *size* $|T|$ of T as the sum of the sizes of transitions in T, where the size of a transition (q, a, dir, q', w') is defined as $|w'| + 1$. With the assumption that only ϵ can be outputted when reading $\triangleright$ or $\triangleleft$, for simplicity, we omit the two end markers $\triangleright$ and $\triangleleft$ in FTs, that is, $\delta \subseteq Q \times \Sigma \times \{1, 0\} \times Q \times \Sigma^*$. Note that each accepting run of an FT has to read through the input string and stop in the right end.

Example 1. We give two examples of 2FTs: (1) the function **escapeString**, and (2) the function **reverse**. The first function backslash-escapes every occurrence of `'` and `"`. This can be modelled by an FT with a single state q_0, transitions $(q_0, \ell, 1, q_0, \ell)$ for each $\ell \neq$ `'` or `"`, $(q_0, \texttt{'}, 1, q_0, \backslash\texttt{'})$, $(q_0, \texttt{"}, 1, q_0, \backslash\texttt{"})$, and the final state q_0. The second function requires us to use a 2FT, which we will only describe informally. The transducer T_{reverse} will firstly move to $\triangleleft$ of the input tape without outputting any symbol. It then scans the input from right to left, while outputting the symbol that is read. When $\triangleright$ is reached, it moves its input head right to $\triangleleft$ (without outputting any further symbol), and accepts. □

3.2 Parametric transducers

We introduce parametric transducers, a transducer model that can capture a wide range of string functions (e.g., all of what have discussed so far).

Definition 3 (Parametric Transducers). *Let Σ be an alphabet, and X be a set of* parameters*. Assume that $\Sigma \cap X = \emptyset$ and let $\Gamma = \Sigma \cup X$. A (nondeterministic) two-way* parametric transducer *(2PT) T over (Σ, X) is a 2FT extended with*

parameter outputs, more precisely, T is a tuple $(\Sigma, X, \triangleright, \triangleleft, Q, q_0, F, \delta)$, where δ is a finite subset of $Q \times \overline{\Sigma} \times \{-1, 0, 1\} \times Q \times \Gamma^*$, satisfying the syntactical constraints of 2FTs. A (nondeterministic one-way) parametric transducer (PT) over (Σ, X) is a 2PT such that $\delta \subseteq Q \times \overline{\Sigma} \times \{0, 1\} \times Q \times \Gamma^*$.

Notice that parameters are only allowed in the output tape. Intuitively, each instantiation of the parameters $x_1, \ldots, x_r$ with strings $w_1, \ldots, w_r \in \Sigma^*$ in a 2PT T gives rise to a 2FT which is obtained from T by replacing each occurrence of x_j in transitions with w_j. We will use $T[w_1/x_1, \ldots, w_r/x_r]$ to denote the 2FT resulting from the instantiation, and $\mathcal{T}(T)$ to denote the set of string tuples $(w, w_1, \ldots, w_r, w')$ such that $(w, w') \in \mathcal{T}(T[w_1/x_1, \ldots, w_r/x_r])$. Similarly to FTs, we also omit $\triangleright, \triangleleft$ in PTs, that is, $\delta \subseteq Q \times \Sigma \times \{0, 1\} \times Q \times \Gamma^*$.

Example 2. The concatenation operation $x \cdot y$ can be simulated by a PT T_{concat} that takes x as the input, output each letter it reads, and output the parameter y in the right end. Specifically, the PT has two states q_0, q_1, transitions $(q_0, \ell, 1, q_0, \ell)$ and $(q_0, \ell, 1, q_1, \ell \cdot y)$ for each $\ell \in \Sigma$, initial state q_0, and final state q_1. (Notice that the head of the PT must stop in the right end of the tape to accept.)

As promised in the Introduction, we now demonstrate how the replaceAll function from [15] can be simulated by PTs. Recall $\mathsf{replaceAll}_p(sub, rep)$ where p is a fixed regular expression. In case that p is a single letter $\mathtt{a}$, the transducer for $\mathsf{replaceAll}_p$ has one state q_0 as both the initial and the final state, and transitions $(q_0, \ell, 1, q_0, \ell)$ for each $\ell \in \Sigma \backslash \{\mathtt{a}\}$, and $(q_0, \mathtt{a}, 1, q_0, rep)$. For the more general case, to resolve ambiguity, [15] uses the semantics of *leftmost and longest matching* that is used by languages such as Python and sed (with the `--posix` flag). We refer to [15] for the details. When this semantics is adopted, we can construct the 2PT for $\mathsf{replaceAll}_p$ in exponential time (see Proposition 2). □

Proposition 2. *For the function $\mathsf{replaceAll}_p$ with the pattern p being a regular expression, we can compute a PT $T_{replaceall}$ of size $2^{\mathcal{O}(|p|^c)}$ for some $c > 0$ such that $\mathcal{T}(T_{replaceall})$ expresses $\mathsf{replaceAll}_p$.*

The proof of Proposition 2 can be found in Appendix B.

Context-sensitive auto-sanitisation example. Web pages are often constructed using templating systems [45] such as Mustache.js or Google Closure. Template HTML contains variables that are instantiated before the page is served. A simple template is given in Figure 1(a) with variables `$li` and `$txt`. A parametric transducer can instantiate the variables with runtime values using the variable parameters. However templating systems may also provide a context-aware "auto-escape" feature. That is, the contents of the replaced variable will be "escaped" so that special characters (or sequences) which may allow code injection are replaced with safe alternatives to avoid attack. The escaping applied depends on the text surrounding the variable use.

For example, if `$li` had the value `javascript: alert(1337)`, clicking the link in the instantiated template would run the JavaScript code `alert(1337)`. To avoid this, Closure wraps URLs beginning with `javascript:` in single quotes.

Providing the value `<script>alert(1337)</script>` to `$txt` would have a similar effect, but this time Closure replaces `<` with `&lt;` and `>` with `&gt;`. A key point is that the safe transduction depends on the context of the variable: the escaping in a URL context is different to that in an HTML context. We can model this using two transducers. The first transducer instantiates the variables, but surrounds them with markers to indicate that the text inside needs to be escaped (Figure 1(b)); we use `[URL]`, `[LRU]`, `[HTML]`, and `[LMTH]`. Since transducers are a generalisation of finite automata, they are able to identify contexts using regular languages. The second transducer then applies the appropriate safe transductions inside the markers. For the URL context, values starting with `javascript:` will be quoted, while in HTML, instances of `<` and `>` are replaced by `&lt;` and `&gt;` respectively. The result is shown in Figure 1(c).

(a)
```
<a href="{$li}"> {$txt} </a>
```
(b)
```
<a href="[URL]javascript: alert(1337)[LRU]">
    [HTML]<script>alert(1337)</script>[LMTH] </a>
```
(c)
```
<a href="'javascript: alert(1337)'">
    &amp;lt;script&amp;gt;alert(1337)&amp;lt;/script&amp;gt; </a>
```

Fig. 1. A simple template (a) with two transduction steps (b, c).

4 A generic decision procedure

In this section, we present a generic decision procedure for checking path feasibility of any (straight-line) symbolic execution S that satisfies one further condition: the pre-image of a regular language under any function $f : (\Sigma^*)^r \to \Sigma^*$ in S satisfies a "*regularity condition*". In the case when $r = 1$, this regularity condition would state that the pre-image of a regular language under the function f is *effectively regular*, i.e. an FA can be computed to represent the pre-image of the regular language under f. This is satisfied by FTs [7–9], and there is a decision procedure for S with $f : \Sigma^* \to \Sigma^*$ being an FT, e.g., see [7, 8, 39].

Example 3. Consider the symbolic execution S

$$\textbf{assert}(x \in L_0); \quad y := f(x); \quad z := g(y); \quad \textbf{assert}(y \in L_1); \quad \textbf{assert}(z \in L_2)$$

where $f, g : \Sigma^* \to \Sigma^*$ satisfy the regularity condition. To check the path feasibility of S, we repeatedly *remove the last assignment* from the program by the pre-image computation. More precisely, we compute the pre-image $g^{-1}(L_2)$ of L_2 under g, which by assumption is a regular language L_2', yielding the equi-path-feasible program:

$$\textbf{assert}(x \in L_0); \quad y := f(x); \quad \textbf{assert}(y \in L_1); \quad \textbf{assert}(y \in L_2')$$

Computing the regular pre-image $L_1' := f^{-1}(L_1 \cap L_2')$ yields a program with only two assertions of regular constraints $x \in L_0$ and $x \in L_1'$, whose satisfiability amounts to checking $L_0 \cap L_1' \neq \emptyset$ by an automata algorithm. □

How do we generalise this generic decision procedure to partial functions $f : (\Sigma^*)^r \to \Sigma^*$ with possibly multiple inputs, which at least subsume parametric transducers? One answer is to use a generalisation of the notion of "regularity" from languages to relations by means of *recognisable relations* [17]. (Other more expressive notions of regularity do exist (see [17]), but they are not suitable for generalising the aforementioned decision procedure for $r = 1$.)

Definition 4 (Recognisable relations). *An r-ary relation $R \subseteq \Sigma^* \times ... \times \Sigma^*$ is* recognisable *if $R = \bigcup_{i=1}^{n} L_1^{(i)} \times ... \times L_r^{(i)}$ where $L_j^{(i)}$ is regular for each $j \in [r]$.*

In other words, a recognisable relation is simply a finite union of cartesian products of regular languages. A *representation* of a recognisable relation R is a collection of tuples $(\mathcal{A}_1^{(i)}, ..., \mathcal{A}_r^{(i)})_{i \in [n]}$ such that $L_j^{(i)} = \mathcal{L}(\mathcal{A}^{(i)})_j$ for each j. Each tuple $(\mathcal{A}_1^{(i)}, ..., \mathcal{A}_r^{(i)})$ is called a *disjunct*, and each FA $\mathcal{A}_j^{(i)}$ is called an *atom*.

Theorem 1. *There is a procedure which, given a symbolic execution S wherein each $f : (\Sigma^*)^r \to \Sigma^*$ satisfies the regularity condition (with respect to recognisable relations), decides whether S is path feasible.*

Proof. We provide a nondeterministic procedure which generalises the case for univariate functions f sketched in Example 3. Let S be a symbolic execution, $y := f(\vec{x})$ (where $\vec{x} = x_1, ..., x_r$) be the last assignment in S, and $\sigma := \{y \in L_1, ..., y \in L_s\}$ be the set of all regular constraints on y in assertions of S. Let S' be the program obtained by removing $y := f(\vec{x})$ along with all assertions with conditionals in σ. Compute the regular language $L := L_1 \cap \cdots \cap L_s$ using the product construction. By assumption, we can compute a finite set of disjuncts $(\mathcal{A}_1^{(i)}, ..., \mathcal{A}_r^{(i)})$ to represent $f^{-1}(L)$. Nondeterministically guess one such disjunct $(\mathcal{A}_1, ..., \mathcal{A}_r)$. Let S'' be the program

$$S'; \quad \textbf{assert}(x_1 \in \mathcal{L}(\mathcal{A}_1)); \quad \cdots \quad ; \quad \textbf{assert}(x_r \in \mathcal{L}(\mathcal{A}_r))$$

Then, S is path-feasible iff there is a nondeterministic guess leading to a construction of S'' that is path-feasible. In sum, the procedure will terminate when S is a conjunction of assertions on input variables, which can be checked via language nonemptiness of FA. □

Complexity consideration. To obtain tight complexity bounds, it is crucial to avoid a product construction before each preimage computation in the above algorithm. To this end, we make two modifications.

Firstly, since we assume that each $f : (\Sigma^*)^r \to \Sigma^*$ in S is a partial function, it follows that $f^{-1}(L_1 \cap L_2) = f^{-1}(L_1) \cap f^{-1}(L_2)$. (This is *false* in general if f is a *relation*.) This means that, for each $y := f(\vec{x})$ and $y \in L := L_1 \cap \cdots \cap L_s$, we can compute $f^{-1}(L)$ by *separately* computing the pre-image $f^{-1}(L_i)$ for each regular constraint $y \in L_i$, i.e., no product construction is performed.

The second modification concerns the *representation* of each atom in a recognisable relation. We propose to use (a variant of) conjunctions of regular constraints called *conjunctive FA*. More precisely, each conjunctive FA is a tuple

$\mathcal{A} = ((Q, \delta), \Omega)$ such that (Q, δ) is a transition graph of an FA and $\Omega \subseteq Q \times Q$ is a *conjunctive acceptance condition*, i.e., a string w is accepted by $\mathcal{A}$ if for *every* $(q, q') \in \Omega$, there is a run of (Q, δ) on w from q to q'. Evidently, a conjunctive FA $\mathcal{A} = ((Q, \delta), \Omega)$ is a *succinct* representation of the product of FAs $\mathcal{B}_{q,q'}$, which has a size exponentially large than that of $\mathcal{A}$ in the worst case. Accordingly, a representation of R is called a *conjunctive representation* if every atom in the representation is a conjunctive FA. Since each FA $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ can be assumed to have only one final state (i.e. $F = \{q_F\}$), each FA can be identified with the conjunctive FA $((Q, \delta), \{(q_0, q_F)\})$. The *size* of a conjunctive representation $((Q, \delta), \Omega)$ is defined as $|Q|$, and the *atom size* of a conjunctive representation of R is defined to be the maximum size of the atoms therein.

We now provide a refined version of Theorem 1 that accounts for complexity (full proof can be found in Appendix A). First, the regularity condition **PreRec** now incorporates conjunctive FAs as representations of atoms: *for each function f in S and each conjunctive FA $\mathcal{A}$, there is an algorithm that runs in $\ell(|f|, |\mathcal{A}|)^{c_0}$ space which enumerates each disjunct of a conjunctive representation of $f^{-1}(\mathcal{A})$, whose atom size is bounded by $\ell(|f|, |\mathcal{A}|)$, where $\ell(\cdot, \cdot)$ is a monotone function and $c_0 > 0$ is a constant.* Here $|f|$ is the size of a representation of f, which will be defined when we instantiate the generic decision procedure in Section 5. We also define several size parameters of S. Let $\mathsf{asgn}(S)$ be the number of assignments in S, $\mathsf{art}(S)$ be the maximum arity of string functions in the assignments of S, $\mathsf{fnsize}(S)$ be the maximum size of the representations of these string functions, $\mathsf{asrt}(S)$ be the number of assertions in S, and $\mathsf{fasize}(S)$ be the maximum size of the FAs appearing in the assertions of S. Moreover, for $\ell : \mathbb{N}^2 \to \mathbb{N}$, we define $\ell^{\langle n \rangle}(j, k)$ $(n \geq 1)$ as $\ell^{\langle 1 \rangle}(j, k) = \ell(j, k)$ and $\ell^{\langle n+1 \rangle}(j, k) = \ell(j, \ell^{\langle n \rangle}(j, k))$.

Theorem 2. *Given a symbolic execution S satisfying **PreRec**, the path feasibility problem of S can be decided* nondeterministically *with space*

$$(\mathsf{art}(S) + 1)^{\mathsf{asgn}(S)} \mathsf{asrt}(S) \left(\ell^{\langle \mathsf{asgn}(S) \rangle}(\mathsf{fnsize}(S), \mathsf{fasize}(S))\right)^{O(1)}$$

5 Instantiations of the generic decision procedure

In our framework each string function $f(x_1, \cdots, x_r)$ $(r \geq 1)$ from the symbolic execution S is modelled by a parametric transducer T with $r-1$ parameters, and the size of f, $|f|$, is naturally defined by the size of T (i.e., $|T|$). For the pre-image of an FA $\mathcal{A}$, $f^{-1}(\mathcal{A})$ is instantiated by $\mathsf{Pre}_T(\mathcal{A})$, i.e., the set of tuples $(w_1, \ldots, w_r)$ such that there is an accepting run of T on w_1 with output $w' \in \mathcal{L}(\mathcal{A})$ when the parameters are instantiated with $w_2, \ldots, w_r$. To apply Theorem 2, one verifies the regularity condition **PreRec** for the respective class of parametric transducers.

5.1 Two-way transducers

Lemma 1. *The regularity condition **PreRec** holds for string functions definable in 2PTs, with $\ell(|T|, |\mathcal{A}|) = 2^{\mathcal{O}(|T||\mathcal{A}| \log(|T||\mathcal{A}|))}$.*

Proof. Assume a 2PT $T = (X, Q, q_0, F, \delta)$ with $X = \{x_1, \cdots, x_r\}$ and a conjunctive FA $\mathcal{A} = ((Q', \delta'), \Omega')$ with $\Omega' \subseteq Q' \times Q'$. Let $\Gamma = \Sigma \cup X$.

To compute $\mathsf{Pre}_T(\mathcal{A})$, we construct a conjunctive 2FA $\mathcal{B}_{T,\mathcal{A},\Omega_{x_1},\cdots,\Omega_{x_r},q}$, where $q \in F$ and $\Omega_{x_1}, \cdots, \Omega_{x_r} \subseteq Q' \times Q'$. Intuitively this is done from T by checking for an accepting run of T on w ending at state q, and simulating an accepting run of $\mathcal{A}$ on the output of T. States of $\mathcal{B}_{T,\mathcal{A},\Omega_{x_1},\cdots,\Omega_{x_r},q}$ comprise $(q, q') \in Q \times Q'$. When a transition of T outputs a string w' from Γ^* where some parameter x_i appears (instead of a constant string), for each occurrence of x_i in w', a pair of states (q'_1, q'_2) is nondeterministically selected from Ω_{x_i}, and the second component of the state of $\mathcal{B}_{T,\mathcal{A},\Omega_{x_1},\cdots,\Omega_{x_r},q}$ will be updated from q'_1 to q'_2 to mimic a possible instantiation of x_i. More precisely, $\mathcal{B}_{T,\mathcal{A},\Omega_{x_1},\cdots,\Omega_{x_r},q}$ is defined as $\mathcal{A}'' = ((Q'', \delta''), \Omega'')$, where $Q'' = Q \times Q'$, $\Omega'' = \{((q_0, p), (q, p')) \mid (p, p') \in \Omega'\}$, and the transition relation δ'' comprises the tuples $((q_1, q'_1), a, dir, (q_2, q'_2))$ such that there is $w' \in \Gamma^*$ satisfying that $(q_1, a, dir, q_2, w') \in \delta$, and either

- $w' \in \Sigma^*$ and $q'_1 \xrightarrow[\mathcal{A}]{w'} q'_2$, or
- suppose $w' = w'_1 x_{i_1} \ldots w'_s x_{i_s} w'_{s+1}$ for some $s \geq 1$, $w'_j \in \Sigma^*$ for each $j \in [s+1]$, and $i_j \in [r]$ for each $j \in [s]$, then there are states $p_0, p_1, \ldots, p_{2s+1} \in Q'$ satisfying that $p_0 = q'_1$, $p_{2s+1} = q'_2$, and

$$p_0 \xrightarrow[\mathcal{A}]{w'_1} p_1 \xrightarrow{\Omega_{x_{i_1}}} p_2 \ \cdots \ p_{2s-2} \xrightarrow[\mathcal{A}]{w'_s} p_{2s-1} \xrightarrow{\Omega_{x_{i_s}}} p_{2s} \xrightarrow[\mathcal{A}]{w'_{s+1}} p_{2s+1},$$

  where $p_{2j-1} \xrightarrow{\Omega_{x_{i_j}}} p_{2j}$ means $(p_{2j-1}, p_{2j}) \in \Omega_{x_{i_j}}$ for each $j \in [s]$.

It is easy to see that the size of $\mathcal{B}_{T,\mathcal{A},\Omega_{x_1},\cdots,\Omega_{x_r},q}$ is at most $|T||\mathcal{A}|$ and $|\Omega''| = |\Omega'|$.

Claim. $(w, w_1, \cdots, w_r) \in \mathsf{Pre}_T(\mathcal{A})$ iff there are $\Omega_{x_1}, \cdots, \Omega_{x_r} \subseteq Q' \times Q'$ and $q \in F$ such that $w \in \mathcal{L}(\mathcal{B}_{T,\mathcal{A},\Omega_{x_1},\cdots,\Omega_{x_r},q})$ and $w_i \in \mathcal{L}(\mathcal{A}[\Omega_{x_i}])$ for each $i \in [r]$, where $\mathcal{A}[\Omega_{x_i}] = ((Q', \delta'), \Omega_{x_i})$. [The proof of the claim is given in Appendix C; it crucially relies on that T is functional otherwise it would *not* be correct.]

Thus, $\mathsf{Pre}_T(\mathcal{A})$ is a recognisable relation equal to

$$\bigcup_{\Omega_{x_1},\cdots,\Omega_{x_r} \subseteq Q' \times Q', q \in F} \mathcal{L}(\mathcal{B}_{T,\mathcal{A},\Omega_{x_1},\cdots,\Omega_{x_r},q}) \times \mathcal{L}(\mathcal{A}[\Omega_{x_1}]) \times \cdots \times \mathcal{L}(\mathcal{A}[\Omega_{x_r}]).$$

To represent $\mathsf{Pre}_T(\mathcal{A})$, we transform the conjunctive 2FA $\mathcal{B}_{T,\mathcal{A},\Omega_{x_1},\cdots,\Omega_{x_r},q}$ to a union of conjunctive FAs as follows: Transform $\mathcal{A}''$ into a one-way transition graph (Q''', δ'''), where Q''' are vectors of states of $\mathcal{A}''$ of lengths at most $2|Q''|-1$ (cf. Proposition 1). Then $\mathcal{L}(\mathcal{B}_{T,\Omega_{x_1},\cdots,\Omega_{x_r},q})$ is the union of $\mathcal{L}(((Q''', \delta'''), \Omega''))$ for $\Omega'' \subseteq Q''' \times Q'''$, which comprises nondeterministically selected pairs $(\vec{\rho}_1, \vec{\rho}_2) \in Q''' \times Q'''$, one for each $(p, p') \in \Omega'$, such that $\vec{\rho}_1[1] = (q_0, p)$ and $\vec{\rho}_2[|\vec{\rho}_2|] = (q, p')$.

Since the size of each conjunctive FA $((Q''', \delta'''), \Omega'')$ is $2^{\mathcal{O}(|T||\mathcal{A}| \log(|T||\mathcal{A}|))}$, we conclude that the **PreRec** assumption holds for string functions definable in 2PTs with $\ell(|T|, |\mathcal{A}|) = 2^{O(|T||\mathcal{A}| \log(|T||\mathcal{A}|))}$. □

Lemma 1 and Theorem 2 entail one can decide the path feasibility of a symbolic execution S with 2PTs in nondeterministic

$$\mathrm{Tower}(\mathsf{asgn}(S), \mathcal{O}(\mathsf{fnsize}(S)\mathsf{fasize}(S)\log(\mathsf{fnsize}(S)\mathsf{fasize}(S))))$$

space (which can be even made in *deterministic* space by Savitch's theorem [46]). Here $\mathrm{Tower}(1,k) = 2^k$ and $\mathrm{Tower}(j+1,k) = 2^{\mathrm{Tower}(j,k)}$. After the result we sketch the lower bound.

Theorem 3. *Given a symbolic execution S with 2PTs, the path feasibility of S can be decided in $\mathsf{asgn}(S)$-exponential space.*

5.2 k-Reversal-bounded transducers

In practice, many string functions need only a small number of reversals for the reading head, rather than full two-way transducers. E.g., the reading head for one-way PT (resp. the string reverse function) needs 0 (resp. 2) reversal(s).

Let $k \in \mathbb{N}$. A k-reversal bounded 2PT T (k-RB2PT) is such that in every run of T, the reading direction is reversed at most k times. Notice that k must be even since accepting runs must stop on the right end of input strings. Evidently, 0-reversal bounded 2PTs are precisely one-way PTs.

Lemma 2. *The regularity condition **PreRec** holds for string functions definable in k-RB2PTs, with $\ell(|T|, |\mathcal{A}|) = (|T||\mathcal{A}|)^{k+1}$.*

We follow the construction in the proof of Lemma 1. One can see that the size of 2FA $\mathcal{B}_{T,\mathcal{A},\Omega_{x_1},\cdots,\Omega_{x_r},q}$ is bounded by $|T||\mathcal{A}|$. Moreover, as T is k-reversal bounded, so is $\mathcal{B}_{T,\mathcal{A},\Omega_{x_1},\cdots,\Omega_{x_r},q}$. Therefore, according to the construction of FA from 2FA (cf. [30]), we have that the size of each obtained conjunctive FA in the recognisable relation (cf. the proof of Lemma 1) is at most $(|T||\mathcal{A}|)^{k+1}$.

Combing Theorem 2 and Lemma 2, we have:

Theorem 4. *Let $k \in \mathbb{N}$. The path feasibility of string constraints with k-RB2PTs is in 2-*EXPSPACE*.*

To see Theorem 4, we invoke Theorem 2 again. One can observe that the space required by the instantiation of the generic procedure is

$$\mathcal{O}(\mathsf{fnsize}(S)^{(k+1)+(k+1)^2+\ldots+(k+1)^m} \cdot \mathsf{fasize}(S)^{(k+1)^m})$$

(where $m = \mathsf{asgn}(S)$), which is doubly exponential in the size of S.

We are also interested in the question whether one could obtain lower complexity bounds for more specific string functions. In particular, we will examine PTs, reverse function modelled by T_{reverse} (cf. Example 1), concatenation and replaceall function modelled by T_{concat} and $T_{\mathsf{replaceAll}_p}$ respectively (cf. Example 2). To instantiate the generic decision procedure for these functions, we verify the regularity condition **PreRec** for them as follows. One may notice that although T_{reverse} is a two-way transducer, its pre-image does *not* bring an exponential blowup as general 2PTs would have.

Lemma 3. *The regularity condition **PreRec** holds for string functions definable in PTs or T_{reverse}, with $\ell(|T|, |\mathcal{A}|) = |T||\mathcal{A}|$.*

The proof is the same as Lemma 1 except constructing $\mathcal{B}_{T,\mathcal{A},\Omega_{x_1},\ldots,\Omega_{x_r},q}$ is simpler, with size at most $|T||\mathcal{A}|$, since the transformation from conjunctive 2FA to conjunctive FA is not needed. For $T_{reverse}$, the pre-image is given by reversing the direction of each transition of $\mathcal{A}$ and swapping initial and final states.

Furthermore, Proposition 2 entails that $T_{\mathsf{replaceAll}_p}$ is of size $2^{\mathcal{O}(|p|^c)}$ for some constant $c > 0$. It follows from Theorem 2 and Lemma 3 that we have:

Theorem 5. *The path feasibility of string constraints with PTs, replaceall, and reverse is* EXPSPACE*-complete.*

We remark the above logic also includes concatenation. The lower bound is from the EXPSPACE-hardness of path feasibility of straight-line string constraints with concatenation and one-way (non-parametric) transducers, both of which are special cases of PTs [39].

6 Symbolic extensions

In this section, we consider an extension of the work in preceding sections to the symbolic setting, where finite-state automata and parametric transducers are replaced by *symbolic automata* [19, 54] and *symbolic parametric transducers* (to be introduced) respectively. This extension is important for practical applications like analysis of XSS vulnerabilities in web applications since the transducers representing sanitisers have to operate over large alphabets (e.g. UTF16), e.g., see [18, 29, 55]. Intuitively, in symbolic automata input letters are symbolically described by unary predicates from a logical theory (called background theory). Likewise, in symbolic parametric transducers output letters are given by terms in the logic theory or by parameters (as in the parametric transducers). Symbolic parametric transducers can be seen as extensions of both parametric transducers introduced in this paper and symbolic transducers in [55].

A concept of background theories will be used in the definition of symbolic automata and symbolic parametric transducers.[3] Intuitively, a background theory is a set of unary first-order logic formulae closed under Boolean connectives and substitutions.

Definition 5 (Background theories). *A background theory Υ is a tuple $(\mathcal{S}, \mathcal{D}, \Psi)$ such that $\mathcal{S} = (\mathcal{F}, \mathcal{P})$ is a signature, where $\mathcal{F}$ (resp. $\mathcal{P}$) is a recursively enumerable set of function symbols (resp. relation symbols), $\mathcal{D}$ is an $\mathcal{S}$-structure $(\mathbb{D}, \mathcal{I})$, where $\mathbb{D}$ is an at most countable set (the data domain), and $\mathcal{I}$ is an interpretation of $\mathcal{S}$ on $\mathbb{D}$, Ψ is a recursively enumerable set of* unary *$\mathcal{S}$-formulae closed under Boolean connectives $\vee, \wedge, \neg$ and substitutions, that is, for each $\mathcal{S}$-term $t(x)$ and $\psi(x) \in \Psi$, we have $\psi[t(x)/x] \in \Psi$. For each $\psi(x) \in \Psi$, we use $\|\psi\|$ to denote the set of* witnesses *of ψ, i.e. $\{d \in \mathbb{D} \mid \mathcal{D} \models \psi(d)\}$. A formula $\psi \in \Psi$ is* satisfiable*, if $\|\psi\| \neq \emptyset$. A background theory $\Upsilon = (\mathcal{S}, \mathcal{D}, \Psi)$ is* decidable *if the satisfiability of $\psi \in \Psi$ is decidable.*

[3] In literature, symbolic automata are defined on a Boolean algebra, which is a background theory without requiring the set of formulae to be closed under substitutions. In this paper, we define symbolic automata on background theories (rather than Boolean algebra) to align to the definition of symbolic parametric transducers.

In the rest of this section, we **fix a decidable background theory** $\Upsilon = (\mathcal{S}, \mathcal{D}, \Psi)$ with $\mathcal{S} = (\mathcal{F}, \mathcal{P})$ and $\mathcal{D} = (\mathbb{D}, \mathcal{I})$. A *data string* is an element of $\mathbb{D}^*$. A *data language* is a set of data strings.

In the sequel, for brevity, we first define symbolic parametric transducers, and then define symbolic transducers as symbolic parametric transducers without parameters and symbolic automata as symbolic transducers without outputs.

Definition 6 (Symbolic parametric transducers). *A nondeterministic two-way symbolic parametric transducer (2SPT) is an extension of a 2ST with parameters. More precisely, a 2SPT T is a tuple $(\rhd, \lhd, Y, Q, q_0, F, \delta)$ where $\rhd$, $\lhd$, Q, q_0, F are defined precisely as in 2FAs, $Y = \{y_1, \dots, y_m\}$ is a finite set of parameters, δ is a finite set of transitions in one of the following forms: 1) symbolic transitions $(q, \psi(x), dir, q', \vec{t})$ such that $q, q' \in Q$, $\psi(x) \in \Psi$, $dir \in \{-1, 1, 0\}$, and $\vec{t} = t_1 \dots t_s$ such that for each $i \in [s]$, either t_i is an $\mathcal{S}$-term where only the variable x occurs, or $t_i \in Y$, 2) non-symbolic transitions $(q, \rhd, dir, q', \epsilon)$, $(q, \lhd, dir', q', \epsilon)$, with $dir \in \{1, 0\}$ and $dir' \in \{-1, 0\}$. A* symbolic transducer *(2ST) is a 2SPT where $Y = \emptyset$ (then Y is omitted). A* symbolic automaton *(2SA) $\mathcal{A}$ is a symbolic transducer where each transition has the empty output (then the outputs are omitted). An SPT (resp. ST, SA) is a 2SPT(resp. 2ST, 2SA) that has no transitions with the direction "-1".*

Semantics of 2SPT. For a 2SPT T with parameters $y_1, \dots, y_m$, each instantiation of $y_1, \dots, y_m$ with data strings $w_1, \dots, w_m$ gives rise to a 2ST, denoted with $T[w_1/y_1, \dots, w_m/y_m]$, by replacing each occurrence of y_i with w_i in the transitions of T. Therefore, we obtain the semantics of 2SPTs from that of 2STs, which will be defined below.

Let $T = (\rhd, \lhd, Q, q_0, F, \delta)$ be a 2ST. A transition $\tau = (q_1, \psi(x), dir, q_2, \vec{t}(x)) \in \delta$ with $\vec{t}(x) = t_1(x) \dots t_r(x)$ in the 2ST T can be concretised into a set of *concrete* transitions $\|\tau\| \subseteq Q \times \mathbb{D} \times \{-1, 0, 1\} \times Q \times \mathbb{D}^r$, where $(q_1, d, dir, q_2, d'_1 \dots d'_r) \in \|\tau\|$ iff $d \in \|\psi\|$ and $d'_j = \mathcal{I}(t_j)(d)$ for each $j \in [r]$. Given a data string $w = d_1, \dots, d_n$, a *run* of T on w is a sequence of tuples $(q_0, i_0, \vec{d'_0}), \dots, (q_m, i_m, \vec{d'_m}) \in Q \times [0, n+1] \times \mathbb{D}^*$ such that $i_0 = 0$, and for every $j \in [m-1]$, one of the following holds,

- $0 < i_j < n+1$, there are $\psi(x) \in \Psi$, $dir \in \{-1, 0, 1\}$, and a sequence of $\mathcal{S}$-terms $\vec{t}(x)$ such that $\tau = (q_j, \psi(x), dir, q_{j+1}, \vec{t}(x)) \in \delta$, $(q_j, d_{i_j}, dir, q_{j+1}, \vec{d'_j}) \in \|\tau\|$, and $i_{j+1} = i_j + dir$,
- $i_j = 0$, $(q_j, \rhd, dir, q_{j+1}, \epsilon) \in \delta$, $\vec{d'_j} = \epsilon$, and $i_{j+1} = i_j + dir$,
- $i_j = n+1$, $(q_j, \lhd, dir, q_{j+1}, \epsilon) \in \delta$, $\vec{d'_j} = \epsilon$, and $i_{j+1} = i_j + dir$.

The run is said to be *accepting* if $i_m = n+1$ and $q_m \in F$. When a run is accepting, $\vec{d'_0}, \dots, \vec{d'_m}$ is the *output* of the run. A data string w' is said to be an output of T on w if there is an accepting run of T on w with output w'. We use $\mathcal{T}(T)$ to denote the *transduction* defined by T, that is, the relation comprising the data-string pairs (w, w') such that w' is an output of T on w.

Let T be a 2SPT with parameters $y_1, \dots, y_m$. We use $\mathcal{T}(T)$ to denote the set of tuples $(w, w_1, \dots, w_m, w')$ such that $(w, w') \in \mathcal{T}(T[w_1/y_1, \dots, w_m/y_m])$.

Symbolic extension of the generic decision procedure. We consider the path feasibility problem of symbolic executions of programs with data strings, where each assertion is of the form $\mathbf{assert}(x \in \mathcal{A})$ with $\mathcal{A}$ being an SA. To check the path feasibility of such programs, we generalise the generic decision procedure in Section 4 by replacing the recognisable relation with the *symbolic recognisable relation*. Due to space limit, we do not address the complexity of the decision procedure here. More details can be found in Appendix E.

Definition 7 (Symbolic recognisable relations). *An r-ary relation $R \subseteq \mathbb{D}^* \times \cdots \times \mathbb{D}^*$ is a* symbolic recognisable *relation if $R = \bigcup_{i=1}^{n} L_1^{(i)} \times \cdots \times L_r^{(i)}$ where $L_j^{(i)}$ is recognised by some SA for each $j \in [r]$.*

Similar to the recognisable relation, we represent an r-ary symbolic recognisable relation as a collection of tuples $(\mathcal{A}_1, \ldots, \mathcal{A}_r)$, where each atom $\mathcal{A}_i$ is an SA. Accordingly, the regularity condition in Section 4 is replaced by the *symbolic* regularity condition, i.e., for each function $f : (\mathbb{D}^*)^r \to \mathbb{D}^*$, the pre-image of a language definable by an SA under f is a symbolic recognisable relation, a representation of which can be computed effectively. Along with the same line as in Section 4–5 via symbolic recognisable relations and symbolic regularity conditions, our main result is as follows.

Theorem 6. *The path feasibility of symbolic executions of programs with 2SPTs is decidable.*

7 Related Work and Future Work

We conclude with a discussion of related work and future work, focussing on (1) decidability, and (2) heuristics and implementation.

Decidability. *Length constraints* — i.e. an assertion $\varphi(\texttt{len}(x_1), \ldots, \texttt{len}(x_n))$, where φ is a Presburger formula and $\texttt{len}(x_i)$ is an integer variable interpreted as the length of the string x_i — have been studied in the context of string solving. It is a major open problem whether the theory of concatenation with length constraints is decidable [24]. Several extensions of this theory are undecidable (e.g. with letter counting [11] and string-number conversion [23]). Several decidable restrictions, however, have been proposed including solved form [24] and acyclicity [2], both of which (like straight-line constraint) impose syntactic restrictions on the way in which string equality can be used in the constraints. It was shown in [39] the decidability of path feasibility for symbolic executions allowing FT and concatenation in the assignments, and regular constraints, and length constraints in the assertions. If we allow the functions $\mathsf{replaceAll}_p(sub, rep)$ in the assignments (instead of FT/concatenation) and length constraints as assertions, path feasibility becomes undecidable [15]. This also implies undecidability of allowing length constraints in our constraint language with parametric transducers. Fortunately, decidability can be easily recovered in some cases. One such case is when the length constraints φ has only one string variable x_1, e.g., $\texttt{len}(x_1) > 7$. In this case, $\varphi(\texttt{len}(x_1))$ can be turned into a regular constraint $x_1 \in L$ for some

L. [This is because the set of integer solutions is effectively a finite union of arithmetic progressions $\bigcup_{i=1}^{n}(a_i + b_i\mathbb{N})$ (where $a_i + b_i\mathbb{N} := \{a_i + b_i n : n \in \mathbb{N}\}$), and each $\texttt{len}(x_i) \in (a_i + b_i\mathbb{N})$ is equivalent to the regular constraint $x_i \in \Sigma^{a_i}(\Sigma^{b_i})^*$.]

The complexity of the theory of concatenation with regular constraints is known to be PSPACE-complete [31, 42]. The complexity of the straight-line logic with concatenation, finite transducers, and regular constraints is EXPSPACE-complete [39]. The same complexity holds when we swap finite transducers with replaceall [15]. For functions, our logic strictly subsumes these two logics (e.g. since it can also express string reverse) and has precisely the same complexity EXPSPACE. One future research avenue is to identify *larger subclasses* of constraints with parametric transducers that are still solvable in the same complexity class.

In this paper, we have combined two powerful formalisms (two-way finite transducers and replaceall) into a single formalism. Since we are considering only two-way transducers that define functions, they are equivalent to two-way deterministic finite transducers, streaming transducers, and MSO-definable transductions [3, 4, 22]. On the other hand, one-way transducers that define functions are strictly more expressive than deterministic one-way transducers [9].

Heuristics and implementation. Theoretical algorithms (e.g. Makanin's algorithm [41]) typically do not directly lead to practical solvers. At the same time, the classes of constraints that are required in practice sometimes require string operations that are not covered by decidable string constraint languages. For these reasons, there is a large amount of work on heuristics for developing practical (often incomplete) string solvers, e.g., [1, 2, 5, 10, 28, 34, 38, 43, 47, 51, 52, 56, 59–62]. Some practical heuristics include bounding string lengths (e.g. [10, 34, 47]), induction, overapproximations [59, 60], interpolation [2], and flat automata [1], to name a few. Focusing on semi-algorithms also allows highly expressive (but undecidable) string constraint languages, e.g., recursively defined functions [51, 52]. Recently, the study of decidability of string constraints have also resulted in automata-theoretic algorithms that are amenable to implementation, e.g., acyclic logic with concatenation, regular constraints, and length constraints [2], and straight-line logic with finite transducers (or replaceall), concatenation, and regular constraints [16, 28, 56]. We leave the development of practical heuristics for our more expressive constraint language for future work.

We mention some interesting benchmarks that are available for string constraints. The first one is Kaluza benchmarks [47], which contain string constraints with concatenation, regular constraints, and length constraints. It was shown in [24] that almost all the constraints are already in *solved form* (in particular, also straight-line). Some of these length constraints are also expressible as regular constraints. The second is SLOG benchmarks [56], which contain string constraints with concatenation, (a restricted class of) replaceall, and regular constraints. The third is SLOTH benchmarks [28], which contains string constraints with concatenation, finite transducers, and regular constraints. Most of these benchmarking examples are expressible in our decidable constraint language.

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 *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2017, Barcelona, Spain, June 18-23, 2017*, pages 602–617, 2017.
2. Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Lukás Holík, Ahmed Rezine, Philipp Rümmer, and Jari Stenman. String constraints for verification. In *CAV*, pages 150–166, 2014.
3. Rajeev Alur and Pavol Cerný. Expressiveness of streaming string transducers. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, pages 1–12, 2010.
4. Rajeev Alur and Jyotirmoy V. Deshmukh. Nondeterministic streaming string transducers. In *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part II*, pages 1–20, 2011.
5. Davide Balzarotti, Marco Cova, Viktoria Felmetsger, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Saner: Composing static and dynamic analysis to validate sanitization in web applications. In *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*, pages 387–401, 2008.
6. Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability*, pages 825–885. 2009.
7. Wilmari Bekker and Valentin Goranko. Symbolic model checking of tense logics on rational Kripke models. In *Infinity in Logic and Computation, International Conference, ILC 2007, Cape Town, South Africa, November 3-5, 2007, Revised Selected Papers*, pages 2–20, 2007.
8. Wilmari Bekker and Valentin Goranko. Symbolic model checking of tense logics on rational Kripke models. *CoRR*, abs/0810.5516, 2008.
9. Jean Berstel. *Transductions and Context-Free Languages*. Teubner-Verlag, 1979.
10. Nikolaj Bjørner, Nikolai Tillmann, and Andrei Voronkov. Path feasibility analysis for string-manipulating programs. In *TACAS*, pages 307–321, 2009.
11. J Richard Büchi and Steven Senger. Definability in the existential theory of concatenation and undecidable extensions of this theory. In *The Collected Works of J. Richard Büchi*, pages 671–683. Springer, 1990.
12. Cristian Cadar, Daniel Dunbar, and Dawson Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, pages 209–224, Berkeley, CA, USA, 2008. USENIX Association.
13. Cristian Cadar, Vijay Ganesh, Peter M. Pawlowski, David L. Dill, and Dawson R. Engler. Exe: Automatically generating inputs of death. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 322–335, New York, NY, USA, 2006. ACM.
14. Cristian Cadar and Koushik Sen. Symbolic execution for software testing: Three decades later. *Commun. ACM*, 56(2):82–90, February 2013.
15. 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.

16. Yan Chen. Z3-replaceall. https://github.com/TinyYan/z3-replaceAll, 2018. Referred in Jan 2018.
17. Christian Choffrut. Relations over words and logic: A chronology. *Bulletin of the EATCS*, 89:159–163, 2006.
18. Loris D'Antoni and Margus Veanes. Static analysis of string encoders and decoders. In *VMCAI*, pages 209–228, 2013.
19. Loris D'Antoni and Margus Veanes. Minimization of symbolic automata. In *POPL*, pages 541–554, 2014.
20. Loris D'Antoni and Margus Veanes. The power of symbolic automata and transducers. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, pages 47–67, 2017.
21. Leonardo De Moura and Nikolaj Bjørner. Satisfiability modulo theories: introduction and applications. *Communications of the ACM*, 54(9):69–77, 2011.
22. Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Log.*, 2(2):216–254, 2001.
23. Vijay Ganesh and Murphy Berzish. Undecidability of a theory of strings, linear arithmetic over length, and string-number conversion. *CoRR*, abs/1605.09442, 2016.
24. 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*, pages 209–226, 2012.
25. Patrice Godefroid, Nils Klarlund, and Koushik Sen. Dart: Directed automated random testing. *SIGPLAN Not.*, 40(6):213–223, June 2005.
26. Google. Closure templates. https://developers.google.com/closure/templates/, 2018. Referred January 2018.
27. Mario Heiderich, Jörg Schwenk, Tilman Frosch, Jonas Magazinius, and Edward Z. Yang. mxss attacks: attacking well-secured web-applications by using innerhtml mutations. In *CCS*, pages 777–788, 2013.
28. 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.
29. Pieter Hooimeijer, Benjamin Livshits, David Molnar, Prateek Saxena, and Margus Veanes. Fast and precise sanitizer analysis with BEK. In *USENIX Security Symposium*, 2011.
30. John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.
31. Artur Jez. Word equations in linear space. *CoRR*, abs/1702.00736, 2017.
32. Yehuda Katz and co. Handlebars.js. http://handlebarsjs.com/, 2018. Referred January 2018.
33. Christoph Kern. Securing the tangled web. *Commun. ACM*, 57(9):38–47, 2014.
34. Adam Kiezun, Vijay Ganesh, Shay Artzi, Philip J. Guo, Pieter Hooimeijer, and Michael D. Ernst. HAMPI: A solver for word equations over strings, regular expressions, and context-free grammars. *ACM Trans. Softw. Eng. Methodol.*, 21(4):25, 2012.
35. James C. King. Symbolic execution and program testing. *Commun. ACM*, 19(7):385–394, 1976.

36. Dexter Kozen. *Automata and Computability*. Springer, 1997.
37. Daniel Kroening and Ofer Strichman. *Decision Procedures*. Springer, 2008.
38. 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*, pages 646–662, 2014.
39. Anthony W. 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 '16, pages 123–136. Springer, 2016.
40. Blake Loring, Duncan Mitchell, and Johannes Kinder. Expose: practical symbolic execution of standalone javascript. In *Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software, Santa Barbara, CA, USA, July 10-14, 2017*, pages 196–199, 2017.
41. Gennady S Makanin. The problem of solvability of equations in a free semigroup. *Sbornik: Mathematics*, 32(2):129–198, 1977.
42. Wojciech Plandowski. Satisfiability of word equations with constants is in PSPACE. *J. ACM*, 51(3):483–496, 2004.
43. Gideon Redelinghuys, Willem Visser, and Jaco Geldenhuys. Symbolic execution of programs with strings. In *SAICSIT*, pages 139–148, 2012.
44. Jacques Sakarovitch. *Elements of automata theory*. Cambridge University Press, 2009.
45. Mike Samuel, Prateek Saxena, and Dawn Song. Context-sensitive auto-sanitization in web templating languages using type qualifiers. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 587–600, New York, NY, USA, 2011. ACM.
46. Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, April 1970.
47. 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, 2010.
48. Klaus U. Schulz. Makanin's algorithm for word equations - two improvements and a generalization. In *Word Equations and Related Topics, First International Workshop, IWWERT '90, Tübingen, Germany, October 1-3, 1990, Proceedings*, pages 85–150, 1990.
49. Koushik Sen, Swaroop Kalasapur, Tasneem G. Brutch, and Simon Gibbs. Jalangi: a selective record-replay and dynamic analysis framework for javascript. In *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18-26, 2013*, pages 488–498, 2013.
50. Koushik Sen, Darko Marinov, and Gul Agha. Cute: a concolic unit testing engine for c. In Michel Wermelinger and Harald Gall, editors, *ESEC/SIGSOFT FSE*, pages 263–272, 2005.
51. Minh-Thai Trinh, Duc-Hiep Chu, and Joxan Jaffar. S3: A symbolic string solver for vulnerability detection in web applications. In *CCS*, pages 1232–1243, 2014.
52. Minh-Thai Trinh, Duc-Hiep Chu, and Joxan Jaffar. Progressive reasoning over recursively-defined strings. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I*, pages 218–240. Springer, 2016.

53. Andrew van der Stock, Brian Glas, Neil Smithline, and Torsten Gigler. OWASP Top 10 – 2017. https://www.owasp.org/index.php/Top_10-2017_Top_10, 2017. Referred January 2018.
54. Gertjan van Noord and Dale Gerdemann. Finite state transducers with predicates and identities. *Grammars*, 4(3):263–286, 2001.
55. Margus Veanes, Pieter Hooimeijer, Benjamin Livshits, David Molnar, and Nikolaj Bjørner. Symbolic finite state transducers: algorithms and applications. In *POPL*, pages 137–150, 2012.
56. 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.
57. Chris Wanstrath. Mustache: Logic-less templates. https://mustache.github.io/, 2009. Referred July 2017.
58. Joel Weinberger, Prateek Saxena, Devdatta Akhawe, Matthew Finifter, Eui Chul Richard Shin, and Dawn Song. A systematic analysis of XSS sanitization in web application frameworks. In *ESORICS*, pages 150–171, 2011.
59. Fang Yu, Muath Alkhalaf, and Tevfik Bultan. Stranger: An automata-based string analysis tool for PHP. In *TACAS*, pages 154–157, 2010. Benchmark can be found at http://www.cs.ucsb.edu/ vlab/stranger/.
60. Fang Yu, Muath Alkhalaf, Tevfik Bultan, and Oscar H. Ibarra. Automata-based symbolic string analysis for vulnerability detection. *Form. Methods Syst. Des.*, 44(1):44–70, 2014.
61. Yunhui Zheng, Vijay Ganesh, Sanu Subramanian, Omer Tripp, Julian Dolby, and Xiangyu Zhang. Effective search-space pruning for solvers of string equations, regular expressions and length constraints. In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, pages 235–254, 2015.
62. Yunhui Zheng, Xiangyu Zhang, and Vijay Ganesh. Z3-str: a Z3-based string solver for web application analysis. In *ESEC/SIGSOFT FSE*, pages 114–124, 2013.

A Proof of Theorem 2 in Section 4

In this proof, we will use the following proposition.

Proposition 3. *For a pair of conjunctive FAs* $\mathcal{A}_1 = ((Q_1, \delta_1), S_1)$ *and* $\mathcal{A}_2 = ((Q_2, \delta_2), S_2)$*, the conjunctive FA* $\mathcal{A} = (Q_1 \cup Q_2, \delta_1 \cup \delta_2, S_1 \cup S_2)$ *recognises* $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$.

This is immediate from the definition of acceptance for FA.

Spelling out the algorithm in full. To facilitate the complexity analysis, we will spell out the modified algorithm in detail. Let S be a symbolic execution satisfying **PreRec** . We give a nondeterministic algorithm to decide the path feasibility of S.

By definition, S has $\mathsf{asgn}(S)$ assignments. First, for each variable $x \in \mathsf{Vars}(S)$, define $\mathcal{E}_{\mathsf{asgn}(S)}(x)$ as the set of conjunctive FAs $((Q, \delta), \{(q_0, q_F)\})$ such that $\mathbf{assert}(x \in \mathcal{A})$ occurs in S, where $\mathcal{A} = (Q, q_0, \{q_F\}, \delta)$. Note that at this stage each conjunctive FA in $\mathcal{E}_{\mathsf{asgn}(S)}(x)$ is actually just a normal FA. Let $i \coloneqq \mathsf{asgn}(S)$ and iterate the following procedure until $i = 0$.

Suppose the i-th assignment of S is $y \coloneqq f(x_1, \cdots, x_r)$ with $r \geq 1$ (note here possibly $x_{j_1} = x_{j_2}$ for some $j_1 < j_2$), and $\mathcal{E}_i(y) = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$, where $\mathcal{A}_j$ is a conjunctive FA for each $j \in [n]$. By **PreRec**, we can enumerate each disjunct $(\mathcal{A}^{(\iota)}_{j,1}, \ldots, \mathcal{A}^{(\iota)}_{j,r})$ of a conjunctive representation of $f^{-1}(\mathcal{A}_j)$ in $(\ell(|f|, |\mathcal{A}_j|))^{c_0}$ space, where each atom is of size at most $\ell(|f|, |\mathcal{A}_j|)$. Then $\mathcal{E}_{i-1}(x)$ for $x \in \{x_1, \cdots, x_r\}$ is constructed as follows:

1. For each $j \in [n]$, nondeterministically select a disjunct $\left(\mathcal{A}_{j,1}, \ldots, \mathcal{A}_{j,r}\right)$ of the conjunctive representation of $f^{-1}(\mathcal{A}_j)$.
2. Let
$$\mathcal{E}_{i-1}(x) \coloneqq \mathcal{E}_i(x) \cup \left\{\mathcal{A}_{j,j'} \mid j \in [n], j' \in [r], x_{j'} = x\right\}.$$
[For each $x \notin \{x_1, \ldots, x_r\}$, let $\mathcal{E}_{i-1}(x) \coloneqq \mathcal{E}_i(x)$.]
3. Let $i \coloneqq i - 1$.

Let $\mathcal{E}(x) \coloneqq \mathcal{E}_0(x)$ for each variable x. To decide the path feasibility of S, we simply show that each collection $\mathcal{E}(x)$ (for each $x \in \mathsf{Vars}(S)$) of conjunctive FAs has a nonempty language intersection.

Detailed complexity analysis. For each i, let N_i be the maximum size of the conjunctive FAs in $\bigcup_{x \in \mathsf{Vars}(S)} \mathcal{E}_i(x)$. Since each string function f satisfies $|f| \leq \mathsf{fnsize}(S)$, we have that $N_{i-1} \leq \ell(|f|, N_i) \leq \ell(\mathsf{fnsize}(S), N_i)$ (note that we have assumed that ℓ is monotone). Furthermore, because each conjunctive FA in $\mathcal{E}_{\mathsf{asgn}(S)}(x)$ for each $x \in \mathsf{Vars}(S)$ is of size bounded by $\mathsf{fasize}(S)$, we have that for each $x \in \mathsf{Vars}(S)$, each conjunctive FA in $\mathcal{E}(x)$ is of size bounded by $\ell^{\langle \mathsf{asgn}(S) \rangle}(\mathsf{fnsize}(S), \mathsf{fasize}(S))$. We emphasise that, according to the **PreRec** assumption, the construction of these conjunctive FAs can be done in nondeterministic $(\ell^{\langle \mathsf{asgn}(S) \rangle}(\mathsf{fnsize}(S), \mathsf{fasize}(S)))^{c_0}$ space.

According to Proposition 3, for each $x \in \mathsf{Vars}(S)$, the conjunctive product FA $\mathcal{A}_x = ((Q_x, \delta_x), S_x)$ of these conjunctive FAs in $\mathcal{E}(x)$ is of size

$$|\mathcal{E}(x)|(\ell^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}(S), \mathsf{fasize}(S))),$$

and $|S_x| \leq (\ell^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}(S), \mathsf{fasize}(S)))^2$. Therefore, the size of the (standard) FA corresponding to $\mathcal{A}_x$ is

$$(|\mathcal{E}(x)|\ell^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}(S), \mathsf{fasize}(S)))^{(\ell^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}(S), \mathsf{fasize}(S)))^2}.$$

Since the nonemptiness of an FA can be solved in nondeterministic logarithmic space, we conclude that the nonemptiness of $\mathcal{A}_x$ can be solved in nondeterministic

$$(\ell^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}(S), \mathsf{fasize}(S)))^2 \log(|\mathcal{E}(x)|\ell^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}(S), \mathsf{fasize}(S)))$$

space.

We now analyse $|\mathcal{E}(x)|$, the size of $\mathcal{E}(x)$. For each i, let M_i be the maximum number of elements in $\mathcal{E}_i(x)$ for $x \in \mathsf{Vars}(S)$. Then we have $M_{i-1} \leq (r+1)M_i \leq (\mathsf{art}(S)+1)M_i$. Since $\mathcal{E}_{\mathsf{asgn}(S)}(x)$ contains at most $\mathsf{asrt}(S)$ elements, we deduce that $\mathcal{E}(x)$ contains at most $(\mathsf{art}(S)+1)^{\mathsf{asgn}(S)}\mathsf{asrt}(S)$ elements.

Therefore, for each $x \in \mathsf{Vars}(S)$, the nonemptiness of $\mathcal{A}_x$ can be solved in nondeterministic

$$(\ell^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}(S), \mathsf{fasize}(S)))^2 \log[(\mathsf{art}(S)+1)^{\mathsf{asgn}(S)}\mathsf{asrt}(S)\ell^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}(S), \mathsf{fasize}(S))]$$

space.

Moreover, from the aforementioned analysis, we know that each FA $\mathcal{A}$ occurring in S gives rise to at most $(\mathsf{art}(S)+1)^{\mathsf{asgn}(S)}$ conjunctive FAs in $\bigcup_{x \in \mathsf{Vars}(S)} \mathcal{E}(x)$. Therefore,

$$\sum_{x \in \mathsf{Vars}(S)} |\mathcal{E}(x)| \leq (\mathsf{art}(S)+1)^{\mathsf{asgn}(S)}\mathsf{asrt}(S).$$

We then deduce that the sum of the sizes of all the conjunctive FAs in $\mathcal{E}(x)$ for all $x \in \mathsf{Vars}(S)$ is bounded by

$$\begin{aligned}&\sum_{x \in \mathsf{Vars}(S)} (|\mathcal{E}(x)|\ell^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}(S), \mathsf{fasize}(S))) \\ &\leq (\mathsf{art}(S)+1)^{\mathsf{asgn}(S)}\mathsf{asrt}(S)\ell^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}(S), \mathsf{fasize}(S)).\end{aligned}$$

In summary, the aforementioned nondeterministic algorithm takes

$$(\mathsf{art}(S)+1)^{\mathsf{asgn}(S)}\mathsf{asrt}(S)\left(\ell^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}(S), \mathsf{fasize}(S))\right)^c$$

space for some constant $c > 0$.

B Proof for expressing $\mathsf{replaceAll}_p$ by PT, claimed in Example 2

Let p be a fixed patten and $\mathsf{replaceAll}_p(sub, rep)$ be an instance of the $\mathsf{replaceAll}$ function, where sub, rep are the subject and replacement respectively. W.l.o.g. we assume that p is given by an FA $\mathcal{A}_0$. For simplicity, we assume that ε does not belong to $\mathcal{L}(\mathcal{A}_0)$. Our goal is to construct a PT $T_p = (Q, q_0, F, \delta)$ to simulate $\mathsf{replaceAll}_p(sub, rep)$.

Intuitively, T_p takes sub as the input string and rep as a parameter. If $sub \in \Sigma^* \backslash (\Sigma^* e_0 \Sigma^*)$, then T_p outputs sub. Otherwise, T_p parses sub into the substrings $v_1 u_1 v_2 u_2 \dots v_l u_l v_{l+1}$ such that

- for each $j \in [l]$, u_j is the leftmost and longest matching of p in the string $(v_1 u_1 \dots v_{j-1} u_{j-1})^{-1} v$,
- $v_{l+1} \in \Sigma^* \backslash (\Sigma^* e_0 \Sigma^*)$,

and outputs $v_1 \cdot rep \cdot v_2 \cdot rep \dots v_l \cdot rep \cdot v_{l+1}$.

T_p is essentially the extension of the parsing automaton $\mathcal{A}_p$ constructed in [15] with outputs.

Since the construction is very similar to that of $\mathcal{A}_p$ in [15], we will only give an intuitive description of the behaviour of T_p below and encourage the readers to check $\mathcal{A}_p$ in [15] for details. Intuitively, in order to search for the leftmost and longest matching of p (i.e. $\mathcal{A}_0$), T_p behaves as follows.

- T_p has two modes, "$\mathsf{searchLeft}$" and "$\mathsf{searchLong}$", which intuitively means searching for the first and last position of the leftmost and longest matching of e_0 respectively.
- When in the "$\mathsf{searchLeft}$" mode, T_p starts a new thread of $\mathcal{A}_0$ in each position and records *the set of states* of the threads into a vector. In addition, it nondeterministically makes a "leftmost" guessing, that is, guesses that the current position is the first position of the leftmost and longest matching. Before making such a guessing, T_p outputs each letter it reads. If it makes such a guessing, it enters the "$\mathsf{searchLong}$" mode, runs the thread started in the current position and searches for the last position of the leftmost and longest matching. Moreover, it stores in a set S the union of the sets of states of all the threads that were started before the current position and continues running these threads to make sure that, in these threads, the final states will not be reached (thus, the current position is indeed the first position of the leftmost and longest matching).
- When in the "$\mathsf{searchLong}$" mode, T_p runs a thread of $\mathcal{A}_0$ to search for the last position of the leftmost and longest matching. If the set of states of the thread contains a final state, then T_p nondeterministically guesses that the current position is the last position of the leftmost and longest matching. If it makes such a guessing, then it resets the set of states of the thread and starts a new round of searching for the leftmost and longest matching. In addition, it stores the original set of states of the thread into a set S and continues running the thread to make sure that in this thread, the final states

will not be reached (thus, the current position is indeed the last position of the leftmost and longest matching). Before making such a guessing in the "searchLong" mode, T_p always outputs ϵ. On the other hand, when the guessing is made, T_p outputs the parameter *rep*.
- Since the length of the vectors of the sets of states of the threads may become unbounded, in order to obtain a finite state automaton, the following trick is applied. Suppose that the vector is $S_1 S_2 \cdots S_n$. For each pair of indices $i, j : i < j$ and each $q \in S_i \cap S_j$, remove q from S_j. The application of this trick is justified by the following arguments: Since q occurs in both S_i and S_j and the thread i was started before the thread j, even if from q a final state can be reached in the future, the position where the thread j was started *cannot* be the first position of the leftmost and longest matching, since the state q is also a state of the thread i and the position where the thread i was started is before the position where the thread i was started.

Since the size of T_p is the same as the parsing automaton $\mathcal{A}_p$ in [15], and it was shown in [15] that the size of $\mathcal{A}_p$ is $2^{\mathcal{O}(|p|^c)}$ for some constant $c > 0$, we conclude that the size of T_p is $2^{\mathcal{O}(|p|^c)}$.

C Details in Section 5.1

Claim. For each tuple of strings $(w, w_1, \cdots, w_r)$, $(w, w_1, \cdots, w_r) \in \mathsf{Pre}_T(\mathcal{A})$ iff there are $\Omega_{x_1}, \cdots, \Omega_{x_r} \subseteq Q' \times Q'$ and $q \in F$ such that $w \in \mathcal{L}(\mathcal{B}_{T,\Omega_{x_1},\cdots,\Omega_{x_r},q})$ and $w_i \in \mathcal{L}(\mathcal{A}[\Omega_{x_i}])$ for each $i \in [r]$, where $\mathcal{A}[\Omega_{x_i}] = ((Q', \delta'), \Omega_{x_i})$.

Proof. Recall $T = (X, Q, q_0, F, \delta)$ with $X = \{x_1, \cdots, x_r\}$ and $\mathcal{A} = ((Q', \delta'), \Omega)$. Suppose $\Gamma = \Sigma \cup X$.

Let $w, w_1, \cdots, w_r$ be strings and $w = a_1 \cdots a_n$ with $a_i \in \Sigma$ for each $i \in [n]$.

"Only if" direction:

Assume that $(w, w_1, \cdots, w_r) \in \mathsf{Pre}_T(\mathcal{A})$.

From the definition of $\mathsf{Pre}_T(\mathcal{A})$, since T is functional, we know that there are a unique string w', and an accepting run of T on w, equipped with the parameters $w_1, \cdots, w_r$, say $(q_0, i_0, w'_0), \ldots, (q_m, i_m, w'_m) \in Q \times [n+1] \times \Gamma^*$, such that

- $i_0 = 0$, $i_m = n + 1$, $q_m \in F$,
- let $a_0 = \triangleright$ and $a_{n+1} = \triangleleft$, then $w'_0 = w'_m = \varepsilon$, and for every $j \in [0, m-1]$, $(q_j, a_{i_j}, dir_{i_j}, q_{j+1}, w'_j) \in \delta$ and $i_{j+1} = i_j + dir_{i_j}$ for some $dir_{i_j} \in \{-1, 0, 1\}$,
- w' is the output corresponding to the run, that is, $w' = w''_1 \cdots w''_{m-1}$, where for each $j \in [m-1]$, $w''_j = w'_j[w_1/x_1, \ldots, w_r/x_r]$,
- $w' \in \mathcal{L}(\mathcal{A})$.

For each $j \in [m-1]$, suppose $w'_j = w'_{j,1} x_{i_{j,1}} \ldots x_{i_{j,s_j}} w'_{j,s_j+1}$ such that $w'_{j,1}, \ldots, w'_{j,s_j+1} \in \Sigma^*$.

Since $w' = w''_1 \cdots w''_{m-1}$ and $w''_j = w'_j[w_1/x_1, \ldots, w_r/x_r]$, from $w' \in \mathcal{L}(\mathcal{A})$, we deduce that for each $\omega = (p, p') \in \Omega$, there is a state sequence

$$q'_{\omega,1,0}, \cdots, q'_{\omega,1,2s_1+1}, q'_{\omega,2,1}, \cdots, q'_{\omega,2,2s_2+1}, \cdots, q'_{\omega,m-1,1}, \cdots, q'_{\omega,m-1,2s_{m-1}+1} \in Q'$$

- $q'_{\omega,1,0} = p$,
- for each $j \in [m-1]$ and $j' \in [s_j]$,

$$q'_{\omega,j,2(j'-1)} \xrightarrow[\mathcal{A}]{w'_{j,j'}} q'_{\omega,j,2j'-1} \xrightarrow[\mathcal{A}]{w_{i_{j,j'}}} q'_{\omega,j,2j'},$$

  and $q'_{\omega,j,2s_j} \xrightarrow[\mathcal{A}]{w'_{j,s_j+1}} q'_{\omega,j,2s_j+1}$, where $q'_{\omega,j,0} = q'_{\omega,j-1,2s_{j-1}+1}$ if $j > 1$,
- $q'_{\omega,m-1,2s_{m-1}+1} = p'$.

For each $i' \in [r]$, let $\Omega_{x_{i'}}$ be the set of state pairs $(q'_{\omega,j,2j'-1}, q'_{\omega,j,2j'})$ such that $\omega \in \Omega$, $j \in [m-1]$, and $x_{i_{j,j'}} = x_{i'}$. Evidently, for each $i' \in [r]$, $w_{i'}$ is accepted by $\mathcal{A}[\Omega_{x_{i'}}]$. With $\Omega_{x_1}, \ldots, \Omega_{x_r}$, we can construct the conjunctive 2SA $\mathcal{B}_{T,\mathcal{A},\Omega_{x_1},\ldots,\Omega_{x_r},q_m} = ((Q'', \delta''), \Omega'')$ as in Section 5.1, where $\Omega'' = \{((q_0, p), (q_m, p')) \mid (p, p') \in \Omega\}$. From the construction of $\mathcal{B}_{T,\mathcal{A},\Omega_{x_1},\ldots,\Omega_{x_r},q_m}$, we know that $\mathcal{B}_{T,\mathcal{A},\Omega_{x_1},\ldots,\Omega_{x_r},q_m}$ contains the following transitions:

- $((q_0, q'_{\omega,j,0}), a_{i_0}, dir_{i_0}, (q_1, q'_{\omega,j,0}))$,
- $((q_j, q'_{\omega,j,0}), a_{i_j}, dir_{i_j}, (q_{j+1}, q'_{\omega,j,2s_j+1}))$ for each $j \in [m-1]$ and $\omega \in \Omega$.

Therefore, for each $\omega = (p, p') \in \Omega$,

$$((q_0, q'_{\omega,1,0}), i_0), \ldots, ((q_m, q'_{\omega,m-1,2s_{m-1}+1}), i_m)$$

is an accepting run of $((Q'', \delta''), \{((q_0, p), (q_m, p'))\})$ on w. We then conclude that w is accepted by $\mathcal{B}_{T,\mathcal{A},\Omega_{x_1},\ldots,\Omega_{x_r},q_m}$.

"If" direction:

Suppose that there are $\Omega_{x_1}, \cdots, \Omega_{x_r} \subseteq Q' \times Q'$ and $q \in F$ such that $w \in \mathcal{L}(\mathcal{B}_{T,\Omega_{x_1},\cdots,\Omega_{x_r},q})$ and $w_i \in \mathcal{L}(\mathcal{A}[\Omega_{x_i}])$ for each $i \in [r]$.

Since $\mathcal{L}(\mathcal{B}_{T,\Omega_{x_1},\cdots,\Omega_{x_r},q}) = \mathcal{L}(\mathcal{A}'')$, we know that there are an accepting run of T on w, say $(q_0, i_0, w'_0), \ldots, (q_m, i_m, w'_m) \in Q \times [n+1] \times \Gamma^*$, such that

- $i_0 = 0$, $i_m = n+1$, $q_m \in F$, $w'_0 = \epsilon$,
- let $a_0 = \triangleright$ and $a_{n+1} = \triangleleft$, then for every $j \in [m-1]$, $(q_j, a_{i_j}, dir_{i_j}, q_{j+1}, w'_j) \in \delta$ and $i_{j+1} = i_j + dir_{i_j}$ for some $dir_{i_j} \in \{-1, 0, 1\}$,
- let $\omega = (p, p') \in \Omega$, and for each $j \in [m-1]$, $w'_j = w'_{j,1} x_{i_{j,1}} \ldots w'_{j,s_j} x_{i_{j,s_j}} w'_{j,s_j+1}$ such that $w'_{j,j'} \in \Sigma^*$ for each $j' \in [s_j + 1]$, then there are states

$$q'_{\omega,1,0}, q'_{\omega,1,1}, \cdots, q'_{\omega,1,2s_1+1}, \cdots, q'_{\omega,m-1,1}, \cdots, q'_{\omega,m-1,2s_{m-1}+1} \in Q'$$

  satisfying that

- $q'_{\omega,1,0} = p$, $q'_{\omega,m-1,2s_{m-1}+1} = p'$, and
- for each $j \in [m-1]$,

$$q'_{\omega,j,0} \xrightarrow[\mathcal{A}]{w'_{j,1}} q'_{\omega,j,1} \xrightarrow{\Omega_{x_{i_{j,1}}}} q'_{\omega,j,2} \ \cdots \ q'_{\omega,j,2s_j-1} \xrightarrow{\Omega_{x_{i_{j,s_j}}}} q'_{\omega,j,2s_j} \xrightarrow[\mathcal{A}]{w'_{j,s_j+1}} q'_{\omega,j,2s_j+1},$$

where $q'_{\omega,j,0} = q'_{\omega,j-1,2s_{j-1}+1}$ if $j > 1$, and $q'_{\omega,j,2j'-1} \xrightarrow{\Omega_{x_{i_{j,j'}}}} q'_{\omega,j,2j'}$ means $(q'_{\omega,j,2j'-1}, q'_{\omega,j,2j'}) \in \Omega_{x_{i_{j,j'}}}$ for each $j' \in [s_j]$.

Let w' be the output of the run on w, equipped with the parameters $w_1, \dots, w_r$, more precisely, $w' = w''_1 \cdots w''_{m-1}$ such that $w''_j = w'_j[w_1/x_1, \dots, w_r/x_r]$ for each $j \in [m-1]$. From the fact that $w_i \in \mathcal{L}(\mathcal{A}[\Omega_{x_i}])$ for each $i \in [r]$, we deduce that for each $\omega = (p, p') \in \Omega$ and $j \in [m-1]$, $q'_{\omega,j,0} \xrightarrow[\mathcal{A}]{w''_j} q'_{\omega,j,2s_j+1}$. Therefore, for each $\omega = (p, p') \in \Omega$, $p \xrightarrow[\mathcal{A}]{w'} p'$. From this, we deduce that w' is accepted by $\mathcal{A} = ((Q', \delta'), \Omega)$. Consequently, $(w, w_1, \cdots, w_r) \in \mathsf{Pre}_T(\mathcal{A})$. □

D Proof of Theorem 5: EXPSPACE upper bound for PTs, replaceall, and reverse

Let S be a string constraint with PTs, replaceall, and reverse.

We first notice that, by Lemma 2, since PTs are corresponding to 0-RB2PTs, they satisfy the regularity condition **PreRec** with $\ell(|T|, |\mathcal{A}|) = |T||\mathcal{A}|$.

For replaceall $\mathsf{replaceAll}_p(sub, rep)$ where p is a fixed regular expression, we notice that, by Proposition 2, the PT T_p modelling this function is of exponential size in that of p, i.e., $|T_p| \in 2^{\mathcal{O}(|p|^c)}$ for some constant $c > 0$. Therefore, according to Lemma 2, $\mathsf{replaceAll}_p(sub, rep)$ satisfies the **PreRec** with $\ell_{replace}(|\mathcal{A}|) = 2^{\mathcal{O}(|p|^c)}|\mathcal{A}|$.

It is also easy to observe reverse (and concatenation) satisfy **PreRec** with $\ell_{conc}(|\mathcal{A}|) = \mathcal{O}(|\mathcal{A}|)$, and $\ell_{reverse}(|\mathcal{A}|) = \mathcal{O}(|\mathcal{A}|)$.

Let $T_1, \dots, T_m$ be an enumeration of PTs in S and $\mathsf{replaceAll}_{p_1}, \dots, \mathsf{replaceAll}_{p_n}$ be an enumeration of replaceall functions in S. Then by Theorem 2, the generic decision procedure runs in nondeterministic

$$(\mathsf{art}(S)+1)^{\mathsf{asgn}(S)}\mathsf{asrt}(S) \prod_{i\in[m], j\in[n]} (|T_i|2^{\mathcal{O}(|p_j|^c)}\mathsf{fasize}(S))^{\mathcal{O}(1)}$$

space.

By Savitch's theorem, we conclude that the path feasibility problem of S can be solved in (deterministic) exponential space.

E Details in Section 6: Symbolic extensions

In this section, we supply the details of Section 6, especially the complexity analysis of the decision procedure.

E.1 Some definitions, notations, and basic facts

We first make it explicit the definition of symbolic automata.

Definition 8 (Symbolic automata). *A (nondeterministic two-way) symbolic automata (2SA) is a tuple $\mathcal{A} = (\triangleright, \triangleleft, Q, q_0, F, \delta)$, where*

- *$\triangleright$, $\triangleleft$, Q, q_0, F are defined precisely as in 2FAs,*
- *δ is a finite set of transitions of one of the following forms,*
  - *symbolic transitions $(q, \psi(x), dir, q') \in Q \times \Psi \times \{-1, 0, 1\} \times Q$, where q, q' are the source and target state, dir is the direction, and $\psi(x)$ is the guard of the transition,*
  - *non-symbolic transitions $(q, \triangleright, dir, q')$ with $dir \in \{1, 0\}$, or $(q, \triangleleft, dir', q')$ with $dir' \in \{-1, 0\}$.*

A 2SA is an SA if there are no transitions with the direction "-1".

Size measures. We first assume that given a formula $\psi \in \Psi$ of size m in Υ, $\mathsf{isSat}(\psi)$ can be decided in $\mathcal{O}(\beta(m))$ space for some monotone function β such that $\beta(m) \geq \log m$. We also assume that the formulae and terms in Υ are encoded in a proper way, e.g. encoded in binary, and let $|\psi|$ and $|t|$ to denote the size (i.e. number of symbols) of ψ and t with respect to this encoding.

Let T be a 2SPT. We define $|T|_s$ and $|T|_d$, the *structural and data size* of T, by taking the guards and outputs of transitions (from the background theory Υ) into consideration. Specifically, $|T|_s$ is defined as the product of the number of transitions and the maximum length of sequences of $\mathcal{S}$-terms in transitions of T; $|T|_d$ is defined as the maximum size of symbolic transitions of T, where the size of a symbolic transition $(q, \psi(x), dir, q', \vec{t}(x))$ with $\vec{t} = t_1 ... t_r$ is defined as $|\psi| + |t_1| + ... + |t_r|$.

The size measures of 2SAs are defined similarly, specifically, $|\mathcal{A}|_s$ is defined as the number of transitions of $\mathcal{A}$, and $|\mathcal{A}|_d$ is defined as the maximum size of symbolic transitions of $\mathcal{A}$, where the size of a symbolic transition $(q, \psi(x), dir, q')$ is defined as $|\psi|$.

Then we state and prove some basic facts about 2SAs.

Proposition 4. *Assume a 2SA $\mathcal{A}$. Then*

- *$\mathcal{A}$ can be transformed into an equivalent SA $\mathcal{A}'$ in time $|\mathcal{A}'|_s \cdot |\mathcal{A}'|_d$, where $|\mathcal{A}'|_s = 2^{\mathcal{O}(|\mathcal{A}|_s \log |\mathcal{A}|_s)}$ and $|\mathcal{A}'|_d = \mathcal{O}(|\mathcal{A}|_s |\mathcal{A}|_d)$;*
- *the nonemptiness of $\mathcal{A}$ can be decided in nondeterministic $\mathcal{O}((\log |\mathcal{A}|_s) + \beta(|\mathcal{A}|_d))$ space.*

Proof. The proof the second result is obtained by guessing a path in $\mathcal{A}$ and was shown in [19, 54].

Next, we prove the first result.

Let $\mathcal{A} = (\Upsilon, \triangleright, \triangleleft, Q, q_0, F, \delta)$ be a 2SA. We show how to construct an equivalent SA $\mathcal{A}' = (\Upsilon, Q', q_0', F', \delta')$. W.l.o.g., we assume that for each transition $(q, \psi, dir, q') \in \delta$, we have $dir \in \{-1, 1\}$. Every 2SA can be turned into one 2SA

satisfying the assumption as follows: Replace each transition $(q, \psi, 0, q') \in \delta$ with three new transitions $(q, \psi, 1, q'')$, $(q'', \mathsf{true}, -1, q')$, and $(q'', \lhd, -1, q')$, where q'' is a fresh state.

The proof is an adaptation of the construction of FA from 2FA based on the notion of crossing sequences (cf., e.g., [30]). Since 2SAs replace the letters in 2FAs with unary predicates, the main technical challenge here is to deal with these predicates (i.e., guards).

We first use an example to illustrate the idea of crossing sequences: Suppose $w = \rhd\, d_1 d_2 d_3 d_4 d_5 \lhd$, and an accepting run of $\mathcal{A}$ on w uses the following sequence of transitions

$$(q_0, \rhd, 1, q_1), (q_1, \psi_1, 1, q_1), (q_1, \psi_2, -1, q_2), (q_2, \psi_3, 1, q_0), (q_0, \psi_4, 1, q_1),$$
$$(q_1, \psi_1, 1, q_1), (q_1, \psi_1, 1, q_1), (q_1, \psi_2, -1, q_2), (q_2, \psi_3, 1, q_0), (q_0, \psi_4, 1, q_1),$$

with $q_1 \in F$. (The run is illustrated in Figure 2.) A *crossing sequence* of the run is a sequence of states on the boundary of two tape cells, e.g. the sequence q_1, q_2, q_0 on the boundary of d_1 and d_2. Note that for technical convenience, the states of the run are put in a way that, whenever a left-transition $(q, \psi, -1, q')$ is taken, q' is put just below q, thus in the *same* crossing sequence as q rather than the cross sequence of th position on the left. Therefore, *the target state of each left-transition is put in the position right-adjacent to the actual position of the reading head*. For instance, in Figure 2, when the transition $(q_1, \psi_2, -1, q_2)$ is taken, q_2 is put just below q_1. Different from the crossing sequences for 2FAs, in Figure 2, naturally we also include the guards of the transitions.

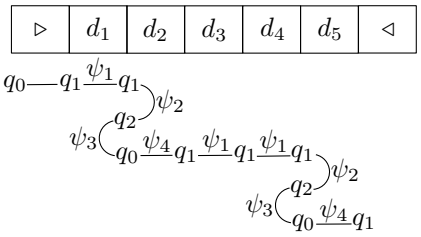

Fig. 2. Accepting runs of $\mathcal{A}$: An example

In order to deal with the guards in the transitions, we adapt the crossing sequences of the 2SA $\mathcal{A}$ as follows: a crossing sequence of $\mathcal{A}$ is a sequence of *transitions* of odd lengths such that the source states of the transitions in the even and odd positions respectively are mutually distinct. Let S_δ denote the set of such sequences of transitions.

In order to construct the set of transitions of $\mathcal{A}'$, we shall define two *partial* functions $\mathsf{LeftMatch}(\rho_1, \rho_2)$ and $\mathsf{RightMatch}(\rho_1, \rho_2)$ for $\rho_1, \rho_2 \in S_\delta$. The intention of $\mathsf{LeftMatch}(\rho_1, \rho_2)$ and $\mathsf{LeftMatch}(\rho_1, \rho_2)$ is explained as follows.

– The intention of $\mathsf{LeftMatch}(\rho_1, \rho_2)$: ρ_1 and ρ_2 are the crossing sequences to the left resp. right of some position i holding a data value d. Initially, we reach the source state of the first transition of ρ_2 by a *left-transition* (thus after this transition, the reading head is in the position i[4]).
– The intention of $\mathsf{RightMatch}(\rho_1, \rho_2)$: ρ_1 and ρ_2 are the crossing sequences to the left resp. right of some position i holding a data value d. Initially, we reach the source state of the first transition of ρ_1 by a *right-transition* (thus after this transition, the reading head is in the position i).

Formally, the domain of $\mathsf{LeftMatch}$ (resp. $\mathsf{RightMatch}$) is defined inductively as follows, where in each case below, the return value is also specified. Let $\rho_1, \rho_2 \in S_\delta$. Then $\mathsf{LeftMatch}(\rho_1, \rho_2)$ is defined iff one of the following constraints holds.

– Case $\rho_1 = \varepsilon$, $\rho_2 = \varepsilon$: $\mathsf{LeftMatch}(\rho_1, \rho_2) = \mathsf{true}$.
– Case $\rho_1 = \tau_{1,1}, \ldots, \tau_{1,k}$ with $k \geq 1$, $\rho_2 = \tau_{2,1}, \ldots, \tau_{2,l}$ with $l \geq 1$, $\tau_{2,1} = (q_1, \psi_{2,1}, -1, p_1)$, p_1 is the source state of $\tau_{1,1}$, and $\mathsf{RightMatch}(\rho_1', \rho_2')$ is defined:
$$\mathsf{LeftMatch}(\rho_1, \rho_2) = \psi_{2,1} \wedge \mathsf{RightMatch}(\rho_1', \rho_2'),$$
where $\rho_1' = \tau_{1,2}, \ldots, \tau_{1,k}$ and $\rho_2' = \tau_{2,2}, \ldots, \tau_{2,l}$.
– Case $\rho_1 = \tau_{1,1}, \ldots, \tau_{1,k}$ with $k \geq 1$, $\rho_2 = \tau_{2,1}, \ldots, \tau_{2,l}$ with $l \geq 1$, $\tau_{2,1} = (q_1, \psi_{2,1}, 1, q_2)$, q_2 is the source state of $\tau_{2,2}$, and $\mathsf{LeftMatch}(\rho_1, \rho_2')$ is defined:
$$\mathsf{LeftMatch}(\rho_1, \rho_2) = \psi_{2,1} \wedge \mathsf{LeftMatch}(\rho_1, \rho_2'),$$
where $\rho_2' = \tau_{2,3}, \ldots, \tau_{2,l}$.

Symmetrically, $\mathsf{RightMatch}(\rho_1, \rho_2)$ is defined iff one of the following constraints holds.

– Case $\rho_1 = \varepsilon$, $\rho_2 = \varepsilon$: $\mathsf{RightMatch}(\rho_1, \rho_2) = \mathsf{true}$.
– Case $\rho_1 = \tau_{1,1}, \ldots, \tau_{1,k}$ with $k \geq 1$, $\rho_2 = \tau_{2,1}, \ldots, \tau_{2,l}$ with $l \geq 1$, $\tau_{1,1} = (p_1, \psi_{1,1}, 1, q_1)$, q_1 is the source state of $\tau_{2,1}$, and $\mathsf{LeftMatch}(\rho_1', \rho_2')$ is defined:
$$\mathsf{RightMatch}(\rho_1, \rho_2) = \psi_{1,1} \wedge \mathsf{LeftMatch}(\rho_1', \rho_2'),$$
where $\rho_1' = \tau_{1,2}, \ldots, \tau_{1,k}$, and $\rho_2' = \tau_{2,2}, \ldots, \tau_{2,l}$.
– Case $\rho_1 = \tau_{1,1}, \ldots, \tau_{1,k}$ with $k \geq 1$, $\rho_2 = \tau_{2,1}, \ldots, \tau_{2,l}$ with $l \geq 1$, $\tau_{1,1} = (p_1, \psi_{1,1}, -1, p_2)$, p_2 is the source state of $\tau_{1,2}$, and $\mathsf{RightMatch}(\rho_1', \rho_2)$ is defined:
$$\mathsf{RightMatch}(\rho_1, \rho_2) = \psi_{1,1} \wedge \mathsf{RightMatch}(\rho_1', \rho_2),$$
where $\rho_1' = \tau_{1,3}, \ldots, \tau_{1,k}$.

We are ready to construct the SA $\mathcal{A}' = (\Upsilon, Q', q_0', F', \delta')$, where

– $Q' = S_\delta \cup \{q_0'\}$, where q_0' is a fresh state,

[4] Recall that the target state of each left-transition is put in the position right-adjacent to the actual position of the reading head.

– F' comprises the set of elements $\rho \in S_\delta$ as follows: suppose $\rho = \tau_1, \ldots, \tau_{2n+1}$, where for each $i \in [n+1]$, $\tau_{2i-1} = (p_{2i-1}, \lhd, -1, q_{2i-1})$, and for each $i \in [n]$, $\tau_{2i} = (p_{2i}, \psi_{2i}, dir_{2i}, q_{2i})$, then ρ must satisfy that $q_{2n+1} \in F$ and $q_{2i-1} = p_{2i}$ for each $i \in [n]$,
– δ' comprises the following transitions:
  - $(\rho, \mathsf{RightMatch}(\rho, \rho'), \rho') \in \delta'$ for all $\rho, \rho' \in S_\delta$ such that $\mathsf{RightMatch}(\rho, \rho')$ is defined;
  - $(q_0', \mathsf{RightMatch}(\rho, \rho'), \rho') \in \delta'$ for all $\rho \in I'$ and $\rho' \in S_\delta$ such that $\mathsf{RightMatch}(\rho, \rho')$ is defined, where I' comprises $\rho'' \in S_\delta$ satisfying the following constraints: $\rho'' = \tau_1, \ldots, \tau_{2n+1}$, for each $i \in [n+1]$, $\tau_{2i-1} = (p_{2i-1}, \psi_{2i-1}, dir_{2i-1}, q_{2i-1})$, for each $i \in [n]$, $\tau_{2i} = (p_{2i}, \rhd, 1, q_{2i})$. In addition, $(q_0, \rhd, 1, p_1) \in \delta$, and for each $i \in [n]$, $q_{2i} = p_{2i+1}$.

Since S_δ contains at most $(|\mathcal{A}|_s)^{2|Q|-1} \leq (|\mathcal{A}|_s)^{2|\mathcal{A}|_s}$ elements, we conclude that

$$|\mathcal{A}'|_s \leq ((|\mathcal{A}|_s)^{2|\mathcal{A}|_s})^2 = (|\mathcal{A}|_s)^{4|\mathcal{A}|_s} \approx 2^{\mathcal{O}(|\mathcal{A}|_s \log |\mathcal{A}|_s)}$$

and

$$|\mathcal{A}'|_d \leq 2(2|Q|-1)|\mathcal{A}|_d \leq 4|\mathcal{A}|_s|\mathcal{A}|_d \approx \mathcal{O}(|\mathcal{A}|_s|\mathcal{A}|_d).$$

□

E.2 The details of the generic decision procedure

Similar to recognisable relations, we also represent an r-ary symbolic recognisable relation as a collection of tuples $(\mathcal{A}_1, \ldots, \mathcal{A}_r)$, where each atom $\mathcal{A}_i$ is a conjunctive representation of SA, namely, of the form $((Q, \delta), \Omega)$ with $\Omega \subseteq Q \times Q$. We will use a conjunctive SA to mean a conjunctive representation of the SA. For a conjunctive SA $\mathcal{C} = ((Q, \delta), \Omega)$, two size measures $|\mathcal{C}|_s$ and $|\mathcal{C}|_d$ are defined similarly to SAs. Moreover, the structural (resp. data) size of a representation of a symbolic recognisable relation is defined as the maximum structural (resp. data) size of the atoms.

The regularity condition **PreRec** in Section 4 is replaced by $\mathbb{S}$**PreRec** presented below.

> **The symbolic regularity condition $\mathbb{S}$PreRec**. For each data string function f in S and each conjunctive SA $\mathcal{A}$, $f^{-1}(\mathcal{A})$ is a symbolic recognisable relation. Furthermore, a representation of $f^{-1}(\mathcal{A})$, whose structural size (resp. data size) is bounded by $\ell_s(|f|_s, |\mathcal{A}|_s)$ (resp. $\ell_d(|f|_s, |f|_d, |\mathcal{A}|_s, |\mathcal{A}|_d)$) for some monotone functions ℓ_s and ℓ_d, can be computed effectively. We also assume that each disjunct of the representation of $f^{-1}(\mathcal{A})$ can be enumerated in $\ell_s(|f|_s, |\mathcal{A}|_s) \cdot \ell_d(|f|_s, |f|_d, |\mathcal{A}|_s, |\mathcal{A}|_d)$ space.

Here $|f|_s$ and $|f|_d$ are the structural and data sizes of a representation of f respectively; the concrete definitions depend on the form of f, which will be given when the generic decision procedure is instantiated later. Furthermore, we define the size of f (resp. $\mathcal{A}$), denoted by $|f|$ (resp. $|\mathcal{A}|$), as $|f|_s \cdot |f|_d$ (resp. $|\mathcal{A}|_s \cdot |\mathcal{A}|_d$).

Let S be a symbolic execution of programs with data strings. We use $|S|$ to denote the sum of $|f|$ and $|\mathcal{A}|$ for data string functions f and SAs $\mathcal{A}$ occurring in S. For a refined complexity analysis, we also define $\mathsf{asgn}(S)$, $\mathsf{art}(S)$, and $\mathsf{asrt}(S)$ similarly to the string setting. Moreover, we use $\mathsf{fnsize}_s(S)$ (resp. $\mathsf{fnsize}_d(S)$) to denote the maximum of $|f|_s$ (resp. $|f|_d$) for data string functions f occurring in the assignments of S, and $\mathsf{fasize}_s(S)$ (resp. $\mathsf{fasize}_d(S)$) to denote the maximum of $|\mathcal{A}|_s$ (resp. $|\mathcal{A}|_d$) for SAs $\mathcal{A}$ occurring in S.

For a binary function $g(k,l)$, $h(i,j,k,l)$, and $n \in \mathbb{N}\backslash\{0\}$, we define $g^{\langle n\rangle}(k,l)$ and $h_g^{\langle n\rangle}(i,j,k,l)$ as follows: $g^{\langle 1\rangle}(k,l) = g(k,l)$, $h_g^{\langle 1\rangle}(i,j,k,l) = h(i,j,k,l)$, and

$$g^{\langle n+1\rangle}(k,l) = g(k, g^{\langle n\rangle}(k,l)),\ h_g^{\langle n+1\rangle}(i,j,k,l) = h(i,j,g^{\langle n\rangle}(k,l), h^{\langle n\rangle}(i,j,k,l)).$$

Theorem 7. *Given a symbolic execution S of programs with data strings satisfying $\mathbb{S}$**PreRec**, the path feasibility problem of S can be decided in* nondeterministic *$M + \beta(M)$ space, where*

$$M = (\mathsf{art}(S)+1)^{\mathsf{asgn}(S)}\mathsf{asrt}(S) \cdot (\ell_s^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fasize}_s(S)))^c \cdot (\ell_d)_{\ell_s}^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fnsize}_d(S), \mathsf{fasize}_s(S), \mathsf{fasize}_d(S))$$

for some constant $c > 0$.

Proof. For each i, let $N_{i,s}$ (resp. $N_{i,d}$) be the maximum structural size (resp. data size) of the conjunctive SAs in $\bigcup_{x\in\mathsf{Vars}(S)} \mathcal{E}_i(x)$. From $\mathbb{S}$**PreRec** and the fact that each string function f satisfies $|f|_s \le \mathsf{fnsize}_s(S)$ and $|f|_d \le \mathsf{fnsize}_d(S)$, we have

$$N_{i-1,s} \le \ell_s(|f|_s, N_{i,s}) \le \ell_s(\mathsf{fnsize}_s(S), N_{i,s})$$

and

$$N_{i-1,d} \le \ell_d(|f|_s, |f|_d, N_{i,c}, N_{i,d}) \le \ell_d(\mathsf{fnsize}_s(S), \mathsf{fnsize}_d(S), N_{i,s}, N_{i,d}).$$

Because each conjunctive SA in $\mathcal{E}_{\mathsf{asgn}(S)}(x)$ satisfies that its control and data size are bounded by $\mathsf{fasize}_c(S)$ and $\mathsf{fasize}_d(S)$ respectively, we have that each conjunctive SA in $\mathcal{E}(x)$ satisfies that its control and data size are bounded by $\ell_s^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fasize}_s(S))$ and

$$(\ell_d)_{\ell_s}^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fnsize}_d(S), \mathsf{fasize}_s(S), \mathsf{fasize}_d(S)).$$

Therefore, the size of each conjunctive SA in $\mathcal{E}(x)$ is bounded by

$$\ell_s^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fasize}_s(S)) \cdot (\ell_d)_{\ell_s}^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fnsize}_d(S), \mathsf{fasize}_s(S), \mathsf{fasize}_d(S)).$$

Furthermore, according to $\mathbb{S}$**PreRec**, the construction of these conjunctive SAs can be done in nondeterministic space bounded by

$$\ell_s^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fasize}_s(S)) \cdot (\ell_d)_{\ell_s}^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fnsize}_d(S), \mathsf{fasize}_s(S), \mathsf{fasize}_d(S))$$

as well.

For each i, let M_i be the maximum number of elements in $\mathcal{E}_i(x)$ for $x \in \mathsf{Vars}(S)$. Then we have $M_{i-1} \leq (\mathsf{art}(S)+1)M_i$. Because for each $x \in \mathsf{Vars}(S)$, $\mathcal{E}_{\mathsf{asgn}(S)}(x)$ contains at most $\mathsf{asrt}(S)$ elements, we have that for each $x \in \mathsf{Vars}(S)$, $\mathcal{E}(x)$ contains at most $(\mathsf{art}(S)+1)^{\mathsf{asgn}(S)}\mathsf{asrt}(S)$ elements.

Therefore, for each $x \in \mathsf{Vars}(S)$, the conjunctive product SA $\mathcal{A}_x = ((Q_x, \delta_x), \Omega_x)$ of these conjunctive SAs in $\mathcal{E}(x)$ has a structural size bounded by

$$(\ell_s^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fasize}_s(S)))^{(\mathsf{art}(S)+1)^{\mathsf{asgn}(S)}\mathsf{asrt}(S)},$$

and data size bounded by

$$(\mathsf{art}(S)+1)^{\mathsf{asgn}(S)}\mathsf{asrt}(S)(\ell_d)_{\ell_s}^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fnsize}_d(S), \mathsf{fasize}_s(S), \mathsf{fasize}_d(S))$$

and $|\Omega_x| \leq (\ell_s^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fasize}_s(S)))^2$. It follows that the structural and data size of the (standard) SA corresponding to $\mathcal{A}_x$ are bounded by

$$(\ell_s^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fasize}_s(S)))^{(\mathsf{art}(S)+1)^{\mathsf{asgn}(S)}\mathsf{asrt}(S)\left(\ell_s^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fasize}_s(S))\right)^2}$$

and

$$\left(\ell_s^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fasize}_s(S))\right)^2 (\mathsf{art}(S)+1)^{\mathsf{asgn}(S)}\mathsf{asrt}(S) \cdot (\ell_d)_{\ell_s}^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fnsize}_d(S), \mathsf{fasize}_s(S), \mathsf{fasize}_d(S))$$

respectively.

Since the nonemptiness of an SA $\mathcal{A}$ can be solved in nondeterministic $(\log |\mathcal{A}|_s) + \beta(|\mathcal{A}|_d)$ space, we conclude that the nonemptiness of $\mathcal{A}_x$ can be solved in nondeterministic space bounded by

$$\begin{aligned}&(\mathsf{art}(S)+1)^{\mathsf{asgn}(S)}\mathsf{asrt}(S) \cdot \left(\ell_s^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fasize}_s(S))\right)^2 \cdot \\ &\qquad \log(\ell_s^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fasize}_s(S))) \\ &+\beta\left(\begin{array}{c}(\ell_s^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fasize}_s(S)))^2(\mathsf{art}(S)+1)^{\mathsf{asgn}(S)}\mathsf{asrt}(S) \cdot \\ (\ell_d)_{\ell_s}^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fnsize}_d(S), \mathsf{fasize}_s(S), \mathsf{fasize}_d(S))\end{array}\right)\end{aligned}$$

In addition, from the fact $M_{i-1} \leq (\mathsf{art}(S)+1)M_i$, we deduce that each conjunctive SA in $\bigcup_{x \in \mathsf{Vars}(S)} \mathcal{E}_{\mathsf{asgn}(S)}(x)$ gives rise to at most $(\mathsf{art}(S)+1)^{\mathsf{asgn}(S)}$ conjunctive SAs in $\bigcup_{x \in \mathsf{Vars}(S)} \mathcal{E}(x)$. Therefore, we have $\sum_{x \in \mathsf{Vars}(S)} |\mathcal{E}(x)| \leq (\mathsf{art}(S)+1)^{\mathsf{asgn}(S)}\mathsf{asrt}(S)$. It follows that $\bigcup_{x \in \mathsf{Vars}(S)} \mathcal{E}(x)$ occupies at most

$$\begin{aligned}&(\mathsf{art}(S)+1)^{\mathsf{asgn}(S)}\mathsf{asrt}(S) \cdot \ell_s^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fasize}_s(S)) \cdot \\ &(\ell_d)_{\ell_s}^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fnsize}_d(S), \mathsf{fasize}_s(S), \mathsf{fasize}_d(S))\end{aligned}$$

space.

In summary, the aforementioned nondeterministic algorithm takes $M + \beta(M)$ space, where

$$M = (\mathsf{art}(S)+1)^{\mathsf{asgn}(S)}\mathsf{asrt}(S) \cdot (\ell_s^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fasize}_s(S)))^c \cdot (\ell_d)_{\ell_s}^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fnsize}_d(S), \mathsf{fasize}_s(S), \mathsf{fasize}_d(S))$$

for some constant $c > 0$. □

E.3 Instantiation of the generic decision procedure to 2SPTs and RB2SPTs.

Lemma 4. *The regularity condition* $\mathbb{S}$***PreRec*** *holds for 2SPTs with*

$$\ell_s(|T|_s, |\mathcal{A}|_s) = 2^{\mathcal{O}(|T|_s |\mathcal{A}|_s^{|T|_s} \log(|T|_s |\mathcal{A}|_s^{|T|_s}))}$$

and

$$\ell_d(|T|_s, |T|_d, |\mathcal{A}|_s, |\mathcal{A}|_d) = \mathcal{O}(|T|_s |T|_d |\mathcal{A}|_s^{|T|_s} |\mathcal{A}|_d).$$

The regularity condition $\mathbb{S}$***PreRec*** *holds for k-RB2SPTs, with*

$$\ell_s(|T|_s, |\mathcal{A}|_s) = \mathcal{O}(|T|_s^{k+1} |\mathcal{A}|_s^{(k+1)|T|_s})$$

and

$$\ell_d(|T|_s, |T|_d, |\mathcal{A}|_s, |\mathcal{A}|_d) = \mathcal{O}(|T|_d |\mathcal{A}|_d).$$

Proof. We first consider 2SPTs. Assume a 2SPT $T = (Y, Q, q_0, F, \delta)$ with $Y = \{y_1, \cdots, y_r\}$ and a conjunctive SA $\mathcal{A} = ((Q', \delta'), \Omega')$ with $\Omega' \subseteq Q' \times Q'$.

Similarly to 2FAs, for $q \in F$ and $\Omega_{y_1}, \cdots, \Omega_{y_r} \subseteq Q' \times Q'$, we introduce a conjunctive 2SA $\mathcal{B}_{T,\mathcal{A},\Omega_{y_1},\cdots,\Omega_{y_r},q} = ((Q'', \delta''), \Omega'')$, where $Q'' = Q \times Q'$, $\Omega'' = \{((q_0, p), (q, p')) \mid (p, p') \in \Omega'\}$, and the transition relation δ'' comprises the tuples $((q_1, q_1'), \psi(x), dir, (q_2, q_2'))$ such that there are $\psi_0(x) \in \Psi$ and $\vec{t}$ satisfying that $(q_1, \psi_0(x), dir, q_2, \vec{t}) \in \delta$, and one of the following constraints holds,

- $\vec{t} = t_1(x) \dots t_m(x)$ is a sequence of $\mathcal{S}$-terms, there are transitions

  $$(p_0, \psi_1(x), p_1), \dots, (p_{m-1}, \psi_m(x), p_m) \in \delta'$$

  such that $p_0 = q_1'$, $p_r = q_2'$, and $\psi = \psi_0 \wedge \psi_1[t_1(x)/x] \wedge \dots \wedge \psi_m[t_m(x)/x]$,
- $\vec{t} = \vec{t}_1 y_{i_1} \dots \vec{t}_m y_{i_m} \vec{t}_{m+1}$ such that $m \geq 1$, $\vec{t}_j = t_{j,1} \dots t_{j,s_j}$ is a sequence of $\mathcal{S}$-terms for each $j \in [m+1]$, and $i_j \in [k]$ for each $j \in [m]$, let $N = (\sum_{j \in [m]} (s_j + 1)) + s_{m+1}$, then there are states $p_0, \dots, p_N \in Q'$ satisfying that $p_0 = q_1'$, $p_N = q_2'$,

  $$\begin{aligned} & p_0 \xrightarrow[\mathcal{A}]{\psi_{1,1}} p_1 \ \cdots \ p_{|\vec{t}_1|-1} \xrightarrow[\mathcal{A}]{\psi_{1,s_1}} p_{|\vec{t}_1|} \xrightarrow{\Omega_{y_{i_1}}} p_{|\vec{t}_1|+1} \ \cdots \\ & p_{N-s_{m+1}-s_m-1} \xrightarrow[\mathcal{A}]{\psi_{m,1}} p_{N-s_{m+1}-s_m} \ \cdots \ p_{N-|\vec{t}_{m+1}|-2} \xrightarrow[\mathcal{A}]{\psi_{m,s_m}} p_{N-s_{m+1}-1} \\ & \xrightarrow{\Omega_{y_{i_m}}} p_{N-s_{m+1}} \xrightarrow[\mathcal{A}]{\psi_{m+1,1}} p_{N-s_{m+1}+1} \ \cdots \ p_{N-1} \xrightarrow[\mathcal{A}]{\psi_{m+1,s_{m+1}}} p_N, \end{aligned}$$

  and $\psi = \psi_0 \wedge \bigwedge_{j \in [m+1], j' \in [s_j]} \psi_{j,j'}[\vec{t}_{j,j'}(x)/x]$.

Evidently, $|\Omega''| = |\Omega'|$. Let L be the maximum length of sequences of $\mathcal{S}$-terms in transitions of T. Then

– the structural size of $\mathcal{B}_{T,\mathcal{A},\Omega_{y_1},\cdots,\Omega_{y_r},q}$ is at most $|T|_s|\mathcal{A}|_s^L \leq |T|_s|\mathcal{A}|_s^{|T|_s}$,
– on the other hand, since the size of $\psi_1[t_1(x)/x] \wedge ... \wedge \psi_r[t_m(x)/x]$ is at most $|T|_d|\mathcal{A}|_d$, we deduce that the data size of $\mathcal{B}_{T,\mathcal{A},\Omega_{y_1},\cdots,\Omega_{y_r},q}$ is at most $|T|_d + |T|_d|\mathcal{A}|_d \approx \mathcal{O}(|T|_d|\mathcal{A}|_d)$.

Therefore, $\mathsf{Pre}_T(\mathcal{A})$ is equal to

$$\bigcup_{\Omega_{y_1},\cdots,\Omega_{y_r} \subseteq Q' \times Q', q \in F} \mathcal{L}(\mathcal{B}_{T,\mathcal{A},\Omega_{y_1},\cdots,\Omega_{y_r},q}) \times \mathcal{L}(\mathcal{A}[\Omega_{y_1}]) \times \cdots \times \mathcal{L}(\mathcal{A}[\Omega_{y_r}])$$

We conclude that $\mathsf{Pre}_T(\mathcal{A})$ is a recognisable relation.

In order to compute a representation of $\mathsf{Pre}_T(\mathcal{A})$, we transform the conjunctive 2SA $\mathcal{B}_{T,\mathcal{A},\Omega_{y_1},\cdots,\Omega_{y_r},q}$ to a union of conjunctive SAs as follows: Transform $\mathcal{B}_{T,\mathcal{A},\Omega_{y_1},\cdots,\Omega_{y_r},q}$ into a one-way transition graph (Q''', δ'''). From Proposition 4, Q''' are vectors of transitions of $\mathcal{B}_{T,\mathcal{A},\Omega_{y_1},\cdots,\Omega_{y_r},q}$ of lengths at most $2|Q''| - 1$. Then $\mathcal{L}(\mathcal{B}_{T,\mathcal{A},\Omega_{y_1},\cdots,\Omega_{y_r},q})$ is the union of $\mathcal{L}(((Q''', \delta'''), \Omega''))$ for $\Omega'' \subseteq Q''' \times Q'''$, which comprises nondeterministically selected pairs $(\vec{\rho}_1, \vec{\rho}_2) \in Q''' \times Q'''$, one for each $(p, p') \in \Omega'$, such that $\vec{\rho}_1[1] = ((q_0, p), -, -)$ and $\vec{\rho}_2[|\vec{\rho}_2|] = (-, -, (q, p'))$.

From Proposition 4, the structural (resp. data) size of each conjunctive SA $((Q''', \delta'''), \Omega'')$ is $2^{\mathcal{O}(|T|_s|\mathcal{A}|_s^{|T|_s} \log(|T|_s|\mathcal{A}|_s^{|T|_s}))}$ (resp. $\mathcal{O}(|T|_s|\mathcal{A}|_s^{|T|_s}|T|_d|\mathcal{A}|_d)$). We conclude that $\mathbb{S}$**PreRec** holds for data string functions definable in 2SPTs with

$$\ell_s(|T|_s, |\mathcal{A}|_s) = 2^{\mathcal{O}(|T|_s|\mathcal{A}|_s^{|T|_s} \log(|T|_s|\mathcal{A}|_s^{|T|_s}))}$$

and

$$\ell_d(|T|_s, |T|_d, |\mathcal{A}|_s, |\mathcal{A}|_d) = \mathcal{O}(|T|_s|\mathcal{A}|_s^{|T|_s}|T|_d|\mathcal{A}|_d).$$

On the other hand, since there are at most $k+1$ transitions in each crossing sequence, from the proof of Proposition 4, we know that $\mathbb{S}$**PreRec** holds for k-2SPTs, with

$$\ell_s(|T|_s, |\mathcal{A}|_s) = \mathcal{O}((|T|_s)^{k+1}(|\mathcal{A}|_s^{|T|_s})^{k+1})$$

and

$$\ell_d(|T|_s, |T|_d, |\mathcal{A}|_s, |\mathcal{A}|_d) = \mathcal{O}(|T|_d|\mathcal{A}|_d).$$

□

From Theorem 7 and Lemma 4, we have the following result.

Theorem 8. *The path feasibility of symbolic execution S of programs with data strings can be decided*

– *in nondeterministic* $\mathrm{Tower}(\mathsf{asgn}(S)+1, \mathcal{O}(|S|^2))+\beta(\mathrm{Tower}(\mathsf{asgn}(S)+1, \mathcal{O}(|S|^2)))$ *space if the data string functions are given by 2SPTs;*
– *in nondeterministic* $|S|^{|S|^{\mathcal{O}(\mathsf{asgn}(S))}}+\beta(|S|^{|S|^{\mathcal{O}(\mathsf{asgn}(S))}})$ *space if the data string functions are given by k-RB2SPTs.*

Proof. Let us first consider 2SPTs. From Lemma 4, we deduce that

$$\begin{aligned}
&\ell_s^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fasize}_s(S)) \\
&= \mathrm{Tower}(\mathsf{asgn}(S), \mathcal{O}((\mathsf{fnsize}_s(S))^2(\mathsf{fasize}_s(S))^{\mathsf{fnsize}_s(S)} \log((\mathsf{fnsize}_s(S))^2(\mathsf{fasize}_s(S))^{\mathsf{fnsize}_s(S)}))) \\
&\leq \mathrm{Tower}(\mathsf{asgn}(S)+1, \mathcal{O}(|S|^2)).
\end{aligned}$$

and

$$(\ell_d)_{\ell_s}^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_c(S), \mathsf{fnsize}_d(S), \mathsf{fasize}_s(S), \mathsf{fasize}_d(S)) \leq \mathrm{Tower}(\mathsf{asgn}(S), \mathcal{O}(|S|^2)).$$

Therefore, according to Theorem 7, the path feasibility problem for 2SPTs can be solved in nondeterministic

$$\mathrm{Tower}(\mathsf{asgn}(S)+1, \mathcal{O}(|S|^2)) + \beta(\mathrm{Tower}(\mathsf{asgn}(S)+1, \mathcal{O}(|S|^2)))$$

space.

For k-RB2SPTs,

$$\ell_s^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fasize}_s(S)) = |S|^{|S|^{\mathcal{O}(\mathsf{asgn}(S))}},$$

and

$$(\ell_d)_{\ell_s}^{\langle \mathsf{asgn}(S)\rangle}(\mathsf{fnsize}_s(S), \mathsf{fnsize}_d(S), \mathsf{fasize}_s(S), \mathsf{fasize}_d(S)) = |S|^{\mathcal{O}(\mathsf{asgn}(S))}.$$

From Theorem 7, the path feasibility problem for k-RB2SPTs can be solved in nondeterministic $|S|^{|S|^{\mathcal{O}(\mathsf{asgn}(S))}} + \beta(|S|^{|S|^{\mathcal{O}(\mathsf{asgn}(S))}})$ space. □