The Journal of Systems & Software 201 (2023) 111673

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

A symbolic algorithm for the case-split rule in solving word constraints with extensions☆

Yu-Fang Chen [a], Vojtěch Havlena [b], Ondřej Lengál [b,*], Andrea Turrini [c,d]

[a] Institute of Information Science, Academia Sinica, 128 Academia Road, Section 2, Nangang, 11500, Taipei, Taiwan
[b] Faculty of Information Technology, Brno University of Technology, Bozetechova 2, 61200, Brno, Czech Republic
[c] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Haidian District, Zhongguancun 4# South Fourth Street, 100190, Beijing, China
[d] Institute of Intelligent Software, 221, Nansha Street West, 511458, Guangzhou, China

ARTICLE INFO

Article history:
Received 2 December 2021
Received in revised form 13 November 2022
Accepted 28 February 2023
Available online 5 March 2023

Keywords:
String constraints
Satisfiability modulo theories
Regular model checking
Nielsen transformation
Finite automata
Monadic second-order logic over strings

ABSTRACT

Case split is a core proof rule in current decision procedures for the theory of string constraints. Its use is the primary cause of the state space explosion in string constraint solving, since it is the only rule that creates branches in the proof tree. Moreover, explicit handling of the *case split* rule may cause recomputation of the same tasks in multiple branches of the proof tree. In this paper, we propose a symbolic algorithm that significantly reduces such a redundancy. In particular, we encode a string constraint as a regular language and proof rules as rational transducers. This allows us to perform similar steps in the proof tree only once, alleviating the state space explosion. We also extend the encoding to handle arbitrary Boolean combinations of string constraints, length constraints, and regular constraints. In our experimental results, we validate that our technique works in many practical cases where other state-of-the-art solvers fail to provide an answer; our Python prototype implementation solved over 50% of string constraints that could not be solved by the other tools.

1. Introduction

Constraint solving is a technique used as an enabling technology in many areas of formal verification and analysis, such as symbolic execution (Cadar et al., 2006; Godefroid et al., 2005; King, 1976; Sen et al., 2013), static analysis (Wang et al., 2017; Gulwani et al., 2008), or synthesis (Gulwani et al., 2011; Osera, 2019; Knoth et al., 2019). For instance, in symbolic execution, feasibility of a path in a program is tested by creating a constraint that encodes the evolution of the values of variables on the given path and checking if it is satisfiable. Due to the features used in the analyzed programs, checking satisfiability of the constraint can be a complex task. For instance, the solver has to deal with different data types, such as Boolean, Integer, Real, or String. Theories for the first three data types are well known, widely developed, and implemented in tools, while the theory for the String data type has started to be investigated only recently in Abdulla et al. (2014), Berzish et al. (2017), Bjørner et al. (2009), Chen et al. (2019a, 2018), Holík et al. (2018), Lin and Majumdar (2018), Liang et al. (2014), Wang et al. (2016), Yu et al. (2014), Abdulla et al. (2017, 2015), Kiezun et al. (2012), Lin and Barceló (2016), Berzish et al. (2021), Reynolds et al. (2019), Blotsky et al. (2018), Stanford et al. (2021), Loring et al. (2019), Trinh et al. (2020) and Chen et al. (2019b), despite having been considered already by A. A. Markov in the late 1960s in connection with Hilbert's 10th problem (see, e.g., Matiyasevich (1968), Durnev and Zetkina (2009) and Kosovskii (1976)).

Most current decision procedures for string constraints involve the so-called *case-split* rule. This rule performs a case split with respect to the possible alignment of the variables. The case-split rule is used in most, if not all, (semi-)decision procedures for string constraints, including Makanin's algorithm in Makanin (1977), Nielsen transformation (Nielsen, 1917) (also known as the Levi's lemma Levi, 1944), and the procedures implemented in most state-of-the-art solvers such as Z3 (Bjørner et al., 2009), CVC4 (Liang et al., 2014), Z3Str3 (Berzish et al., 2017), Norn (Abdulla et al., 2014), and many more. In this paper, we will explain the general idea of our symbolic approach using the Nielsen transformation, which is the simplest of the approaches; nonetheless, we believe that the approach is applicable also to other procedures.

Consider the *word equation* $xz = yw$, the primary type of *atomic string constraints* considered in this paper, where x, z, y, and w are *word variables*. When establishing satisfiability of the word equation, the Nielsen transformation (introduced in Nielsen

☆ Editor: Earl Barr.

* Corresponding author.
E-mail addresses: yfc@iis.sinica.edu.tw (Y.-F. Chen), ihavlena@fit.vutbr.cz (V. Havlena), lengal@fit.vutbr.cz (O. Lengál), turrini@ios.ac.cn (A. Turrini).

https://doi.org/10.1016/j.jss.2023.111673
0164-1212/

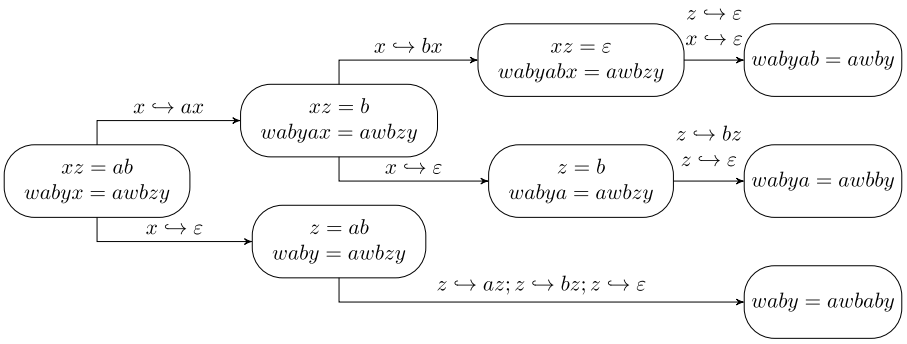


Fig. 1. A partial proof tree of applying the Nielsen transformation on the string constraint $xz = ab \wedge wabyx = awbzy$. The leaves are the outcome of completely processing the first word equation $xz = ab$. Branches leading to contradictions are omitted.

(1917)) proceeds by first performing a case split based on the possible alignments of the variables x and y, the first symbol of the left and right-hand sides of the equation, respectively. More precisely, it reduces the satisfiability problem for $xz = yw$ into satisfiability of (at least) one of the following four cases (1) y is a prefix of x, (2) x is a prefix of y, (3) x is an empty string, and (4) y is an empty string. Note that these cases are not disjoint: for instance, the empty string is a prefix of every variable. For these cases, the Nielsen transformation generates the following equations.

For the case (1), i.e., y is a prefix of x, all occurrences of x in $xz = yw$ are replaced with yx', where x' is a fresh word variable (we denote this case as $x \hookrightarrow yx'$), i.e., we obtain the equation $yx'z = yw$, which can be simplified to $x'z = w$. In fact, since the transformation $x \hookrightarrow yx'$ removes all occurrences of the variable x, we can just reuse the variable x and perform the transformation $x \hookrightarrow yx$ instead (and take this into account when constructing a model later).

Case (2) of the Nielsen transformation is just a symmetric counterpart of case (1) discussed above. For cases (3) and (4), x and y, respectively, are replaced by empty strings. Taking into account all four possible transformations of the equation $xz = yw$, we obtain the following equations:

$$(1)\ xz = w \quad (2)\ z = yw \quad (3)\ z = yw \quad (4)\ xz = w$$

(Note that the results for (1) and (4) coincide, as well as the results for (2) and (4).) If $xz = yw$ has a solution, then at least one of the above equations has a solution, too. The Nielsen transformation keeps applying the transformation rules on the obtained equations, building a proof tree and searching for a tautology of the form $\epsilon = \epsilon$.

Treating each of the obtained equations separately can cause some redundancy (as we could already see above). Let us consider the example in Fig. 1, where we apply the Nielsen transformation to solve the string constraint $xz = ab \wedge wabyx = awbzy$, where x, z, w, and y are word variables and a and b are constant symbols. After processing the first word equation $xz = ab$, we obtain a proof tree with three very similar leaf nodes $wabyab = awby$, $wabya = awbby$, and $waby = awbaby$, which share the prefixes $waby$ and awb on the left and right-hand side of the equations, respectively. If we continue applying the Nielsen transformation on the three leaf nodes, we will create three very similar subtrees, with almost identical operations. In particular, the nodes near the root of such subtrees, which transform $waby\ldots = awb\ldots$, are going to be essentially the same. The resulting proof trees will therefore start to differ only after processing such a common part. Therefore, handling those equations separately will cause that some operations will be performed multiple times. If the proof tree of each word equation has n leaves and the string constraint is a conjunction of k word equations, we might need to create n^k similar subtrees.

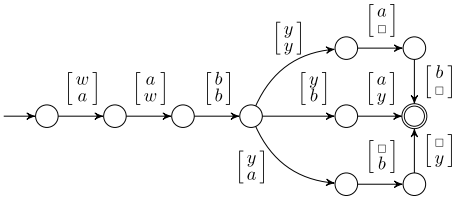


Fig. 2. A finite automaton encoding the three equations $wabyab = awby$, $wabya = awbby$, and $waby = awbaby$.

The case split can be performed more efficiently if we process the common part of the said leaves together using a symbolic encoding. In this paper, we use an encoding of a set of equations as a regular language, which is represented by a *finite automaton*. An example is given in Fig. 2, which shows a finite automaton over a 2-track alphabet, where each of the two tracks represents one side of the equation. For instance, the equation $wabyab = awby$ is represented by the word $\left[\begin{smallmatrix}w\\a\end{smallmatrix}\right]\left[\begin{smallmatrix}a\\w\end{smallmatrix}\right]\left[\begin{smallmatrix}b\\b\end{smallmatrix}\right]\left[\begin{smallmatrix}y\\y\end{smallmatrix}\right]\left[\begin{smallmatrix}a\\\square\end{smallmatrix}\right]\left[\begin{smallmatrix}b\\\square\end{smallmatrix}\right]$ accepted by the automaton, where the $\square$ symbol is a padding used to make sure that both tracks are of the same length.

Given our regular language-based symbolic encoding, we need a mechanism to perform the Nielsen transformation steps on a set of equations encoded as a regular language. We show that the transformations can be encoded as *rational relations*, represented using *finite transducers*, and the whole satisfiability checking problem can be encoded within the framework of *regular model checking*. We will provide more details on how this is done in Sections 3–6 stepwise. In Section 3, we describe the approach for a simpler case where the input is a *quadratic word equation*, i.e., a word equation with at most two occurrences of every variable. In this case, the Nielsen transformation is sound and complete, that is, the solution it returns is correct and it returns a solution whenever a solution exists. In Section 4, we extend the technique to support the *conjunction* of *non-quadratic* word equations. In Section 5, we extend our approach to support arbitrary Boolean combination of string constraints. Section 6 extends our framework with two additional types of atomic string constraints—*length* and *regular constraints*—which can constrain the length of values assigned to word variables and their membership in a regular language, respectively.

We have implemented our approach in a prototype Python tool called Retro and evaluated its performance on two benchmark sets: $\texttt{Kepler}_{22}$ obtained from Le and He (2018) and PyEx-Hard obtained by running the PyEx symbolic execution engine on Python programs from Reynolds et al. (2017) and collecting examples on which CVC4 or Z3 fail. Retro solved most of the problems in $\texttt{Kepler}_{22}$ (on which CVC4 and Z3 do not perform well). Moreover, it solved over 50% of the benchmarks in PyEx-Hard that could be solved by neither CVC4 nor Z3.

This paper is an extended version of the paper that appeared in the proceedings of APLAS'20 (Chen et al., 2020), containing complete proofs of the presented lemmas and theorems and further extending the presented technique to handle (i) arbitrary Boolean combination of string constraints (Section 5), (ii) length constraints (Section 6.1), and (iii) regular constraints (Section 6.2).

2. Preliminaries

An *alphabet* Σ is a finite set of *characters* and a *word* over Σ is a sequence $w = a_1 \dots a_n$ of characters from Σ, with ϵ denoting the *empty word*. We use $w_1.w_2$ (and often just w_1w_2) to denote the *concatenation* of words w_1 and w_2. Σ^* is the set of all words over Σ, $\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$, and $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$. A *language* over Σ is a subset L of Σ^*. Given a word $w = a_1 \dots a_n$, we use $|w|$ to denote the length n of w and $|w|_a$ to denote the number of occurrences of the character $a \in \Sigma$ in w. Further, we use $w[i]$ to denote a_i, the ith character of w, and $w[i:]$ to denote the word $a_i \dots a_n$. When $i > n$, the value of $w[i]$ and $w[i:]$ is in both cases $\perp$, a special *undefined* value, which is different from all other values and also from itself (i.e., $\perp \neq \perp$). We use Σ^k for $k \geq 2$ to denote the *stacked alphabet* consisting of k-tuples of symbols from Σ, e.g., $\left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right] \in \Sigma^2$ for $a, b \in \Sigma$.

2.1. Automata and transducers

A *(finite) k-tape transducer* is a quintuple $\mathcal{T} = (Q, \Sigma, \Delta, Q_i, Q_f)$ such that Q is a finite set of *states*, Σ is an alphabet, $\Delta \subseteq Q \times \Sigma_\epsilon^k \times Q$ is a set of *transitions* of the form $q \text{-}\!\!\left\{a^1,\dots,a^k\right\}\!\!\rightarrow s$ for $a^1, \dots, a^k \in \Sigma_\epsilon$, $Q_i \subseteq Q$ is a set of *initial states*, and $Q_f \subseteq Q$ is a set of *final states*. A run π of $\mathcal{T}$ over a k-tuple of words $(w_1, \dots, w_k)$ is a sequence of transitions $q_0 \text{-}\!\!\left\{a_1^1,\dots,a_1^k\right\}\!\!\rightarrow q_1$, $q_1 \text{-}\!\!\left\{a_2^1,\dots,a_2^k\right\}\!\!\rightarrow q_2$, $\dots$, $q_{n-1} \text{-}\!\!\left\{a_n^1,\dots,a_n^k\right\}\!\!\rightarrow q_n \in \Delta$ such that for each $i \in [1, k]$ we have $w_i = a_1^i a_2^i \dots a_n^i$ (note that a_m^i can be ϵ, so w_i and w_j may be of a different length, for $i \neq j$). The run π is *accepting* if $q_0 \in Q_i$ and $q_n \in Q_f$, and a k-tuple $(w_1, \dots, w_k)$ is *accepted* by $\mathcal{T}$ if there exists an accepting run of $\mathcal{T}$ over $(w_1, \dots, w_k)$. The *language* $\mathcal{L}(\mathcal{T})$ of $\mathcal{T}$ is defined as the k-ary relation $\mathcal{L}(\mathcal{T}) = \{(w_1, \dots, w_k) \in (\Sigma^*)^k \mid (w_1, \dots, w_k)$ is accepted by $\mathcal{T}\}$. We call the class of relations accepted by transducers *rational relations*. $\mathcal{T}$ is *length-preserving* if no transition in Δ contains ϵ; the class of relations accepted by length-preserving transducers is named as *regular relations*. For a 2-tape transducer $\mathcal{T}$ and $(w_1, w_2) \in \mathcal{L}(\mathcal{T})$, we denote w_1 as an input and w_2 as an output of $\mathcal{T}$. A *finite automaton* (FA) is a 1-tape finite transducer; languages accepted by finite automata are called *regular languages*. See, e.g., Pin (2021) for more details on automata and transducers.

Given two k-ary relations R_1, R_2, we define their *concatenation* $R_1.R_2 = \{(u_1v_1, \dots, u_kv_k) \in (\Sigma^*)^k \mid (u_1, \dots, u_k) \in R_1 \wedge (v_1, \dots, v_k) \in R_2\}$ and given two binary relations R_1, R_2, we define their *composition* $R_1 \circ R_2 = \{(x, z) \in (\Sigma^*)^2 \mid \exists y \in \Sigma^* : (x, y) \in R_2 \wedge (y, z) \in R_1\}$. Given a k-ary relation R we define $R^0 = \{\epsilon\}^k$, $R^{i+1} = R.R^i$ for $i \geq 0$. *Iteration* of R is then defined as $R^* = \bigcup_{i \geq 0} R^i$. Given a language $\mathcal{L} \subseteq \Sigma^*$ and a binary relation R, we use $R(\mathcal{L})$ to denote the language $\{y \in \Sigma^* \mid \exists x \in \mathcal{L} : (x, y) \in R\}$, called the *$R$-image* of $\mathcal{L}$. We also use $R^{-1}(w)$ to denote the language $\{u \mid (w, u) \in R\}$, called the *preimage* of a word w.

Proposition 1 (*Berstel, 1979*)**.** *The following propositions hold:*

(i) The class of binary rational relations is closed under (finite) union, composition, concatenation, and iteration; it is not closed under intersection and complement.
(ii) For a binary rational relation R, a regular language $\mathcal{L}$, and a word w, the languages $R(\mathcal{L})$ and $R^{-1}(w)$ are also effectively regular (i.e., they can be computed).
(iii) The class of regular relations is closed under Boolean operations.

2.2. String constraints

Let Σ be an alphabet and $\mathbb{X}$ be a set of *word variables* ranging over Σ^* s.t. $\mathbb{X} \cap \Sigma = \emptyset$. We use $\Sigma_\mathbb{X}$ to denote the extended alphabet $\Sigma \cup \mathbb{X}$. An *assignment of* $\mathbb{X}$ is a mapping $I : \mathbb{X} \to \Sigma^*$. A *word term* is a string over the alphabet $\Sigma_\mathbb{X}$. We lift an assignment I to word terms by defining $I(\epsilon) = \epsilon$, $I(a) = a$, and $I(x.w) = I(x).I(w)$, for $a \in \Sigma$, $x \in \Sigma_\mathbb{X}$, and $w \in \Sigma_\mathbb{X}^*$. A *word equation* φ_e is of the form $t_1 = t_2$ where t_1 and t_2 are word terms. I is a *model* of φ_e if $I(t_1) = I(t_2)$. We call a word equation an *atomic string constraint*. A *string constraint* is obtained from atomic string constraints using Boolean connectives ($\wedge, \vee, \neg$), with the semantics defined in the standard manner. A string constraint is *satisfiable* if it has a model. Given a word term $t \in \Sigma_\mathbb{X}^*$, a variable $x \in \mathbb{X}$, and a word term $u \in \Sigma_\mathbb{X}^*$, we use $t[x \mapsto u]$ to denote the word term obtained from t by replacing all occurrences of x by u, e.g. $(abxcxy)[x \mapsto cy] = abcyccyy$. We call a string constraint ψ *quadratic* if each variable has at most two occurrences in ψ, and *cubic* if each variable has at most three occurrences in ψ.

We use the following terminology. Let Φ be a string constraint. A (semi-)algoritm **A** for solving string constraints is

1. *sound* if it holds that if **A** returns an assignment I to the variables of Φ, then I is a model of Φ,
2. *complete* if it holds that if Φ is satisfiable, then **A** returns a model of Φ in a finite number of steps, and
3. *terminating* if it holds that **A** always returns an assignment or `false` in a finite number of steps.

2.3. Monadic second-order logic on strings (MSO(Str))

We define *monadic second-order logic on strings* (MSO(Str)) (Büchi, 1960) over the alphabet Γ as follows. Let $\mathbb{W}$ be a countable set of *string variables* whose values range over Γ^* and $\mathbb{P}$ be a countable set of *set (second-order) position variables* whose values range over finite subsets of $\mathbb{N}_1 = \mathbb{N} \setminus \{0\}$ such that $\mathbb{W} \cap \mathbb{P} = \emptyset$. A formula φ of MSO(Str) is defined as

$$\varphi ::= P \subseteq R \mid P = R + 1 \mid w[P] = a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \forall^\mathbb{P} P(\varphi) \mid \forall^\mathbb{W} w(\varphi)$$

where $P, R \in \mathbb{P}$, $w \in \mathbb{W}$, and $a \in \Gamma$. We use $\varphi(w_1, \dots, w_k)$ to denote that the free variables of φ are contained in $\{w_1, \dots, w_k\}$.

The semantics of MSO(Str) is defined in Fig. 3. An MSO(Str) *variable assignment* is an assignment $\sigma : \mathbb{W} \cup \mathbb{P} \to (\Gamma^* \cup 2^{\mathbb{N}_1})$ that respects the types of variables with the additional requirement that for every $u, v \in \mathbb{W}$ we have $|\sigma(u)| = |\sigma(v)|$. (We often omit unused variables in σ.) We use $|\sigma|$ to denote the value $|\sigma(w)|$ of any $w \in \mathbb{W}$. Note that $|\sigma|$ is well defined since we assume that all $w \in \mathbb{W}$ are mapped to strings of the same length. The notation $\sigma[x \mapsto v]$ denotes a variant of σ where the assignment of variable x is changed to the value v.

We call an MSO(Str) formula a *string formula* if it contains no free position variables. Such a formula (with k free string variables) denotes a k-ary relation over Γ^*. In particular, given an MSO(Str) string formula $\varphi(w_1, \dots, w_k)$ with k free string variables $w_1, \dots, w_k$, we use $\mathcal{L}(\varphi)$ to denote the relation $\{(x_1, \dots, x_k) \in (\Gamma^*)^k \mid \{w_1 \mapsto x_1, \dots, w_k \mapsto x_k\} \models \varphi\}$. In the special case of $k = 1$, φ denotes a language $\mathcal{L}(\varphi) \subseteq \Gamma^*$.

Proposition 2 (*Thatcher and Wright, 1968*)**.** *The class of languages denoted by* MSO(Str) *string formulae with* 1 *free string variable is exactly the class of regular languages. Furthermore, the class of relations denoted by* MSO(Str) *string formulae with k free string variables, for $k > 1$, is exactly the class of regular relations.*

$$
\begin{array}{ll}
\sigma \models P \subseteq R & \text{iff } \sigma(P) \text{ is a subset of } \sigma(R) \\
\sigma \models P = R + 1 & \text{iff } \sigma(P) = \{r + 1 \mid r \in \sigma(R) \text{ and } r + 1 \leq |\sigma|\} \\
\sigma \models w[P] = a & \text{iff for all } p \in P \text{ it holds that } \sigma(w)[p] \text{ is } a \\
\sigma \models \varphi_1 \wedge \varphi_2 & \text{iff } \sigma \models \varphi_1 \text{ and } \sigma \models \varphi_2 \\
\sigma \models \neg\varphi & \text{iff not } \sigma \models \varphi \\
\sigma \models \forall^{\mathbb{P}} P(\varphi) & \text{iff for all } v \subseteq \{1, \ldots, |\sigma|\} \text{ it holds that } \sigma[P \mapsto v] \models \varphi \\
\sigma \models \forall^{\mathbb{W}} w(\varphi) & \text{iff for all } v \in \Gamma^{|\sigma|} \text{ it holds that } \sigma[w \mapsto v] \models \varphi
\end{array}
$$

Fig. 3. Semantics of MSO(Str).

$$
\begin{array}{rcl@{\qquad}rcl}
\varphi \vee \psi & \triangleq & \neg(\varphi \wedge \psi) & \exists^{\mathbb{P}} P(\varphi) & \triangleq & \neg\forall^{\mathbb{P}} P(\neg\varphi) \\
\varphi \rightarrow \psi & \triangleq & \neg\varphi \vee \psi & \exists^{\mathbb{W}} w(\varphi) & \triangleq & \neg\forall^{\mathbb{W}} w(\neg\varphi) \\
P = R & \triangleq & P \subseteq R \ \wedge \ R \subseteq P & P = \emptyset & \triangleq & \forall^{\mathbb{P}} R(P \subseteq R) \\
Sing(P) & \triangleq & \neg(P = \emptyset) \ \wedge \ \forall^{\mathbb{P}} R(R \subseteq P \rightarrow (R = \emptyset \ \vee \ R = P)) & & & \\
p \in R & \triangleq & Sing(p) \wedge p \subseteq R & & & \\
p \leq r & \triangleq & \forall^{\mathbb{P}} T((p \in T \wedge \forall^{\mathbb{P}} u(u \in T \rightarrow \exists^{\mathbb{P}} v(v = u + 1 \wedge v \in T))) \rightarrow r \in T) & & & \\
p < r & \triangleq & p \leq r \wedge \neg(p = r) & & & \\
x = 1 & \triangleq & \forall^{\mathbb{P}} u(u \leq x \rightarrow u = x) & x = \$ & \triangleq & \forall^{\mathbb{P}} u(x \leq u \rightarrow u = x) \\
w_1[P] = w_2[R] & \triangleq & \bigvee_{a \in \Gamma} (w_1[P] = a \wedge w_2[R] = a) & & &
\end{array}
$$

for all $j \geq 1$ and any term γ:

$$
\begin{array}{rcl}
P = R + j & \triangleq & \exists^{\mathbb{P}} S_1 \ldots \exists^{\mathbb{P}} S_j (S_1 = P + 1 \wedge S_2 = S_1 + 1 \wedge \ldots \wedge R = S_j + 1) \\
p = j & \triangleq & \exists^{\mathbb{P}} z(z = 1 \wedge p = z + (j - 1)) \\
w[P + j] = \gamma & \triangleq & \exists^{\mathbb{P}} S(S = P + j \wedge w[S] = \gamma) \\
w[P - j] = \gamma & \triangleq & \exists^{\mathbb{P}} S(P = S + j \wedge w[S] = \gamma) \\
w[\$ - j] = \gamma & \triangleq & \exists^{\mathbb{P}} r \exists^{\mathbb{P}} s(r = \$ \wedge r = s + j \wedge w[s] = \gamma) \\
w[j] = \gamma & \triangleq & \exists^{\mathbb{P}} p(p = j \wedge w[p] = \gamma) \\
p \geq j & \triangleq & \exists^{\mathbb{P}} s(s = j \wedge p \geq s)
\end{array}
$$

Fig. 4. Syntactic sugar for MSO(Str).

Syntactic sugar for MSO(STR). In Fig. 4, we define the standard syntactic sugar to allow us to write more concise MSO(Str) formulae. Most of the sugar is standard, let us, however, explain some of the less standard notation: $Sing(P)$ denotes that P is a singleton set of positions, $p \leq r$ denotes that p and r are single positions and that p is less than or equal to r, $x = 1$ and $x = \$$ denote that x is the first and the last position respectively, and $P = R + j$ denotes that P is equal to R with all positions incremented by j. We also extend our syntax to allow first-order variables (we abuse notation and use the same quantifier notation as for second-order variables, but denote the first-order variable with a lowercase letter):

$\forall^{\mathbb{P}} p(\varphi) \triangleq \forall^{\mathbb{P}} P(Sing(P) \rightarrow \varphi[p \mapsto P])$

$\exists^{\mathbb{P}} p(\varphi) \triangleq \exists^{\mathbb{P}} P(Sing(P) \wedge \varphi[p \mapsto P])$

where $\varphi[p \mapsto P]$ denotes the substitution of all free occurrences of p in φ by P.

2.4. Nielsen transformation

As already briefly mentioned in the introduction, the Nielsen transformation can be used to check satisfiability of a conjunction of word equations. We use the three rules shown in Fig. 5; besides the rules $x \hookrightarrow \alpha x$ and $x \hookrightarrow \epsilon$ that we have seen in the introduction, there is also the (trim) rule, used to remove a shared prefix from both sides of the equation.

$$
\frac{\alpha u = \alpha v}{u = v}\ (\text{trim}) \qquad \frac{xu = v}{u[x \mapsto \varepsilon] = v[x \mapsto \varepsilon]}\ (x \hookrightarrow \varepsilon)
$$

$$
\frac{xu = \alpha v}{x(u[x \mapsto \alpha x]) = v[x \mapsto \alpha x]}\ (x \hookrightarrow \alpha x)
$$

Fig. 5. Rules of the Nielsen transformation, with $x \in \mathbb{X}$, $\alpha \in \Sigma_{\mathbb{X}}$, and $u, v \in \Sigma_{\mathbb{X}}^*$. Symmetric rules are omitted.

Given a system of word equations, multiple Nielsen transformations might be applicable to it, resulting in different transformed equations on which other Nielsen transformations can be performed, as shown in Fig. 1. Trying all possible transformations generates a tree (or a graph in general) whose nodes contain conjunctions of word equations and whose edges are labeled with the applied transformation. The conjunction of word equations in the root of the tree is satisfiable if and only if at least one of the

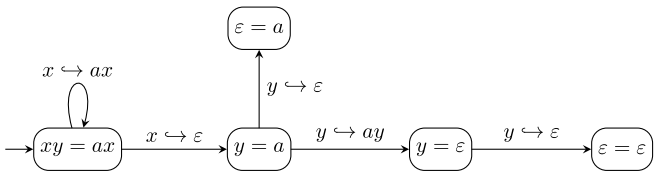


Fig. 6. Proof graph of the equation $xy = ax$ generated by the Nielsen transformation.

leaves in the graph is a tautology, i.e., it contains a conjunction of the form $\epsilon = \epsilon \wedge \cdots \wedge \epsilon = \epsilon$. As an example, consider the satisfiable equation $xy = ax$ where x, y are word variables and a is a symbol with the proof graph in Fig. 6.

Lemma 3 (*cf. Makanin, 1977; Diekert, 2002*)**.** *The Nielsen transformation is sound and complete. Moreover, if the system of word equations is quadratic, the proof graph is finite.*

Lemma 3 is correct even if we construct the proof tree using the following strategy: every application of $x \hookrightarrow \alpha x$ or $x \hookrightarrow \epsilon$ is followed by as many applications of the (trim) rule as possible. We use $x \rightarrowtail \alpha x$ to denote the application of one $x \hookrightarrow \alpha x$ rule followed by as many applications of (trim) as possible, and $x \rightarrowtail \epsilon$ for the application of $x \hookrightarrow \epsilon$ followed by (trim).

2.5. Regular model checking

Regular model checking (RMC) (cf. Kesten et al. (2001), Wolper and Boigelot (1998), Bouajjani et al. (2000), Abdulla (2012) and Bouajjani et al. (2012)) is a framework for verifying infinite state systems. In RMC, each *system configuration* is represented as a word over an alphabet Σ. The set of *initial configurations* $\mathcal{I}$ and *destination configurations* $\mathcal{D}$ are captured as regular languages over Σ. The *transition relation* $\mathcal{T}$ is captured as a binary rational relation over Σ^*. A regular model checking *reachability problem* is represented by the triplet $(\mathcal{I}, \mathcal{T}, \mathcal{D})$ and asks whether $\mathcal{T}^{rt}(\mathcal{I}) \cap \mathcal{D} \neq \emptyset$, where $\mathcal{T}^{rt}$ represents the reflexive and transitive closure of $\mathcal{T}$. One way how to solve the problem is to start computing the sequence $\mathcal{T}^{(0)}(\mathcal{I}), \mathcal{T}^{(1)}(\mathcal{I}), \mathcal{T}^{(2)}(\mathcal{I}), \ldots$ where $\mathcal{T}^{(0)}(\mathcal{I}) = \mathcal{I}$ and $\mathcal{T}^{(n+1)}(\mathcal{I}) = \mathcal{T}(\mathcal{T}^{(n)}(\mathcal{I}))$. During the computation of the sequence, we can check whether we find $\mathcal{T}^{(i)}(\mathcal{I})$ that overlaps with $\mathcal{D}$, and if yes, we can deduce that $\mathcal{D}$ is reachable. On the other hand, if we obtain a sequence such that $\bigcup_{0 \leq i < n} \mathcal{T}^i(\mathcal{I}) \supseteq \mathcal{T}^n(\mathcal{I})$, we know that we have explored all possible system configurations without reaching $\mathcal{D}$, so $\mathcal{D}$ is unreachable. The RMC reachability problem is in general *undecidable* (this can be easily shown, e.g., by a reduction from Turing machine configuration reachability).

3. Solving word equations using RMC

In this section, we describe a symbolic RMC-based framework for solving string constraints. The framework is based on encoding a string constraint into a regular language and encoding steps of the Nielsen transformation as a rational relation. Satisfiability of a string constraint is then reduced to a reachability problem of RMC.

3.1. Nielsen transformation as word operations

In the following, we describe how the Nielsen transformation of a single word equation can be expressed as operations on words. We view a word equation $eq\colon \mathrm{t}_\ell = \mathrm{t}_r$ as a pair of word terms $e_{eq} = (\mathrm{t}_\ell, \mathrm{t}_r)$ corresponding to the left and right hand sides of the equation respectively; therefore $e_{eq} \in \Sigma_{\mathbb{X}}^* \times \Sigma_{\mathbb{X}}^*$. Without loss of generality we assume that $\mathrm{t}_\ell[1] \neq \mathrm{t}_r[1]$; if this is not the case, we pre-process the equation by applying the (trim) Nielsen transformation (cf. Fig. 5) to trim the common prefix of t_ℓ and t_r.

Example 1. The word equation $eq_1\colon xay = yx$ is represented by the pair of word terms $e_1 = (xay, yx)$. The full proof graph generated by applying the Nielsen transformation is depicted in Fig. 7. □

A rule of the Nielsen transformation (cf. Section 2.4) is represented using a (partial) function $\tau : (\Sigma_{\mathbb{X}}^* \times \Sigma_{\mathbb{X}}^*) \to (\Sigma_{\mathbb{X}}^* \times \Sigma_{\mathbb{X}}^*)$. Given a pair of word terms $(\mathrm{t}_\ell, \mathrm{t}_r)$ of a word equation eq, the function τ transforms it into a pair of word terms of a word equation eq' that would be obtained by performing the corresponding step of the Nielsen transformation on eq. Before we express the rules of the Nielsen transformation, we define functions performing the corresponding substitution. For $x \in \mathbb{X}$ and $\alpha \in \Sigma_{\mathbb{X}}$ we define

$$\tau_{x \mapsto \alpha x} = \{ (\mathrm{t}_\ell, \mathrm{t}_r) \mapsto (\mathrm{t}'_\ell, \mathrm{t}'_r) \mid \mathrm{t}'_\ell = \mathrm{t}_\ell[x \mapsto \alpha x] \wedge \mathrm{t}'_r = \mathrm{t}_r[x \mapsto \alpha x] \} \text{ and}$$
$$\tau_{x \mapsto \epsilon} = \{ (\mathrm{t}_\ell, \mathrm{t}_r) \mapsto (\mathrm{t}'_\ell, \mathrm{t}'_r) \mid \mathrm{t}'_\ell = \mathrm{t}_\ell[x \mapsto \epsilon] \wedge \mathrm{t}'_r = \mathrm{t}_r[x \mapsto \epsilon] \}. \quad (1)$$

The function $\tau_{x \mapsto \alpha x}$ performs a substitution $x \mapsto \alpha x$ while the function $\tau_{x \mapsto \epsilon}$ performs a substitution $x \mapsto \epsilon$.

Example 2. Consider the pair of word terms $e_1 = (xay, yx)$ from Example 1. The application $\tau_{x \mapsto yx}(e_1)$ would produce the pair $e_2 = (yxay, yyx)$ and the application $\tau_{x \mapsto \epsilon}(e_1)$ would produce the pair $e_3 = (ay, y)$. □

The functions introduced above do not take into account the first symbols of each side and do not remove a common prefix of the two sides of the equation, which is a necessary operation for the Nielsen transformation to terminate. Let us, therefore, define

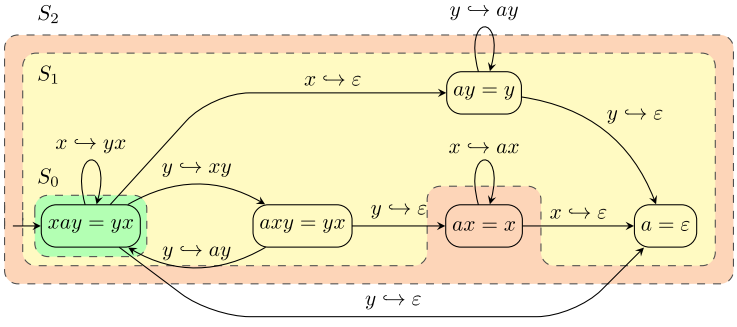


Fig. 7. Proof graph for a run of the Nielsen transformation on the equation $xay = yx$. The sets S_0, S_1, and S_2 are the sets of nodes explored in 0, 1, and 2 steps of our algorithm, respectively.

the following function, which trims (the longest) matching prefix of word terms of the two sides of an equation:

$$\tau_{trim} = \{ (t_\ell, t_r) \mapsto (t'_\ell, t'_r) \mid \exists i \geq 1 \forall j < i \, \big(t_\ell[i] \neq t_r[i] \wedge t_\ell[j] = t_r[j] \wedge t'_\ell = t_\ell[i{:}] \wedge t'_r = t_r[i{:}] \big) \}. \quad (2)$$

Example 3. Continuing in our running example, the application $\tau_{trim}(e_2)$ produces the pair $e'_2 = (xay, yx)$ and, furthermore, $\tau_{trim}(e_3)$ produces the pair $e'_3 = (ay, y)$. □

Now we are ready to define functions corresponding to the rules of the Nielsen transformation. In particular, the rule $x \rightarrowtail \alpha x$ and its symmetric variant (i.e., x is the first symbol of either left or right side of an equation) for $x \in \mathbb{X}$ and $\alpha \in \Sigma_\mathbb{X}$ (cf. Section 2.4) can be expressed using the function

$$\tau_{x \rightarrowtail \alpha x} = \tau_{trim} \circ \{ (t_\ell, t_r) \mapsto \tau_{x \mapsto \alpha x}(t_\ell, t_r) \mid (t_r[1] = x \wedge t_\ell[1] = \alpha) \vee (t_r[1] = \alpha \wedge t_\ell[1] = x) \} \quad (3)$$

while the rule $x \rightarrowtail \epsilon$ and its symmetric variant for $x \in \mathbb{X}$ can be expressed as the function

$$\tau_{x \rightarrowtail \epsilon} = \tau_{trim} \circ \{ (t_\ell, t_r) \mapsto \tau_{x \mapsto \epsilon}(t_\ell, t_r) \mid (t_\ell[1] = x \vee t_r[1] = x) \}. \quad (4)$$

If we keep applying the functions defined above on individual pairs of word terms, while searching for the pair (ϵ, ϵ)—which represents the case when a solution to the original equation eq exists—, we would obtain the Nielsen transformation graph (cf. Section 2.4). In the following, we show how to perform the steps symbolically on a representation of a *set* of word equations at once.

3.2. Symbolic algorithm for word equations

In this section, we describe the main idea of our symbolic algorithm for solving word equations. We first focus on the case of a single word equation and in subsequent sections extend the algorithm to a richer class.

Our algorithm is based on applying the transformation rules not on a single equation, but on a whole *set of equations* at once. For this, we define the relations $\mathcal{T}_{x \rightarrowtail \alpha x}$ and $\mathcal{T}_{x \rightarrowtail \epsilon}$ that aggregate the versions of $\tau_{x \rightarrowtail \alpha x}$ and $\tau_{x \rightarrowtail \epsilon}$ for all possible $x \in \mathbb{X}$ and $\alpha \in \Sigma_\mathbb{X}$. The signature of these relations is $(\Sigma^*_\mathbb{X} \times \Sigma^*_\mathbb{X}) \times (\Sigma^*_\mathbb{X} \times \Sigma^*_\mathbb{X})$ and they are defined as follows:

$$\mathcal{T}_{x \rightarrowtail \alpha x} = \bigcup_{y \in \mathbb{X}, \alpha \in \Sigma_\mathbb{X}} \tau_{y \rightarrowtail \alpha y} \qquad \mathcal{T}_{x \rightarrowtail \epsilon} = \bigcup_{y \in \mathbb{X}} \tau_{y \rightarrowtail \epsilon} \quad (5)$$

Note the following two properties of the relations: (i) they produce outputs of all possible Nielsen transformation steps applicable with the first symbols on the two sides of the equations and (ii) they include the *trimming* operation.

We compose the introduced relations into a single one, denoted as $\mathcal{T}_{step}$ and defined as $\mathcal{T}_{step} = \mathcal{T}_{x \rightarrowtail \alpha x} \cup \mathcal{T}_{x \rightarrowtail \epsilon}$. The relation $\mathcal{T}_{step}$ can then be used to compute *all successors* of a set of word terms of equations in one step. For a set of word terms S we can compute the $\mathcal{T}_{step}$-image of S to obtain all successors of pairs of word terms in S. The initial configuration, given a word equation $eq : t_\ell = t_r$, is the set $E_{eq} = \{(t_\ell, t_r)\}$.

Example 4. Lifting our running example to the introduced notions over sets, we start with the set $E_{eq} = S_0 = \{e_1 = (xay, yx)\}$. After applying $\mathcal{T}_{step}$ on E_{eq}, we obtain the set $S_1 = \{e'_2 = (xay, yx), e'_3 = (ay, y), (axy, yx), (a, \epsilon)\}$. The pairs e'_2 and e'_3 were described earlier, the pair (axy, yx) is obtained by the transformation $\tau_{y \rightarrowtail xy}$, and the pair (a, ϵ) is obtained by the transformation $\tau_{y \rightarrowtail \epsilon}$. If we continue by computing $\mathcal{T}_{step}(S_1)$, we obtain the set $S_2 = S_1 \cup \{(ax, x)\}$, as shown in Fig. 7 (the pair (ax, x) was obtained from (axy, yx) using the transformation $\tau_{y \rightarrowtail \epsilon}$). □

Using the symbolic representation, we can formulate the problem of checking satisfiability of a word equation eq as the task of

- either testing whether $(\epsilon, \epsilon) \in \mathcal{T}^{rt}_{step}(E_{eq})$; if the membership holds, it means that the constraint eq is satisfiable, or
- finding a set (called *unsat-invariant*) E_{inv} such that $E_{eq} \subseteq E_{inv}$, $(\epsilon, \epsilon) \notin E_{inv}$, and $\mathcal{T}_{step}(E_{inv}) \subseteq E_{inv}$, implying that eq is unsatisfiable.

In the following sections, we show how to encode the problem into the RMC framework.

Example 5. To proceed in our running example, when we apply $\mathcal{T}_{step}$ on S_2, we get $\mathcal{T}_{step}(S_2) \subseteq S_2$. Since $e_1 \in S_2$ and $(\epsilon, \epsilon) \notin S_2$, the set S_2 is our unsat-invariant, which means that eq_1 is unsatisfiable. □

3.3. Towards symbolic encoding

Let us now discuss some possible encodings of the word equations satisfiability problem into RMC. Recall that our task is to find an encoding such that the encoded equation (corresponding to initial configurations in RMC) and satisfiability condition (corresponding to destination configurations) are regular languages and the transformation (transition) relation is a rational relation. We start by describing two possible methods of encodings that do not work, analyze why they cannot be used, and then describe a working encoding that we do use.

The first idea about how to encode a set of word equations as a regular language is to encode a pair $e_{eq} = (t_\ell, t_r)$ as a word $t_\ell \cdot \circledcirc \cdot t_r$, where $\circledcirc \notin \Sigma_\mathbb{X}$. One immediately finds out that although the transformations $\tau_{x \rightarrowtail \alpha x}$ and $\tau_{x \rightarrowtail \epsilon}$ are rational (i.e., expressible using a transducer), the transformation τ_{trim}, which removes the longest matching prefix from both sides, is not (a transducer with an unbounded memory to remember the prefix would be required).

The second attempt of an encoding might be to encode $e_{eq} = (t_\ell, t_r)$ as a rational binary relation, represented, e.g., by a (not necessarily length-preserving) 2-tape transducer (with one tape for t_ℓ and the other tape for t_r) and use four-tape transducers to represent the transformations (with two tapes for t_ℓ and t_r and two tapes for t'_ℓ and t'_r). The transducers implementing $\tau_{x \rightarrowtail yx}$ and $\tau_{x \rightarrowtail \epsilon}$ can be constructed easily and so can be the transducer implementing τ_{trim}, so this solution looks appealing. One, however, quickly realizes that there is an issue in computing $\mathcal{T}_{step}(E_{eq})$. In particular, since E_{eq} and $\mathcal{T}_{step}$ are both represented as rational relations, the intersection $(E_{eq} \times \Sigma^*_\mathbb{X} \times \Sigma^*_\mathbb{X}) \cap \mathcal{T}_{step}$, which needs to be computed first, may not be rational any more. Why? Suppose $E_{eq} = \{(a^m b^n, c^m) \mid m, n \geq 0\}$ and $\mathcal{T}_{step} = \{(a^m b^n, c^n, \epsilon, \epsilon) \mid m, n \geq 0\}$. Then the intersection $(E_{eq} \times \Sigma^*_\mathbb{X} \times \Sigma^*_\mathbb{X}) \cap \mathcal{T}_{step} = \{(a^n b^n, c^n, \epsilon, \epsilon) \mid n \geq 0\}$ is clearly not rational any more.

3.4. Symbolic encoding of quadratic equations into RMC

We therefore converge on the following method of representing word equations by a regular language. A set of pairs of word terms is represented as a regular language over a 2-track alphabet with padding $\Sigma^2_{\mathbb{X},\square}$, where $\Sigma_{\mathbb{X},\square} = \Sigma_\mathbb{X} \cup \{\square\}$, using an FA. For instance, $e_1 = (xay, yx)$ would be represented by the regular language $\left[\begin{smallmatrix} x & a & y \\ y & x & \square \end{smallmatrix}\right]\left[\begin{smallmatrix} \square \\ \square \end{smallmatrix}\right]^*$. In other words, the equation e_1 has

many encodings that differ by the padding, with $\left[\begin{smallmatrix} x & a & y \\ y & x & \square \end{smallmatrix}\right]$ being the shortest encoding. The valid representation of the equation contains all of these encodings. On the other hand, Nielsen transformations are represented by (in general, length non-preserving) *binary* rational relations over the 2-track alphabet $\Sigma^2_{\mathbb{X},\square}$ (the first item of each pair refers to an encoding of an equation and the second one refers to the particular transformation applied to the encoding). For instance, the transformation $\tau_{x \mapsto \epsilon}$ is represented by a rational relation containing, e.g., the pair $\left(\left[\begin{smallmatrix} x & a & y & \square \\ y & x & \square & \square \end{smallmatrix}\right], \left[\begin{smallmatrix} a & y & \square \\ y & \square & \square \end{smallmatrix}\right]\right)$ and the transformation $\tau_{y \mapsto xy}$ is represented by a rational relation containing, e.g., the pair $\left(\left[\begin{smallmatrix} x & a & y & \square \\ y & x & \square & \square \end{smallmatrix}\right], \left[\begin{smallmatrix} a & x & y \\ y & x & \square \end{smallmatrix}\right]\right)$.

Formally, we first define the *equation encoding function* $\mathsf{eqencode} : (\Sigma^*_{\mathbb{X}})^2 \to (\Sigma^2_{\mathbb{X},\square} \setminus \{\left[\begin{smallmatrix} \square \\ \square \end{smallmatrix}\right]\})^*$ such that for a pair of word terms $\mathtt{t}_\ell = a_1 \dots a_n$ and $\mathtt{t}_r = b_1 \dots b_m$ (without loss of generality we assume that $n \geq m$), we have $\mathsf{eqencode}(\mathtt{t}_\ell, \mathtt{t}_r) = \left[\begin{smallmatrix} a_1 & \cdots & a_m & a_{m+1} & \cdots & a_n \\ b_1 & \cdots & b_m & \square & \cdots & \square \end{smallmatrix}\right]$. We also lift eqencode to sets of pairs of word terms $S \subseteq \Sigma^*_{\mathbb{X}} \times \Sigma^*_{\mathbb{X}}$ as $\mathsf{eqencode}(S) = \{\mathsf{eqencode}(\mathtt{t}_\ell, \mathtt{t}_r) \mid (\mathtt{t}_\ell, \mathtt{t}_r) \in S\}$.

Let σ be a symbol. We define the *padding* of a word w with respect to σ as the language $\mathsf{pad}_\sigma = \{(w, w') \mid w' \in \{w\}.\{\sigma\}^*\}$, i.e., it is a set of words obtained from w by extending it by an arbitrary number of σ's. Moreover, we also create a (length non-preserving) transducer T_{trim} that performs trimming of its input; this is easy to implement by a two-state transducer that replaces a prefix of symbols of the form $\left[\begin{smallmatrix} \beta \\ \beta \end{smallmatrix}\right]$ with ϵ, for $\beta \in \Sigma_{\mathbb{X},\square}$. We define the function encode, used for encoding word equations into regular languages, as $\mathsf{encode} = T_{trim} \circ \mathsf{pad}_{\left[\begin{smallmatrix} \square \\ \square \end{smallmatrix}\right]} \circ \mathsf{eqencode}$, i.e., it takes an encoding of the equation, adds padding, and trims the maximum shared prefix of the two sides of the equation. For example, $\mathsf{encode}(bxay, byx) = \left[\begin{smallmatrix} x & a & y \\ y & x & \square \end{smallmatrix}\right]\left[\begin{smallmatrix} \square \\ \square \end{smallmatrix}\right]^*$. Moreover, for an equation $e \in \Sigma^*_{\mathbb{X}} \times \Sigma^*_{\mathbb{X}}$, a word $w \in \mathsf{encode}(e)$, and a Nielsen rule ρ, we use $w[\rho]$ to denote the set $\mathsf{encode}(\tau_\rho(e))$.

Lemma 4. *Given a word equation $eq : \mathtt{t}_\ell = \mathtt{t}_r$ for $\mathtt{t}_\ell, \mathtt{t}_\ell \in \Sigma^*_{\mathbb{X}}$, the set* encode($eq$) *is regular.*

Proof.

Without loss of generality we assume that $|\mathtt{t}_r| \leq |\mathtt{t}_\ell|$. We give the following MSO(Str) formula that encodes eq:

$$\begin{aligned} \varphi_{eq}(w, w') \triangleq & \bigwedge_{1 \leq k \leq |\mathtt{t}_\ell|} w[k] = \mathtt{t}_\ell[k] \wedge \bigwedge_{1 \leq k \leq |\mathtt{t}_r|} w'[k] = \mathtt{t}_r[k] \\ & \wedge \bigwedge_{|\mathtt{t}_r| < k \leq |\mathtt{t}_\ell|} w'[k] = \square \wedge \\ & \forall^{\mathbb{P}} p((p > |\mathtt{t}_\ell|) \to (w[p] = \square \wedge w'[p] = \square)) \end{aligned} \tag{6}$$

From Proposition 2, it follows that $\mathcal{L}(\varphi_{eq})$ is a regular binary relation and, moreover, it can be interpreted as a regular language over the composed alphabet $\Sigma^2_{\mathbb{X},\square}$. Since the image of a regular language with respect to a rational relation (realizing T_{trim}) is also regular (cf. Proposition 1), it follows that $\mathsf{encode}(\mathcal{L}(\varphi_{eq}))$ is also regular. □

Using the presented encoding, when trying to express the $\tau_{x \mapsto \alpha x}$ and $\tau_{x \mapsto \epsilon}$ transformations, we, however, encounter an issue with the need of an unbounded memory. For instance, for the language $L = \left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]^*$, the transducer implementing $\tau_{x \mapsto yx}$ would need to remember how many times it has seen x on the first track of its input (indeed, the image of L with respect to $\tau_{x \mapsto yx}$, i.e., the set $\{\mathsf{encode}(u, v) \mid \exists n \geq 0 : u = (yx)^n \wedge v = y^n \square^n\}$, is no longer regular).

We address this issue in several steps: first, we give a rational relation that correctly represents the transformation rules for cases where the equation eq is quadratic, and extend our algorithm to equations with more occurrences of variables in Section 4. Let us define the following, more general, restriction of $\tau_{x \mapsto \alpha x}$ to equations with at most $i \in \mathbb{N}$ occurrences of variable x:

$$\tau^{\leq i}_{x \mapsto \alpha x} = \tau_{x \mapsto \alpha x} \cap \{((\mathtt{t}_\ell, \mathtt{t}_r), (w, w')) \mid w, w' \in \Sigma^*_{\mathbb{X}}, |\mathtt{t}_\ell.\mathtt{t}_r|_x \leq i\}. \tag{7}$$

We define $\tau^{\leq i}_{x \mapsto \epsilon}$, $\tau^{\leq i}_{x \mapsto \alpha x}$, and $\tau^{\leq i}_{x \mapsto \epsilon}$ similarly.

3.4.1. Encoding Nielsen transformations as rational relations

Next, in order to be able to perform the operations given by $\tau^{\leq i}_{x \mapsto \epsilon}$ and $\tau^{\leq i}_{x \mapsto \alpha x}$ on our encoding within the RMC framework, we need to encode them as rational relations. In this section, we define rational relations $\mathcal{T}^{\leq i}_{x \mapsto \epsilon}$ and $\mathcal{T}^{\leq i}_{x \mapsto \alpha x}$ that do exactly this encoding. We obtain these relations in successive steps, by defining several intermediate formulae with the transformation as subscript and the number of variables as superscript, e.g., $\varphi^{\leq n}_{x \mapsto \epsilon}$ and $\psi^{n}_{x \mapsto \alpha x}$.

We begin with defining some useful MSO(Str) predicates for an MSO(Str) string variable w, a word constraint variable x, and positions $k_1, \dots, k_m$.

$$ordered(k_1, \dots, k_m) \triangleq \bigwedge_{1 \leq i < m} k_i < k_{i+1} \tag{8}$$

$$alleq^w_x(k_1, \dots, k_m) \triangleq \bigwedge_{1 \leq i \leq m} w[k_i] = x \tag{9}$$

$$\begin{aligned} occur^w_x(k_1, \dots, k_m) \triangleq \; & ordered(k_1, \dots, k_m) \wedge alleq^w_x(k_1, \dots, k_m) \\ & \wedge \forall^{\mathbb{P}} j \Big(w[j] = x \to \bigvee\nolimits_{1 \leq i \leq m} j = k_i \Big) \end{aligned} \tag{10}$$

We use the following MSO(Str) formula to define the transformation $x \mapsto \epsilon$ for n occurrences of x in a single string. The formula guesses n positions of x and then ensures that all symbols in w' are correctly shifted. In particular, the symbols on positions smaller than i_1 are copied from w without change. Symbols in w on positions between i_ℓ and $i_{\ell+1}$ are shifted ℓ positions to the left in w'. And the remaining positions in w' are filled with $\square$'s.

$$\begin{aligned} \psi^n_{x \mapsto \epsilon}(w, w') \triangleq \exists^{\mathbb{P}} i_1, \dots, i_n \Big(& occur^w_x(i_1, \dots, i_n) \wedge \\ & \forall^{\mathbb{P}} j (j < i_1 \to w'[j] = w[j]) \wedge \\ & \bigwedge_{1 \leq k < n} \forall^{\mathbb{P}} j ((i_k < j < i_{k+1}) \to w'[j - k] = w[j]) \wedge \\ & \forall^{\mathbb{P}} j (i_n < j \to w'[j - n] = w[j]) \wedge \\ & \bigwedge_{1 \leq k \leq n} w'[\$ - k] = \square \Big), \end{aligned} \tag{11}$$

where $w'[\$ - k] = \square$ stands for the formula $\exists^{\mathbb{P}} r \exists^{\mathbb{P}} s (r = \$ \wedge r = s + k \wedge w'[s] = \square)$ (cf. Fig. 4). We extend $\psi^n_{x \mapsto \epsilon}$ to describe the relation on pairs of strings:

$$\psi'^{n}_{x \mapsto \epsilon}(\mathtt{t}_\ell, \mathtt{t}_r, \mathtt{t}'_\ell, \mathtt{t}'_r) \triangleq \bigvee_{0 \leq k \leq n} \psi^k_{x \mapsto \epsilon}(\mathtt{t}_\ell, \mathtt{t}'_\ell) \wedge \psi^{n-k}_{x \mapsto \epsilon}(\mathtt{t}_r, \mathtt{t}'_r) \tag{12}$$

$$\psi'^{\leq n}_{x \mapsto \epsilon}(\mathtt{t}_\ell, \mathtt{t}_r, \mathtt{t}'_\ell, \mathtt{t}'_r) \triangleq \bigvee_{0 \leq k \leq n} \psi'^{k}_{x \mapsto \epsilon}(\mathtt{t}_\ell, \mathtt{t}_r, \mathtt{t}'_\ell, \mathtt{t}'_r) \tag{13}$$

$$\varphi^{\leq n}_{x \mapsto \epsilon}(\mathtt{t}_\ell, \mathtt{t}_r, \mathtt{t}'_\ell, \mathtt{t}'_r) \triangleq (\mathtt{t}_\ell[1] = x \vee \mathtt{t}_r[1] = x) \wedge \psi'^{\leq n}_{x \mapsto \epsilon}(\mathtt{t}_\ell, \mathtt{t}_r, \mathtt{t}'_\ell, \mathtt{t}'_r) \tag{14}$$

Next, we define the transformation $x \mapsto \alpha x$ for n occurrences of x in a single string. The formula guesses n positions of x in w. Then, it ensures that on $i_\ell + \ell$ position in w' there is the symbol α and all other symbols from w are copied to the correct positions in

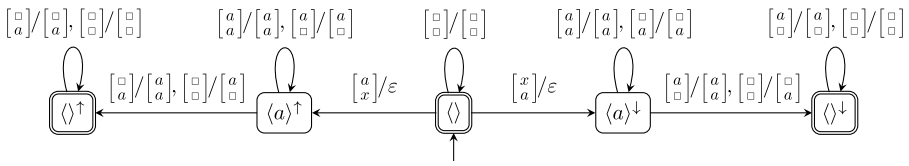


Fig. 8. Example of a transducer realizing the relation $T^{\leq 1}_{x\mapsto\epsilon}$ for the set of variables $\mathbb{X} = \{x\}$ and the alphabet $\Sigma = \{a\}$. The names of the states of the transducers denote symbols that the transducers remember to output on the tape given by the symbols ↑ and ↓.

w'. In particular, symbols in w on positions between i_ℓ and $i_{\ell+1}$ are shifted ℓ positions to the right in w'.

$$\begin{aligned}\psi^n_{x\mapsto\alpha x}(w, w') \triangleq \exists^{\mathbb{P}} i_1, \ldots, i_n \Big(& occur^w_x(i_1, \ldots, i_n) \wedge \\ & \forall^{\mathbb{P}} j (j \leq i_1 \rightarrow w'[j] = w[j]) \wedge \\ & \bigwedge_{1\leq k<n} w'[i_k + k] = \alpha \wedge \\ & \quad \forall^{\mathbb{P}} j ((i_k < j \leq i_{k+1}) \rightarrow w'[j + k] = w[j]) \wedge \\ & w'[i_n + n] = \alpha \wedge \\ & \forall^{\mathbb{P}} j (i_n < j \rightarrow w'[j + n] = w[j]) \wedge \\ & \bigwedge_{1\leq k\leq n} w[\$ - k] = \square \Big)\end{aligned} \tag{15}$$

We extend $\psi^n_{x\mapsto\alpha x}$ to describe the relation on pairs of strings:

$$\psi'^n_{x\mapsto\alpha x}(\mathsf{t}_\ell, \mathsf{t}_r, \mathsf{t}'_\ell, \mathsf{t}'_r) \triangleq \bigvee_{0\leq k\leq n} \psi^k_{x\mapsto\alpha x}(\mathsf{t}_\ell, \mathsf{t}'_\ell) \wedge \psi^{n-k}_{x\mapsto\alpha x}(\mathsf{t}_r, \mathsf{t}'_r) \tag{16}$$

$$\psi'^{\leq n}_{x\mapsto\alpha x}(\mathsf{t}_\ell, \mathsf{t}_r, \mathsf{t}'_\ell, \mathsf{t}'_r) \triangleq \bigvee_{0\leq k\leq n} \psi'^k_{x\mapsto\alpha x}(\mathsf{t}_\ell, \mathsf{t}_r, \mathsf{t}'_\ell, \mathsf{t}'_r) \tag{17}$$

$$\begin{aligned}\varphi^{\leq n}_{x\mapsto\alpha x}(\mathsf{t}_\ell, \mathsf{t}_r, \mathsf{t}'_\ell, \mathsf{t}'_r) \triangleq \; & \big((\mathsf{t}_\ell[1] = x \wedge \mathsf{t}_r[1] = \alpha) \vee (\mathsf{t}_\ell[1] = \alpha \\ & \wedge \mathsf{t}_r[1] = x) \big) \\ & \wedge \psi'^{\leq n}_{x\mapsto\alpha x}(\mathsf{t}_\ell, \mathsf{t}_r, \mathsf{t}'_\ell, \mathsf{t}'_r)\end{aligned} \tag{18}$$

The constructed formulae $\varphi^{\leq n}_{x\mapsto\epsilon}$ and $\varphi^{\leq n}_{x\mapsto\alpha x}$ describe regular relations with arity 4 over the alphabet $\Sigma_{\mathbb{X},\square}$. Since they are regular (i.e., length-preserving), they can also be interpreted as binary relations over the composed alphabet $\Sigma^2_{\mathbb{X},\square}$. This interpretation can easily be done by modifying a length-preserving transducer corresponding to $\varphi^{\leq n}_{x\mapsto\epsilon}$ and $\varphi^{\leq n}_{x\mapsto\alpha x}$ respectively in a way that each transition $q \text{-}\!\!\upharpoonright\!\!\{a_1,a_2,a_3,a_4\}\!\!\upharpoondown\!\!\text{-} r$ is replaced by the transition $q \text{-}\!\!\upharpoonright\!\!\{\left[\begin{smallmatrix}a_1\\a_2\end{smallmatrix}\right],\left[\begin{smallmatrix}a_3\\a_4\end{smallmatrix}\right]\}\!\!\upharpoondown\!\!\text{-} r$ (with the same sets of states). When interpreted as binary relations in this way, they denote the relations containing pairs (without loss of generality, we show this only for $\varphi^{\leq n}_{x\mapsto\epsilon}$)

$$\left(\begin{bmatrix} u_1 & \cdots & u_m & u_{m+1} & \cdots & u_n \\ v_1 & \cdots & v_m & \square & \cdots & \square \end{bmatrix} \begin{bmatrix}\square\\\square\end{bmatrix}^i , \begin{bmatrix} w_1 & \cdots & w_k & w_{k+1} & \cdots & w_\ell \\ z_1 & \cdots & z_k & \square & \cdots & \square \end{bmatrix} \begin{bmatrix}\square\\\square\end{bmatrix}^j \right)$$

such that

- $\tau^{\leq n}_{x\mapsto\epsilon}((u_1 \ldots u_n, v_1 \ldots v_m)) = (w_1 \ldots w_\ell, z_1 \ldots z_k)$ for $\varphi^{\leq n}_{x\mapsto\epsilon}$,
- $\tau^{\leq n}_{x\mapsto\alpha x}((u_1 \ldots u_n, v_1 \ldots v_m)) = (w_1 \ldots w_\ell, z_1 \ldots z_k)$ for $\varphi^{\leq n}_{x\mapsto\alpha x}$, and
- $\max(m, n) + i = \max(k, \ell) + j$.

Let us now consider $\varphi^{\leq n}_{x\mapsto\epsilon}$. We note that for every $((u, v), (w, z)) \in \tau^{\leq n}_{x\mapsto\epsilon}$, there will indeed be a corresponding pair in $\varphi^{\leq n}_{x\mapsto\epsilon}$ (actually, there will be infinitely many such pairs that differ in the number of used padding symbols).

In order to get closer to $\tau_{x\rightarrowtail\epsilon}$, we need to modify the relation of $\varphi^{\leq n}_{x\mapsto\epsilon}$ to also perform trimming of the shared prefix. We do this modification by taking the (length-preserving) two-track transducer $T^{\leq n}_{x\mapsto\epsilon}$ that recognizes $\varphi^{\leq n}_{x\mapsto\epsilon}$ (it can be constructed due to Proposition 2). Moreover, we also create a (length non-preserving) transducer T_{trim} that performs trimming of its input; this is easy to implement by a two-state transducer that replaces a prefix of symbols of the form $\left[\begin{smallmatrix}\beta\\\beta\end{smallmatrix}\right]$ with ϵ, for $\beta \in \Sigma_{\mathbb{X},\square}$. By composing the two transducers, we obtain $T^{\leq n}_{x\rightarrowtail\epsilon} = T_{trim} \circ T^{\leq n}_{x\mapsto\epsilon}$. An example of a transducer realizing $T^{\leq 1}_{x\mapsto\epsilon}$ is shown in Fig. 8.

We can repeat the previous reasoning for $\varphi^{\leq n}_{x\mapsto\alpha x}$ in a similar way to obtain the (length non-preserving) transducer $T^{\leq n}_{x\rightarrowtail\alpha x}$.

Algorithm 1: Solving a string constraint φ using RMC

Input: Encoding $\mathcal{I}$ of a formula φ (the initial set), transformers $\mathcal{T}_{x\rightarrowtail\alpha x}$, $\mathcal{T}_{x\rightarrowtail\epsilon}$, and the destination set $\mathcal{D}$
Output: A model of φ if φ is satisfiable, false otherwise

```
reach_0 := I;
processed := ∅;
T := T_{x↣αx} ∪ T_{x↣ε};
i := 0;
while reach_i ⊈ processed do
    if D ∩ reach_i ≠ ∅ then
        return
          ExtractModel({T}_{j∈{0,...,i}}, D, reach_0, ..., reach_i);
    processed := processed ∪ reach_i;
    reach_{i+1} := satur ∘ T(reach_i);      // satur• ∘ T(reach_i)
    i++;
return false;
```

Lemma 5. *It holds that* $\tau^{\leq n}_{x\rightarrowtail\epsilon}((u_1 \ldots u_n, v_1 \ldots v_m)) = (w_1 \ldots w_\ell, z_1 \ldots z_k)$ *iff*

$$\left(\begin{bmatrix} u_1 & \cdots & u_m & u_{m+1} & \cdots & u_n \\ v_1 & \cdots & v_m & \square & \cdots & \square \end{bmatrix} \begin{bmatrix}\square\\\square\end{bmatrix}^i , \begin{bmatrix} w_1 & \cdots & w_k & w_{k+1} & \cdots & w_\ell \\ z_1 & \cdots & z_k & \square & \cdots & \square \end{bmatrix} \begin{bmatrix}\square\\\square\end{bmatrix}^j \right) \in \mathcal{L}(T^{\leq n}_{x\rightarrowtail\epsilon})$$

for some $i, j \in \mathbb{N}$.

Further, it holds that $\tau^{\leq n}_{x\rightarrowtail\alpha x}((u_1 \ldots u_n, v_1 \ldots v_m)) = (w_1 \ldots w_\ell, z_1 \ldots z_k)$ *iff*

$$\left(\begin{bmatrix} u_1 & \cdots & u_m & u_{m+1} & \cdots & u_n \\ v_1 & \cdots & v_m & \square & \cdots & \square \end{bmatrix} \begin{bmatrix}\square\\\square\end{bmatrix}^i , \begin{bmatrix} w_1 & \cdots & w_k & w_{k+1} & \cdots & w_\ell \\ z_1 & \cdots & z_k & \square & \cdots & \square \end{bmatrix} \begin{bmatrix}\square\\\square\end{bmatrix}^j \right) \in \mathcal{L}(T^{\leq n}_{x\rightarrowtail\alpha x})$$

for some $i, j \in \mathbb{N}$.

Proof. The proof follows from the above described construction of transducers $T^{\leq n}_{x\rightarrowtail\alpha x}$ and $T^{\leq n}_{x\rightarrowtail\epsilon}$. □

3.4.2. RMC for quadratic equations

In Algorithm 1, we give a high-level algorithm for solving string constraints using RMC. The algorithm is parameterized by the following: (i) a regular language $\mathcal{I}$ encoding a formula φ (the initial set), (ii) rational relations given by the transducers $\mathcal{T}_{x\rightarrowtail\alpha x}$ and $\mathcal{T}_{x\rightarrowtail\epsilon}$, and (iii) the destination set $\mathcal{D}$ (also given as a regular language). The algorithm tries to solve the RMC problem $(\mathcal{I}, \mathcal{T}_{x\rightarrowtail\alpha x} \cup \mathcal{T}_{x\rightarrowtail\epsilon}, \mathcal{D})$ by an iterative unfolding of the

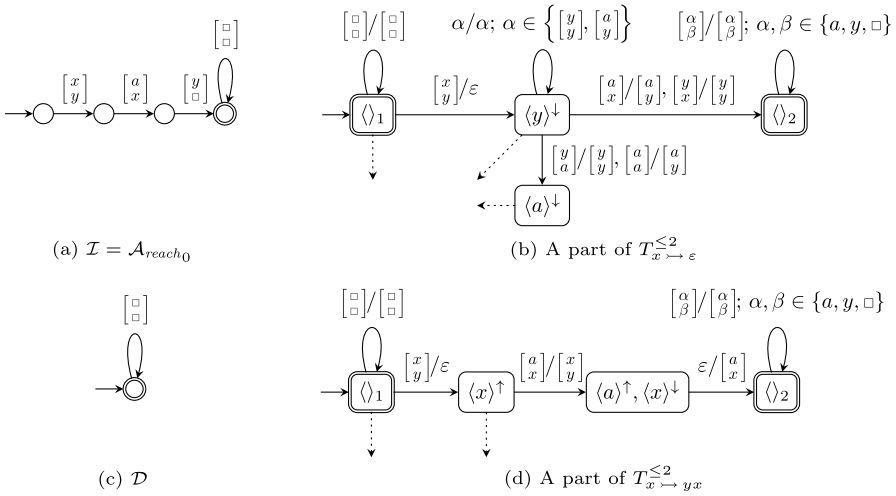


Fig. 9. Input of Algorithm 1 for solving the word equation $xay = yx$. The size of $\mathcal{T}^{eq}_{x \rightarrowtail \epsilon}$ and $\mathcal{T}^{eq}_{x \rightarrowtail yx}$ would be prohibitively large, so we only give relevant parts of transducers $T^{\leq 2}_{x \rightarrowtail \epsilon}$ and $T^{\leq 2}_{x \rightarrowtail yx}$. Some transitions use shorthand notation with the obvious meaning.

Function `ExtractModel`$(\{\mathcal{T}_j\}_{j\in\{0,\ldots,i\}}, \mathcal{D}, reach_0, \ldots, reach_i)$

1 $I(a) := a$ for each $a \in \Sigma$, $I(x) := \epsilon$ for each $x \in \mathbb{X}$;
2 let $w_i \in \mathcal{D} \cap reach_i$;
3 **for** $\ell = i$ **downto** 1 **do**
4 | let $w_{\ell-1} \in \mathcal{T}_\ell^{-1}(w_\ell) \cap reach_{\ell-1}$;
5 | let ρ be a rule s.t. $w_\ell \in w_{\ell-1}[\rho]$;
6 | **if** $\rho = y \rightarrowtail \alpha y$ **then**
7 | | $I(y) := I(\alpha).I(y)$;
8 **return** I;

transition relation $\mathcal{T}$ computed in Line 3, looking for an element w_i from $\mathcal{D}$. If such an element is found in the set $reach_i$, we call Function ExtractModel to extract a model of the original word equation by starting a backward run from w_i, computing pre-images $w_{i-1}, \ldots, w_1$ over transformers $\mathcal{T}_{x \rightarrowtail \alpha x}$ and $\mathcal{T}_{x \rightarrowtail \epsilon}$ (restricting them to $reach_j$ for every w_j), while updating values of the variables according to the transformations that were performed. Note that *ExtractModel* uses a more general interface allowing to specify a transducer for each backward step (Line 4 of Function ExtractModel). This is utilized later in Section 4; here, we just pass i copies of $\mathcal{T}$. Algorithm 1 also employs *saturation* of the sets of reachable configurations defined as:

$$satur(L) = \left\{ u \;\middle|\; w \in L, w \in u.\begin{bmatrix}\square\\ \square\end{bmatrix}^* \right\}. \tag{19}$$

Intuitively, $satur(L)$ removes some occurrences (possibly none of them) of the padding symbol at the end of all words from L. If L is a regular language, $satur(L)$ is regular as well (from an FA representing L we can get $satur(L)$ by saturating its transitions over the padding symbol). We saturate the sets of reachable configurations, because we want to keep the shortest words (i.e., words without padding symbols)—e.g., the transformer $\mathcal{T}^{eq}_{x \rightarrowtail \epsilon}$ need not generate all shortest words.

Algorithm 1 follows a *breadth-first search* (BFS) strategy: from the initial set $\mathcal{I}$, we apply both transformers $\mathcal{T}_{x \rightarrowtail \alpha x}$ and $\mathcal{T}_{x \rightarrowtail \epsilon}$ on all elements of $\mathcal{I}$ at the same time, before repeatedly applying the transformers on the result. This corresponds to a breadth-first application of the transformers if we applied them one element of $\mathcal{I}$ at a time.

Our first instantiation of the algorithm is for checking satisfiability of a single quadratic word equation $eq\colon \mathsf{t}_\ell = \mathsf{t}_r$. We instantiate the RMC problem as the tuple $(\mathcal{I}^{eq}, \mathcal{T}^{eq}_{x \rightarrowtail \alpha x} \cup \mathcal{T}^{eq}_{x \rightarrowtail \epsilon}, \mathcal{D}^{eq})$ where

$$\mathcal{I}^{eq} = \mathsf{encode}(\mathsf{t}_\ell, \mathsf{t}_r) \qquad \mathcal{T}^{eq}_{x \rightarrowtail \alpha x} = \bigcup_{y \in \mathbb{X}, \alpha \in \Sigma_{\mathbb{X}}} T^{\leq 2}_{y \rightarrowtail \alpha y} \qquad \mathcal{D}^{eq} = \left\{\begin{bmatrix}\square\\ \square\end{bmatrix}\right\}^*$$
$$\mathcal{T}^{eq}_{x \rightarrowtail \epsilon} = \bigcup_{y \in \mathbb{X}} T^{\leq 2}_{y \rightarrowtail \epsilon}$$

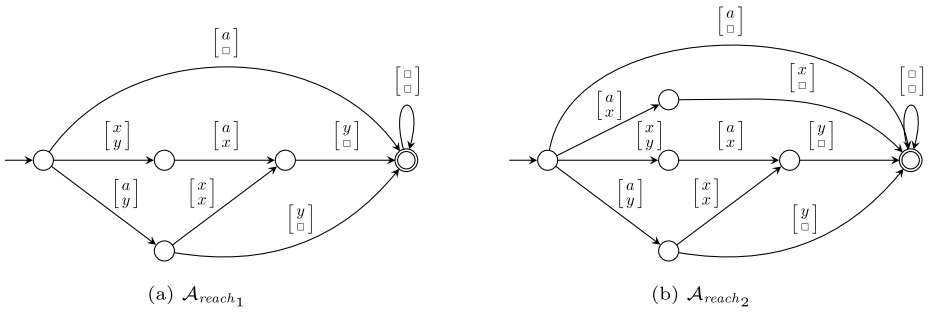
Lemma 6. *Algorithm 1 instantiated with* $(\mathcal{I}^{eq}, \mathcal{T}^{eq}_{x \rightarrowtail \alpha x} \cup \mathcal{T}^{eq}_{x \rightarrowtail \epsilon}, \mathcal{D}^{eq})$ *is sound, complete, and terminating if* $eq\colon \mathsf{t}_\ell = \mathsf{t}_r$ *is quadratic.*

Proof. We encode the nodes of a Nielsen proof graph as strings. The initial node $\mathsf{t}_\ell = \mathsf{t}_r$ corresponds to a string from $\mathcal{I}^{eq}$. Due to the padding, the final node $\epsilon = \epsilon$ corresponds to a string from $\mathcal{D}^{eq}$. The relations $\mathcal{T}^{eq}_{x \rightarrowtail \alpha x}$ and $\mathcal{T}^{eq}_{x \rightarrowtail \epsilon}$ implement Nielsen rules $x \rightarrowtail \alpha x$ and $x \rightarrowtail \epsilon$. From Lemma 3 we have that the Nielsen proof graph is finite. Since our approach implements the *breadth-first search* (BFS) strategy, it is sound, complete, and terminating (see Fig. 10). □

Example 6. Consider the word equation $eq_1\colon xay = yx$ from Section 3.1. We apply Algorithm 1 on the encoded equation $\mathcal{I} = \mathsf{encode}(xay, yx) = \begin{bmatrix} x & a & y\\ y & x & \square\end{bmatrix}\begin{bmatrix}\square\\ \square\end{bmatrix}^*$. The inputs of the algorithm are in Fig. 9. In the first iteration of the main loop, the regular set *processed*, represented by its minimal FA $\mathcal{A}_{reach_1}$, is given in Fig. 10(a) (the FA also corresponds to $reach_1$ and represents the set S_1 from Example 4). In particular, consider, e.g.,

$$\left(\begin{bmatrix} x & a & y\\ y & x & \square\end{bmatrix}, \begin{bmatrix} a & y & \square\\ y & \square & \square\end{bmatrix}\right) \in \mathcal{T}^{eq}_{x \rightarrowtail \epsilon}.$$

After saturation we get that $\begin{bmatrix} a & y\\ y & \square\end{bmatrix} \in reach_1$. In the second (and also the last) iteration, the set *processed* is given by the minimal FA $\mathcal{A}_{reach_2}$ in Fig. 10(b) (which also corresponds to $reach_2$ and $reach_3$, and the set S_2 from Example 4). Since $reach_3 \subseteq processed$, the algorithm terminates with `false`, establishing the unsatisfiability of eq_1. □



(a) $\mathcal{A}_{reach_1}$ (b) $\mathcal{A}_{reach_2}$

Fig. 10. Examples of finite automata $\mathcal{A}_{reach_1}$, $\mathcal{A}_{reach_2}$ representing sets of configurations reachable in one and two steps, respectively, when solving the word equation $xay = yx$ using Algorithm 1.

4. Solving a system of word equations using RMC

In the previous section we described how to solve a single quadratic word equation in the RMC framework. In this section we focus on an extension of this approach to handle a system of word equations of the form

$$\Phi : \mathsf{t}_\ell^1 = \mathsf{t}_r^1 \wedge \mathsf{t}_\ell^2 = \mathsf{t}_r^2 \wedge \cdots \wedge \mathsf{t}_\ell^n = \mathsf{t}_r^n. \tag{20}$$

In the first step we need to encode the system Φ as a regular language. For this we extend the encode function to a system of word equations by defining

$$\mathsf{encode}(\Phi) = \mathsf{encode}(\mathsf{t}_\ell^1, \mathsf{t}_r^1).\left\{\begin{bmatrix}\#\\\#\end{bmatrix}\right\}.\cdots.\left\{\begin{bmatrix}\#\\\#\end{bmatrix}\right\}.\mathsf{encode}(\mathsf{t}_\ell^n, \mathsf{t}_r^n), \tag{21}$$

where # is a delimiter symbol, $\# \notin \Sigma_{\mathbb{X},\square}$. From Lemma 4 we know that $\mathsf{encode}(\mathsf{t}_\ell^i, \mathsf{t}_r^i)$ is regular for all $1 \leq i \leq n$. Moreover, since regular languages are closed under concatenation (Proposition 1), the set $\mathsf{encode}(\Phi)$ is also regular. Because each equation is now separated by a delimiter, we need to extend the destination set to $\left\{\begin{bmatrix}\square\\\square\end{bmatrix}, \begin{bmatrix}\#\\\#\end{bmatrix}\right\}^*$.

For the transition relation, we need to extend $\tau_{x\rightarrowtail\alpha x}^{\leq i}$ and $\tau_{x\rightarrowtail\epsilon}^{\leq i}$ from the previous section to support delimiters. An application of a rule $x \rightarrowtail \alpha x$ on a system of equations can be described as follows: the rule $x \rightarrowtail \alpha x$ is applied to the first non-empty equation and the rest of the equations are modified according to the substitution $x \mapsto \alpha x$. The substitution on the other equations is performed regardless of their first symbols. The procedure is analogous for the rule $x \rightarrowtail \epsilon$. A series of applications of the rules can reduce the number of equations, which then leads to a string in our encoding with a prefix from $\left\{\begin{bmatrix}\square\\\square\end{bmatrix}, \begin{bmatrix}\#\\\#\end{bmatrix}\right\}^*$. The relation implementing $x \rightarrowtail \alpha x$ or $x \rightarrowtail \epsilon$ on an encoded system of equations skips this prefix. Formally, the rule $x \rightarrowtail \alpha x$ for a system of equations where every equation has at most i occurrences of every variable is given by the following relation:

$$T_{x\rightarrowtail\alpha x}^{\wedge eq,i} = T_{skip}.T_{x\rightarrowtail\alpha x}^{\leq i}.\left(\left\{\begin{bmatrix}\#\\\#\end{bmatrix} \mapsto \begin{bmatrix}\#\\\#\end{bmatrix}\right\}.(T_{trim} \circ S_{x\mapsto\alpha x}^{\leq i})\right)^*, \tag{22}$$

where $T_{skip} = \left\{\begin{bmatrix}\square\\\square\end{bmatrix} \mapsto \begin{bmatrix}\square\\\square\end{bmatrix}, \begin{bmatrix}\#\\\#\end{bmatrix} \mapsto \begin{bmatrix}\#\\\#\end{bmatrix}\right\}^*$ and $S_{x\mapsto\alpha x}^{\leq i}$ is the binary transducer representing the formula $\psi_{x\mapsto\alpha x}^{\prime\leq i}$ from Section 3.4.1 (which does not look at the first symbol on the tape). The relation $T_{x\rightarrowtail\epsilon}^{\wedge eq,i}$ is defined in a similar manner. By construction and from the closure properties of rational relations (cf. Proposition 1), it is clear that $T_{x\rightarrowtail\alpha x}^{\wedge eq,i}$ and $T_{x\rightarrowtail\epsilon}^{\wedge eq,i}$ are rational.

4.1. Quadratic case

When Φ is quadratic, its satisfiability problem can be reduced to an RMC problem $(\mathcal{I}_\Phi^{\wedge eq}, \mathcal{T}_{x\rightarrowtail\alpha x}^{\wedge eq} \cup \mathcal{T}_{x\rightarrowtail\epsilon}^{\wedge eq}, \mathcal{D}^{\wedge eq})$ instantiating Algorithm 1 where

$$\mathcal{I}_\Phi^{\wedge eq} = \mathsf{encode}(\Phi) \qquad \mathcal{T}_{x\rightarrowtail\alpha x}^{\wedge eq} = \bigcup_{y\in\mathbb{X},\alpha\in\Sigma_{\mathbb{X}}} T_{y\rightarrowtail\alpha y}^{\wedge eq,2} \qquad \mathcal{D}^{\wedge eq} = \left\{\begin{bmatrix}\square\\\square\end{bmatrix}, \begin{bmatrix}\#\\\#\end{bmatrix}\right\}^*$$
$$\mathcal{T}_{x\rightarrowtail\epsilon}^{\wedge eq} = \bigcup_{y\in\mathbb{X}} T_{y\rightarrowtail\epsilon}^{\wedge eq,2}$$

Rationality of $\mathcal{T}_{x\rightarrowtail\alpha x}^{\wedge eq}$ and $\mathcal{T}_{x\rightarrowtail\epsilon}^{\wedge eq}$ follows directly from Proposition 1. In addition, we also need to modify Algorithm 1 such that in Line 9, we substitute *satur* with *satur*$^\bullet$ defined as follows:

$$satur^\bullet(L) = \left\{u_1.u_2 \dots u_k \mid w \in L, w \in u_1.\begin{bmatrix}\square\\\square\end{bmatrix}^+.u_2.\begin{bmatrix}\square\\\square\end{bmatrix}^+ \cdots u_k \text{ for some } k \in \mathbb{N}\right\}.$$

Intuitively, *satur*$^\bullet(L)$ modifies *satur* to take into account the fact that we now work with several word equations encoded into a single word, where they are separated by delimiters. Now, *satur*$^\bullet$ does not only remove paddings at the end of the word (when $k = 1$), but also in the middle.

The soundness and completeness of our procedure for a system of quadratic word equations is summarized by the following lemma.

Lemma 7. *Algorithm 1 instantiated with* $(\mathcal{I}_\Phi^{\wedge eq}, \mathcal{T}_{x\rightarrowtail\alpha x}^{\wedge eq} \cup \mathcal{T}_{x\rightarrowtail\epsilon}^{\wedge eq}, \mathcal{D}^{\wedge eq})$ *is sound, complete, and terminating if* Φ *is quadratic.*

Proof. We encode the nodes of a Nielsen proof graph as strings. The initial node Φ corresponds to a string from $\mathcal{I}_\Phi^{\wedge eq}$ (conjunction can be seen as the delimiter #). Because of the padding, the final node $\epsilon = \epsilon \wedge \cdots \wedge \epsilon = \epsilon$ corresponds to a string from $\mathcal{D}^{\wedge eq}$. The relations $\mathcal{T}_{x\rightarrowtail\alpha x}^{\wedge eq}$ and $\mathcal{T}_{x\rightarrowtail\epsilon}^{\wedge eq}$ implement the Nielsen rules $x \rightarrowtail \alpha x$ and $x \rightarrowtail \epsilon$. From Lemma 3 we have that the Nielsen proof graph is finite (and hence the potential final node is in a finite depth). Since our approach implements the BFS strategy, it is both sound, complete, and terminating. □

4.2. General case

Let us now consider the general case when the system Φ is not quadratic. In this section, we show that this general case is also reducible to an extended version of RMC.

We first apply Algorithm 2 to a general system of string constraints Φ in order to get an equisatisfiable cubic system of word equations Φ'. If the input of the transformation is a system of equations with n symbols, then the output of the transformation will, in the worst case, contain $\frac{n}{2}$ additional word equations and $\frac{n}{2}$ additional literals, so the transformation is linear. Then, we

Algorithm 2: Transformation to a cubic system of equations

Input: System of word equations Φ
Output: Equisatisfiable cubic system of word equations Ψ

```
1 Ψ := Φ;
2 while ∃x ∈ 𝕏 s.t. x occurs more than three times in Ψ do
3 |   Replace two occurrences of x in Φ by a fresh word
  |     variable x′ to obtain a new system Ψ′;
4 |   Ψ := (Ψ′ ∧ x = x′);
5 return Ψ;
```

can use the transition relations $T^{\wedge eq,3}_{x\mapsto\alpha x}$ and $T^{\wedge eq,3}_{x\mapsto\epsilon}$ to construct transformations of the encoded system Φ'.

Lemma 8. *Any system of word equations can be transformed by Algorithm 2 to an equisatisfiable cubic system of word equations.*

Proof. Let Φ be the input system of word equations. Observe that in every iteration of Algorithm 2, the number of occurrences of a variable x is decreased by one and a new variable x' with exactly three occurrences is introduced. □

One more issue we need to solve is to make sure that we work with a cubic system of word equations in every step of our algorithm. It may happen that a transformation of the type $x \mapsto yx$ increases the number of occurrences of the variable y by one, so if there had already been three occurrence of y before the transformation, the result will not be cubic any more. More specifically, assume a cubic system of word equations $x.\mathrm{t}_\ell = y.\mathrm{t}_r \wedge \Phi$, where x and y are distinct word variables and t_ℓ and t_r are word terms. If we apply the transformation $x \mapsto yx$, we will obtain $x(\mathrm{t}_\ell[x \mapsto yx]) = \mathrm{t}_r[x \mapsto yx] \wedge \Phi[x \mapsto yx]$. Observe that (i) the number of occurrences of y is first *reduced by one* because the first y on the right-hand side of $x.\mathrm{t}_\ell = y.\mathrm{t}_r$ is removed and (ii) the number of occurrences of y can be at most *increased by two* because there exist at most two occurrences of x in t_ℓ, t_r, and Φ. Therefore, after the transformation $x \mapsto yx$, a cubic system of word equations might become a *(y-)quartic system of word equations* (at most four occurrences of the variable y and at most three occurrences of any other variable). For this reason, we need to apply the conversion to the cubic system after each transformation.

Given a fresh variable v, we use $\mathcal{C}_v$ to denote the transformation from a single-quartic system of word equations to a cubic system of equations using the fresh variable v.

Lemma 9. *The relation $T_{\mathcal{C}_v}$ performing the transformation $\mathcal{C}_v$ on an encoded single-quartic system of equations is rational.*

Proof. We show how we can create a transducer for the transformation from a single-quartic system of word equations to a cubic system of word equations.

In the first step, we create the transducer $\mathcal{T}^{sq}_{x,x_i}$ that accepts only input that is an encoding of a x-quartic system of word equations. This can be done by using states to trace the number of occurrences of variables (we only need to count up to four). For an encoding of a x-quartic system of word equations, the transducer $\mathcal{T}^{sq}_{x,x_i}$ returns an encoding that is obtained by replacing the first two occurrences of x from the input to x_i and concatenating the language $\left[\begin{smallmatrix}\#\\\#\end{smallmatrix}\right]\left[\begin{smallmatrix}x\\x_i\end{smallmatrix}\right]\left[\begin{smallmatrix}\square\\\square\end{smallmatrix}\right]^*$ at the end.

In the second step, we create the transducer $\mathcal{T}_{cub}$ that accepts only encodings of a cubic system of word equations and returns the same encodings.

Algorithm 3: Solving a general string constraint φ using RMC

Input: Encoding $\mathcal{I}$ of a formula φ (the initial set),
ordered set of indices $\mathbb{V} = \{v_1, v_2, \dots\}$,
parameterized transformers $\mathcal{T}^v_{x\mapsto\alpha x}$, $\mathcal{T}^v_{x\mapsto\epsilon}$ where $v \in \mathbb{V}$, and
the destination set $\mathcal{D}$
Output: A model of φ if φ is satisfiable, false otherwise

```
1  reach_0 := I;
2  processed := ∅;
3  T^v := T^v_{x↦αx} ∪ T^v_{x↦ε};
4  i := 0;
5  while reach_i ⊈ processed do
6  |   if D ∩ reach_i ≠ ∅ then
7  |   |   return
   |   |     ExtractModel({T^{v_j}}_{j∈{0,...,i}}, D, reach_0, ..., reach_i);
8  |   processed := processed ∪ reach_i;
9  |   reach_{i+1} := satur• ∘ T^{v_i}(reach_i);
10 |   𝕏 := 𝕏 ∪ {v_i};
11 |   i++;
12 return false;
```

Now we have

$$T_{\mathcal{C}_v} = \mathcal{L}(\mathcal{T}_{cub}) \cup \bigcup_{x\in\mathbb{X}} \mathcal{L}(\mathcal{T}^{sq}_{x,v}). \tag{23}$$

The lemma then follows by Proposition 1. □

To express solving a system of string constraints Φ in the terms of a (modified) RMC, we first convert Φ (using Algorithm 2) to an equisatisfiable cubic system Φ'. The satisfiability of a system of word equations Φ can be reduced to a modified RMC problem $(\mathcal{I}^{\wedge eq}_\Phi, \mathcal{T}^{v_i,\wedge eq}_{x\mapsto\alpha x} \cup \mathcal{T}^{v_i,\wedge eq}_{x\mapsto\epsilon}, \mathcal{D}^{\wedge eq})$ instantiating Algorithm 1 with the following components:

$$\mathcal{I}^{\wedge eq}_\Phi = \mathsf{encode}(\Phi') \qquad \mathcal{T}^{v,\wedge eq}_{x\mapsto\alpha x} = T_{\mathcal{C}_v} \circ \bigcup_{y\in\mathbb{X},\alpha\in\Sigma_\mathbb{X}} T^{\wedge eq,3}_{y\mapsto\alpha y} \qquad \mathcal{D}^{\wedge eq} = \left\{\left[\begin{smallmatrix}\square\\\square\end{smallmatrix}\right], \left[\begin{smallmatrix}\#\\\#\end{smallmatrix}\right]\right\}^*$$
$$\mathcal{T}^{v,\wedge eq}_{x\mapsto\epsilon} = T_{\mathcal{C}_v} \circ \bigcup_{y\in\mathbb{X}} T^{\wedge eq,3}_{y\mapsto\epsilon}$$

For the modified RMC algorithm, we need to assume that $\mathbb{V} = \{v_1, v_2, \dots\} \cap \Sigma_\mathbb{X} = \emptyset$, where $\mathbb{V}$ is a set of *fresh index variables*. We also need to update Line 3 of Algorithm 1 to $\mathcal{T}^v := \mathcal{T}^v_{x\mapsto\alpha x} \cup \mathcal{T}^v_{x\mapsto\epsilon}$ and Line 9 to $reach_{i+1} := \mathcal{T}^{v_i}(reach_i)$; $\mathbb{X} := \mathbb{X} \cup \{v_i\}$; to allow using a new variable v_i in every iteration (here, in every iteration of the algorithm, $\mathcal{T}^v$ will be instantiated with a new value of the v parameter). The entire algorithm is shown in Algorithm 3. Rationality of $\mathcal{T}^{v,\wedge eq}_{x\mapsto\alpha x}$ and $\mathcal{T}^{v,\wedge eq}_{x\mapsto\epsilon}$ follows directly from Proposition 1.

Lemma 10. *Algorithm 3 instantiated with $(\mathcal{I}^{\wedge eq}_\Phi, \mathbb{V}, \mathcal{T}^{v,\wedge eq}_{x\mapsto\alpha x} \cup \mathcal{T}^{v,\wedge eq}_{x\mapsto\epsilon}, \mathcal{D}^{\wedge eq})$ is sound if Φ is cubic.*

Proof. We again encode the nodes of a Nielsen proof graph as strings. The initial and final node correspond to strings from the encoded initial $\mathcal{I}^{\wedge eq}_\Phi$ and final language $\mathcal{D}^{\wedge eq}$, respectively. The relations $\mathcal{T}^{v,\wedge eq}_{x\mapsto\alpha x}$ and $\mathcal{T}^{v,\wedge eq}_{x\mapsto\epsilon}$ implement the Nielsen rules $x \mapsto \alpha x$ and $x \mapsto \epsilon$. For an arbitrary system of word equations the Nielsen proof graph may be infinite. However, since the transformation $\mathcal{C}_v$ preserves satisfiability, the procedure is sound. □

Completeness. Since the previous approach can in each step introduce a new equation (due to the transducer $T_{\mathcal{C}_v}$ transforming the system of equations to a cubic one), completeness is not guaranteed in general. In order to get a sound and complete procedure

for cubic equations, it is necessary to use transducers with an increasing number of symbols being rewritten. In particular, we need to use transducers implementing the following relations:

$$\mathcal{T}^{i,\wedge eq\star}_{x\mapsto\alpha x} = \bigcup_{y\in\mathbb{X},\alpha\in\Sigma_{\mathbb{X}}} T^{\wedge eq,2^{i+1}}_{y\mapsto\alpha y} \qquad \mathcal{T}^{i,\wedge eq\star}_{x\mapsto\epsilon} = \bigcup_{y\in\mathbb{X}} T^{\wedge eq,2^{i+1}}_{y\mapsto\epsilon}$$

with $i \in \mathbb{N}_1$ being the number of iteration (moreover, it is also necessary to skip Line 10 in Algorithm 3). After each step, the maximum number of occurrences of a variable in the system is in the worst case multiplied by two. Therefore, in the ith step, the number of occurrence of a variable in the system can be up to 2^{i+1}, which can be handled by $\mathcal{T}^{i,\wedge eq\star}_{x\mapsto\alpha x}$ and $\mathcal{T}^{i,\wedge eq\star}_{x\mapsto\epsilon}$.

Lemma 11. *Algorithm 3 instantiated with* $(\mathcal{I}^{\wedge eq}_{\Phi}, \mathbb{N}, \mathcal{T}^{i,\wedge eq\star}_{x\mapsto\alpha x} \cup \mathcal{T}^{i,\wedge eq\star}_{x\mapsto\epsilon}, \mathcal{D}^{\wedge eq})$ *and modified as described above is sound and complete if* Φ *is cubic.*

Proof. As in previous cases, we encode the nodes of a Nielsen proof graph as strings. Since the transducer $T^{\wedge eq,\ell}_{y\mapsto\alpha y}$ correctly implements the Nielsen rule $y \mapsto \alpha y$ for systems where each variable occurs at most ℓ times, it suffices to show that in the ith iteration in Algorithm 3, the number of occurrences of each variable is bounded by 2^{i+1}. The soundness and completeness then follows from the BFS strategy of Algorithm 3 and Lemma 3.

Consider a system of equations Φ s.t. each variable occurs at most n times. Application of the rule of the form $y \mapsto \epsilon$ does not increase the number of occurrences of y. The situation is different for a rule of the form $y \mapsto \alpha y$ s.t. $\alpha \in \mathbb{X}$. Furthermore, observe that there are at most n occurrences of y and $(n-1)+(n-1) = 2n-2$ occurrences of α in the modified system (each y is replaced by αy and the first occurrence of α is removed from the modified system). The maximum number of occurrences of variables in the ith iteration of Algorithm 3 is hence bounded by 2^{i+1} (note that this is a loose upper bound, since the number of occurrences increases to $2n-2 \leq 2n$ in the worst case), provided that Φ is cubic. □

Termination. Since the Nielsen transformation does not guarantee termination for the general case, neither does our algorithm. Investigation of possible symbolic encodings of complete algorithms, e.g. Makanin's algorithm (Makanin, 1977), is our future work.

5. Handling Boolean combination of string constraints

In this section, we will extend the procedure from handling a *conjunction* of word equations into a procedure that handles their arbitrary Boolean combination. An obvious approach is by combining the solutions we have given in Sections 3 and 4 with standard DPLL(T)-based solvers and use our procedure to handle the string theory. We can, however, solve the whole formula with our procedure by using the encoding proposed in this section. Although we do not have hope that the presented solution can compete with the highly-optimized DPLL(T)-based solvers, it (also taking into account the extensions from Section 6) makes our framework more robust, by having a homogeneous automata-based encoding for a quite general class of constraints. When one, e.g., tries to extend the approach by using abstraction (cf. Bouajjani et al. (2012)) to accelerate termination or by *learning* the invariant in the spirit of Neider and Jansen (2013), they can then still treat the encoding of the whole system of constraints uniformly within the framework of RMC.

The negation of word equations can be handled in the standard way. For instance, we can use the approach given by Abdulla et al. (2014) to convert a negated word equation $\mathrm{t}_\ell \neq \mathrm{t}_r$ to the following string constraint:

$$\bigvee_{c\in\Sigma}(\mathrm{t}_\ell = \mathrm{t}_r \cdot cx \vee \mathrm{t}_\ell \cdot cx = \mathrm{t}_r) \vee \bigvee_{c_1,c_2\in\Sigma, c_1\neq c_2} (\mathrm{t}_\ell = yc_1x_1 \wedge \mathrm{t}_r = yc_2x_2) \tag{24}$$

The first part of the constraint says that either t_ℓ is a strict prefix of t_r or the other way around. The second part says that t_ℓ and t_r have a common prefix y and start to differ in the next characters c_1 and c_2. For word equations connected using $\wedge$ and $\vee$, we apply distributive laws to obtain an equivalent formula in the conjunctive normal form (CNF) whose size is at worst exponential in the size of the original formula. Note that we cannot use the Tseitin transformation (Tseitin, 1983), since it may introduce fresh negated variables and their removal using Eq. (24) would destroy the CNF form.

Let us now focus on how to express solving a string constraint Φ composed of an arbitrary Boolean combination of word equations using a (modified) RMC. We start by removing inequalities in Φ using Eq. (24), then we convert the system without inequalities into CNF, and, finally, we apply the procedure in Lemma 8 to convert the CNF formula to an equisatisfiable and cubic CNF Φ'. For deciding satisfiability of Φ' in the terms of RMC, both the transition relations and the destination set remain the same as in the previous section (general case). The only difference is the initial configuration because the system is not a conjunction of terms any more but rather a general formula in CNF. For this, we extend the definition of encode to a clause $c : (\mathrm{t}^1_\ell = \mathrm{t}^1_r \vee \ldots \vee \mathrm{t}^n_\ell = \mathrm{t}^n_r)$ as $\mathsf{encode}(c) = \bigcup_{1\leq j\leq n} \mathsf{encode}(\mathrm{t}^j_\ell, \mathrm{t}^j_r)$. Then the initial configuration for Φ' is given as

$$\mathcal{I}^{\wedge\vee eq}_{\Phi'} = \mathsf{encode}(c_1).\left\{\begin{bmatrix}\#\\\#\end{bmatrix}\right\}.\ldots.\left\{\begin{bmatrix}\#\\\#\end{bmatrix}\right\}.\mathsf{encode}(c_m), \tag{25}$$

where Φ' is of the form $\Phi' : c_1 \wedge \ldots \wedge c_m$ and each clause $c_i : (\mathrm{t}^1_\ell = \mathrm{t}^1_r \vee \ldots \vee \mathrm{t}^{n_i}_\ell = \mathrm{t}^{n_i}_r)$. We obtain the following lemma directly from Proposition 1.

Lemma 12. *The initial set* $\mathcal{I}^{\wedge\vee eq}_{\Phi'}$ *is regular.*

The transition relation and the destination set are the same as the ones in the previous section, i.e., $\mathcal{T}^{v,\wedge\vee eq}_{x\mapsto\alpha x} = \mathcal{T}^{v,\wedge eq}_{x\mapsto\alpha x}$, $\mathcal{T}^{v,\wedge\vee eq}_{x\mapsto\epsilon} = \mathcal{T}^{v,\wedge eq}_{x\mapsto\epsilon}$, and $\mathcal{D}^{\wedge\vee eq} = \mathcal{D}^{\wedge eq}$. The soundness of our procedure for a Boolean combination of word equations is summarized by the following lemma. The completeness can be achieved, as in Section 4.2, by transducers with an increasing number of rewritten symbols.

Lemma 13. *Given a Boolean combination of word equations* Φ*, Algorithm 3 instantiated with* $(\mathcal{I}^{\wedge\vee eq}_{\Phi'}, \mathbb{V}, \mathcal{T}^{v,\wedge\vee eq}_{x\mapsto\alpha x} \cup \mathcal{T}^{v,\wedge\vee eq}_{x\mapsto\epsilon}, \mathcal{D}^{\wedge\vee eq})$ *is sound.*

Proof. A system of full word equations can be converted according to the steps described above to an equisatisfiable system in CNF $\Psi : \bigwedge_{i=1}^n c_i$ where every c_i is a disjunction of equalities. Then, Ψ is satisfiable if there is some $\phi : \bigwedge_{i=1}^n \mathrm{t}^i_\ell = \mathrm{t}^i_r$ where $(\mathrm{t}^i_\ell = \mathrm{t}^i_r) \in c_i$ for all i. Moreover, we have $\mathsf{encode}(\phi) \in \mathcal{I}^{\wedge\vee eq}_{\Phi}$. From Lemma 10 (and from the BFS strategy of RMC), we get that our algorithm is sound in proving Φ is satisfiable. □

Completeness is guaranteed if we consider the transducers $\mathcal{T}^{i,\wedge eq\star}$ for Φ in the same way as in Section 4.2. Regarding termination, it cannot be guaranteed, due to the corresponding result in Section 4.2 that holds for a special case of the Boolean combination of string equations we consider here.

6. Extensions

In this section, we discuss how to extend our RMC-based framework to support the following two types of *atomic constraints*:

(i) A *length constraint* φ_i is a formula of Presburger arithmetic over the values of $|x|$ for $x \in \mathbb{X}$, where $|\cdot| : \mathbb{X} \to \mathbb{N}$ is the word length function (to simplify the notation we use a formula of Presburger arithmetic with free variables $\mathbb{X}$ and we keep in mind that the value assigned to $x \in \mathbb{X}$ corresponds in fact to $|x|$).

(ii) A *regular constraint* φ_r is a conjunction of atoms of the form $x \in \mathcal{L}(\mathcal{A})$ (or their negation) where x is a word variable and $\mathcal{A}$ is an FA representing a regular language.

6.1. Length constraints

In order to extend our framework to solve word equations with length constraints, we encode them as regular languages, and we encode the effect of Nielsen transformations on the lengths of variables as regular relations. Let us start with defining *atomic length constraints*:

$$\varphi_{len} ::= a_1 x_1 + \cdots + a_n x_n \leq c$$

for string variables $x_1, \ldots, x_n \in \mathbb{X}$ and integers $a_1, \ldots, a_n, c \in \mathbb{Z}$ (we will also use formulae in a less restricted form, which can always be translated to the defined one using standard arithmetic rules). Given a variable assignment $I : \mathbb{X} \to \Sigma^*$, it holds that I is a model of φ_{len}, written as $I \models \varphi_{len}$, iff $a_1 \cdot |I(x_1)| + \cdots + a_n \cdot |I(x_n)| \leq c$. We note that the satisfiability of a string constraint with only atomic length constraints connected via Boolean connectives (i.e., no word equations) corresponds to the satisfiability of a Boolean combination of constraints in *integer linear programming*, which is an NP-complete problem (Karp, 1972).

We will show that length constraints can be encoded into our framework using standard automata-based techniques for dealing with constraints in Presburger arithmetic (Presburger, 1929; Wolper and Boigelot, 2000, 2002). First, let us define how a first-order variable ranging over $\mathbb{N}$ is represented in MSO(Str). Let $LSBF : \mathbb{N} \to 2^{\mathbb{N}}$ be a function representing the *least-significant bit first* binary encoding of a number such that for $n \in \mathbb{N}$, we define $LSBF(n)$ to be the finite set $S \subseteq \mathbb{N}$ for which $n = \sum_{i \in S} 2^i$. For instance, $LSBF(42) = \{1, 3, 5\}$ because $42 = 2^1 + 2^3 + 2^5$. Moreover, we define the *positional* least-significant bit first binary encoding of a number $n \in \mathbb{N}$ as $LSBF_p(n) = \{\ell + 1 \mid \ell \in LSBF(n)\}$.

Proposition 14. *Let $\varphi_{len}(x_1, \ldots, x_n)$ be an atomic length constraint. Then there exists an* MSO(Str) *formula $\psi_{len}(X_1, \ldots, X_n)$ with free position variables $X_1, \ldots, X_n$ such that an assignment $\sigma : \{x_1, \ldots, x_n\} \to \mathbb{N}$ is a model of φ_{len} iff the assignment $\sigma' = \{X_i \mapsto LSBF_p(v_i) \mid \sigma(x_i) = v_i\}$ is a model of ψ_{len}.*

Proof. A possible encoding of φ_{len} into ψ_{len} is given, e.g., in Glenn and Gasarch (1996). □

Recall that in automata-based approaches to Presburger arithmetic (such as Glenn and Gasarch (1996), Wolper and Boigelot (2000) and Wolper and Boigelot (2002)), a formula φ with k free variables is translated into an automaton $\mathcal{A}_\varphi$ over the alphabet $\mathbb{B}^k$ for $\mathbb{B} = \{0, 1\}$. A model of φ is represented as a word $w \in (\mathbb{B}^k)^*$ in the language of $\mathcal{A}_\varphi$ such that projecting the track for variable x_i from w gives us the *LSBF* encoding of the value of x_i in the model. For instance, if $\begin{smallmatrix} x: \\ y: \end{smallmatrix}\begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \in \mathcal{L}(\mathcal{A}_\varphi)$, then the assignment $\{x \mapsto 9, y \mapsto 5\}$ is a model of φ because 1001 is a LSBF binary encoding of the number 9 and 1010 encodes the number 5.

In order to encode dealing with length constraints into our framework, Proposition 14 is not sufficient: we also need to be able to represent how the transformations modify those constraints and how the constraints restrict the space of possible solutions. In the following paragraphs, we provide the details about our approach.

Consider a word w_σ encoding an assignment $\sigma : \mathbb{X} \to \mathbb{N}$. The transformation $x \rightarrowtail yx$ for $x, y \in \mathbb{X}$ applied to w_σ produces a word $w_{\sigma'}$ encoding the assignment $\sigma' = \sigma \lhd \{x \mapsto \sigma(x) - \sigma(y)\}$ if $\sigma(x) \geq \sigma(y)$, where $\sigma' = \sigma \lhd \{x \mapsto n\}$ is defined as $\sigma'(x) = n$ and $\sigma'(y) = \sigma(y)$ for all $y \neq x$. The transformation $x \rightarrowtail ax$, for $a \in \Sigma$ produces a word $w_{\sigma'}$ encoding the assignment $\sigma' = \sigma \lhd \{x \mapsto \sigma(x) - 1\}$ if $\sigma(x) \geq 1$. Finally, the transformation $x \rightarrowtail \epsilon$ does not change the word w_σ, but imposes the restriction $\sigma(x) = 0$. Formally, the transformations are described using the following formulae:

$$\begin{aligned} &\varphi^{len}_{x \rightarrowtail yx}(x, y, x') \triangleq x \geq y \wedge x' = x - y, \\ &\varphi^{len}_{x \rightarrowtail ax}(x, x') \triangleq x \geq 1 \wedge x' = x - 1, \text{ and} \\ &\varphi^{len}_{x \rightarrowtail \epsilon}(x) \triangleq x = 0. \end{aligned} \tag{26}$$

From Propositions 14 and 2, it follows that the relations denoted by the formulae are regular. We will denote the transducers encoding those relations as $T^{len}_{x \rightarrowtail yx}$, $T^{len}_{x \rightarrowtail ax}$, and $T^{len}_{x \rightarrowtail \epsilon}$ respectively.

Let us now focus on how to adjust the initial and destination sets for an equation with a length constraint $\varphi_i(\mathbb{X})$ with free variables $\mathbb{X}$. The initial set is extended by all encoded models of φ_i. Formally, the part of the initial set related to the length constraint is given as $\mathcal{I}_{\varphi_i} = \mathcal{L}(\varphi_i)$ (which is a subset of $(\mathbb{B}^{|\mathbb{X}|})^*$) and the part of the destination set as $\mathcal{D}_{len} = (\mathbb{B}^{|\mathbb{X}|})^*$.

Satisfiability of a quadratic equation $eq : \mathtt{t}_\ell = \mathtt{t}_r$ with the length constraint φ_i can then be expressed as the RMC problem $(\mathcal{I}^{len}_{\varphi_i}, \mathcal{T}^{len}_{x \rightarrowtail \alpha x} \cup \mathcal{T}^{len}_{x \rightarrowtail \epsilon}, \mathcal{D}^{len}_{\varphi_i})$ instantiating Algorithm 1 with items given as follows (note the use of a fresh delimiter $\#_{len}$ for length constraints):

$$\begin{aligned} \mathcal{I}^{len}_{\varphi_i} &= \mathcal{I}^{eq}.\{\#_{len}\}.\mathcal{I}_{\varphi_i} \qquad \mathcal{D}^{len}_{\varphi_i} = \mathcal{D}^{eq}.\{\#_{len}\}.(\mathbb{B}^{|\mathbb{X}|})^* \\ \mathcal{T}^{len}_{x \rightarrowtail \alpha x} &= \bigcup_{y \in \mathbb{X}, z \in \mathbb{X}} T^{\leq 2}_{y \rightarrowtail zy}.\{\#_{len} \mapsto \#_{len}\}.T^{len}_{y \rightarrowtail zy} \cup \\ &\quad \bigcup_{y \in \mathbb{X}, a \in \Sigma} T^{\leq 2}_{y \rightarrowtail ay}.\{\#_{len} \mapsto \#_{len}\}.T^{len}_{y \rightarrowtail ay} \\ \mathcal{T}^{len}_{x \rightarrowtail \epsilon} &= \bigcup_{y \in \mathbb{X}} T^{\leq 2}_{y \rightarrowtail \epsilon}.\{\#_{len} \mapsto \#_{len}\}.T^{len}_{y \rightarrowtail \epsilon} \end{aligned}$$

Extensions to a system of equations and (more generally) a Boolean combination of constraints can be done in the same manner as in Sections 4 and 5.

Rationality of $\mathcal{T}^{len}_{x \rightarrowtail \alpha x}$ and $\mathcal{T}^{len}_{x \rightarrowtail \epsilon}$ follows directly from Proposition 1. The soundness and completeness of our algorithm is summarized by Lemma 15. Termination is an open problem even for quadratic equations (cf. Büchi and Senger (1990)).

Lemma 15. *Given a quadratic word equation $eq : \mathtt{t}_\ell = \mathtt{t}_r$ with the length constraint φ_i, Algorithm 1 instantiated with $(\mathcal{I}^{len}_{\varphi_i}, \mathcal{T}^{len}_{x \rightarrowtail \alpha x} \cup \mathcal{T}^{len}_{x \rightarrowtail \epsilon}, \mathcal{D}^{len}_{\varphi_i})$ is sound and complete.*

Proof. We can generalize nodes of the Nielsen proof graph to pairs of the form $(\mathtt{t}'_\ell = \mathtt{t}'_r, f)$ where f is a mapping assigning lengths to variables from $\mathbb{X}$. The transformation rules can be straightforwardly generalized to take into account also the lengths. The initial nodes are pairs $(\mathtt{t}_\ell = \mathtt{t}_r, f)$ where f is a model of φ_i. The final nodes are nodes $(\epsilon = \epsilon, g)$ where g is arbitrary. Note that the generalized graph is not necessarily finite even for quadratic equations. Nevertheless, if the equation is satisfiable then there is a finite path from an initial node to a final node.

Directly from the definition of $\mathcal{I}^{len}_{\varphi_i}$ we have that the initial nodes of the generalized proof graph are encoded strings from $\mathcal{I}^{len}_{\varphi_i}$ and the final nodes correspond to $\mathcal{D}^{len}_{\varphi_i}$. We can also see that the transformation rules correspond to the encoded relations $\mathcal{T}^{len}_{x\mapsto\alpha x}$ and $\mathcal{T}^{len}_{x\mapsto\epsilon}$. Since the search in Algorithm 1 implements a BFS strategy, we get that our (semi-)algorithm is sound and complete in proving satisfiability. □

For the general (non-quadratic) case and the case of a Boolean combination of constraints, we can obtain a sound and complete (though non-terminating) procedure by using the transducers $\mathcal{T}^{i,\wedge eq\star}$ in the same way as in Section 4.2 and modifying the proof of Lemma 15 accordingly.

6.2. Regular constraints

Our second extension of the framework is the support of regular constraints as a conjunction of atoms of the form $x \in \mathcal{L}(\mathcal{A})$ for an FA $\mathcal{A}$ over Σ (note that the negation of an atom $x \notin \mathcal{L}(\mathcal{A})$ can be converted to the positive atom $x \in \mathcal{L}(\mathcal{A}^{\complement})$ where $\mathcal{A}^{\complement}$ is a complement of the FA $\mathcal{A}$). In particular, we assume that regular constraints are represented by a conjunction φ_r of ℓ atoms of the form

$$\varphi_r \triangleq \bigwedge_{i=1}^{\ell} (x_i \in \mathcal{L}(\mathcal{A}_i)), \tag{27}$$

where $\mathcal{A}_i$ is an FA for each $1 \leq i \leq \ell$. Without loss of generality, we assume that the automata occurring in φ_r have pairwise disjoint sets of states and, further, we use $\mathcal{A}_r = (Q_r, \Sigma, \delta_r, I_r, F_r)$ to denote the automaton constructed as the disjoint union of all automata occurring in formula φ_r. Note that the disjoint union of two FAs $\mathcal{A}_1 = (Q_1, \Sigma, \Delta_1, Q_i^1, Q_f^1)$ and $\mathcal{A}_1 = (Q_2, \Sigma, \Delta_2, Q_i^2, Q_f^2)$ is the FA $\mathcal{A}_1 \uplus \mathcal{A}_2 = (Q_1 \uplus Q_2, \Sigma, \Delta_1 \uplus \Delta_2, Q_i^1 \uplus Q_i^2, Q_f^1 \uplus Q_f^2)$.

The approach we developed here is inspired by the approach in Norn (cf. (Abdulla et al., 2015)), but the idea needed to be significantly modified to fit in our (more proof-based) framework. In particular, we encode regular constraints as words over symbols of the form $\langle x, p, q\rangle$ where $x \in \mathbb{X}$ and $p, q \in Q_r$. We denote the set of all such symbols as $\Gamma_{\mathbb{X},\mathcal{A}_r}$. Moreover, we treat the words as sets of symbols and hence we assume a fixed linear order $\preccurlyeq$ over symbols to allow a unique representation. In particular, for a word $w \in \Gamma^*_{\mathbb{X},\mathcal{A}_r}$ we use $w_{\preccurlyeq}$ to denote the string containing symbols sorted by $\preccurlyeq$ with no repetitions of symbols. A single atom $x \in \mathcal{L}(\mathcal{A}_i)$ for $\mathcal{A} = (Q_i, \Sigma, \delta_i, I_i, F_i)$ can be encoded as a set of words $\mathsf{encode}(x \in \mathcal{L}(\mathcal{A}_i)) = \{\langle x, p, q\rangle \mid p \in I_i, q \in F_i\}$. The set represents all possible combinations of initial and final states in $\mathcal{A}_i$. The initial set $\mathcal{I}_{\varphi_r}$ is then defined as

$$\mathcal{I}_{\varphi_r} = \{w_{\preccurlyeq} \in \Gamma^*_{\mathbb{X},\mathcal{A}_r} \mid w \in \mathsf{encode}(x_1 \in \mathcal{L}(\mathcal{A}_1)) \ldots \mathsf{encode}(x_\ell \in \mathcal{L}(\mathcal{A}_\ell))\}. \tag{28}$$

Note that $\mathcal{I}_{\varphi_r}$ is finite for a finite $\mathbb{X}$, therefore it is a regular language.

Let us now describe the effect of the Nielsen transformation on the regular constraint part. Consider a word w encoding a set of symbols from $\Gamma_{\mathbb{X},\mathcal{A}_r}$. Then, the transformation $x \mapsto yx$ for $x, y \in \mathbb{X}$ applied to w produces words w' encoding sets where each occurrence of a symbol $\langle x, p, q\rangle$ is replaced with all possible pairs of symbols $\langle y, p, s\rangle$ and $\langle x, s, q\rangle$ where $p \rightsquigarrow s$ and $s \rightsquigarrow q$ in $\mathcal{A}_r$ (we use $q \rightsquigarrow q'$ to denote that there is a path from q to q' in the transition diagram of $\mathcal{A}_r$). Similarly, the transformation $x \mapsto ax$ for $x \in \mathbb{X}, a \in \Sigma$ applied to w produces words w' encoding sets where each occurrence of a symbol $\langle x, p, q\rangle$ is replaced with all possible symbols $\langle x, r, q\rangle$ where $p \text{-}\{a\}\!\triangleright r$ in $\mathcal{A}_r$. Finally, by the transformation $x \mapsto \epsilon$ we obtain a string $w' = w$ only if all symbols of w related to the variable x are of the form $\langle x, q, q\rangle$ for $q \in Q$. Formally, we first define the function expanding a single symbol for variables x and y as

$$exp_{x,y}(\sigma) = \begin{cases} \{\langle y, p, r\rangle.\langle x, r, q\rangle \mid p \rightsquigarrow r \rightsquigarrow q \text{ in } \mathcal{A}_r\} & \text{if } \sigma = \langle x, p, q\rangle, \\ \{\sigma\} & \text{otherwise.} \end{cases} \tag{29}$$

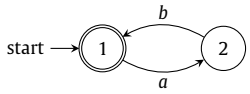
Similarly, we define the expansion

$$exp_{x,a}(\sigma) = \begin{cases} \{\langle x, r, q\rangle \mid p \text{-}\{a\}\!\triangleright r \rightsquigarrow q \text{ in } \mathcal{A}_r\} & \text{if } \sigma = \langle x, p, q\rangle, \\ \{\sigma\} & \text{otherwise.} \end{cases} \tag{30}$$

Then, the transformations $x \mapsto yx$, $x \mapsto ax$, and $x \mapsto \epsilon$ can be described by the following relations:

$$\begin{aligned} T^{reg}_{x\mapsto yx} &= \{(w, u_{\preccurlyeq}) \mid u \in exp_{x,y}(w[1]) \ldots exp_{x,y}(w[|w|])\}, \\ T^{reg}_{x\mapsto ax} &= \{(w, u_{\preccurlyeq}) \mid u \in exp_{x,a}(w[1]) \ldots exp_{x,a}(w[|w|])\}, \text{ and} \\ T^{reg}_{x\mapsto\epsilon} &= \Big\{ (w, w) \mid \forall 1 \leq i \leq |w| : \\ &\qquad \forall p, q \in Q : w[i] = \langle x, p, q\rangle \Rightarrow p = q \Big\}. \end{aligned} \tag{31}$$

Example 7. Consider the regular constraint $x \in \mathcal{L}(\mathcal{A}_x)$ with $\mathcal{A}_x$ given below.



Then, the corresponding values will be as follows:

$$\begin{aligned} &\mathsf{encode}(x \in \mathcal{L}(\mathcal{A}_x)) = \{\langle x, 1, 1\rangle\} \\ &\quad exp_{x,y}(\langle x, 1, 1\rangle) = \{\langle y, 1, 1\rangle.\langle x, 1, 1\rangle,\ \langle y, 1, 2\rangle.\langle x, 2, 1\rangle\} \\ &\quad exp_{x,a}(\langle x, 1, 1\rangle) = \{\langle x, 2, 1\rangle\} \end{aligned}$$

Moreover, $T^{reg}_{x\mapsto yx}$ will, e.g., contain pairs $(\langle x, 1, 1\rangle.\langle y, 1, 2\rangle, u_{\preccurlyeq})$ with

$$u_{\preccurlyeq} \in \{\langle x, 1, 1\rangle.\langle y, 1, 1\rangle.\langle y, 1, 2\rangle,\ \langle x, 2, 1\rangle.\langle y, 1, 2\rangle\}$$

(we assume that $\preccurlyeq$ is a lexicographic ordering on the components). Note that symbols in the words are sorted by $\preccurlyeq$ with duplicates removed. Similarly, $T^{reg}_{x\mapsto ax}$ will, e.g., contain the pair $(\langle x, 1, 1\rangle.\langle y, 1, 2\rangle,\ \langle x, 2, 1\rangle.\langle y, 1, 2\rangle)$, and $T^{reg}_{x\mapsto\epsilon}$ will contain $(\langle x, 1, 1\rangle.\langle y, 1, 2\rangle,\ \langle x, 1, 1\rangle.\langle y, 1, 2\rangle)$. □

The following lemma shows that the transformations are rational. In the proof, we first construct MSO(Str) formulae realizing necessary set operations on strings and the effect of the expanding function. Based on them, we construct formulae realizing the transformations, by means of the relation $\mathsf{pad}_{\square}(T)$ appending to w and w' with $(w, w') \in T$ an arbitrary number of symbols $\square$ (cf. Section 3.4).

Lemma 16. *The relations* $\mathsf{pad}_{\square}(T^{reg}_{x\mapsto yx})$, $\mathsf{pad}_{\square}(T^{reg}_{x\mapsto ax})$, *and* $\mathsf{pad}_{\square}(T^{reg}_{x\mapsto\epsilon})$ *are rational.*

Proof. In this proof, we extend the total order $\preccurlyeq$ on $\Gamma_{\mathbb{X},\mathcal{A}_r}$ to a total order on $\Gamma_{\mathbb{X},\mathcal{A}_r} \cup \{\square\}$ such that $\forall \sigma \in \Gamma_{\mathbb{X},\mathcal{A}_r} : \sigma \preccurlyeq \square$. (We note that encoding $\preccurlyeq$ in MSO(Str) is simple.) We define the relations $T^{reg}_{x\mapsto yx}$ and $T^{reg}_{x\mapsto\epsilon}$ using MSO(Str). The relation $T^{reg}_{x\mapsto ax}$ can be defined analogously to $T^{reg}_{x\mapsto yx}$.

$$\psi^{reg}_{x\mapsto yx}(w, w') \triangleq \exists^{\mathbb{W}} u_1, u_2, u_3 \Big(filter_x(u_1, u_2, w) \wedge expand^y_x(u_1, u_3) \wedge union(u_2, u_3, w') \wedge ordSet(w') \Big) \tag{32}$$

$$\psi^{reg}_{x\rightarrowtail\epsilon}(w, w') \triangleq \forall^{\mathbb{P}} i\Big(w[i] = w'[i] \wedge \bigvee_{\substack{\xi\in\Gamma_{\mathbb{X}\setminus\{x\},\mathcal{A}_r}\cup \\ \{\langle x,q,q\rangle \mid q\in Q_r\}}} w'[i] = \xi\Big) \tag{33}$$

where $filter_x(u, v, w)$ partitions the symbols of w to u and v such that u contains symbols that are of the form $\langle x, -, -\rangle$, i.e., of the form $\langle x, q, s\rangle$ for arbitrary q and s, and v contains the remaining ones; $expand^y_x(u, v)$ replaces each symbol $\langle x, p, q\rangle$ in u with $\langle y, p, s\rangle$ and $\langle x, s, q\rangle$ in v; and *union* is a set-like union. These predicates (including auxiliary predicates) are defined as follows:

$$\sigma \in w \triangleq \exists^{\mathbb{P}} i(w[i] = \sigma) \tag{34}$$

$$set(u) \triangleq \neg\exists^{\mathbb{P}} i, j(i \neq j \wedge u[i] = u[j]) \tag{35}$$

$$ordSet(u) \triangleq set(u) \wedge \forall^{\mathbb{P}} i, j(i < j \rightarrow u[i] \preccurlyeq u[j]) \tag{36}$$

$$filter_x(u, v, w) \triangleq$$
$$\forall^{\mathbb{P}} i \Bigg(\bigwedge_{q,s\in Q_r} (w[i] = \langle x, q, s\rangle \rightarrow (u[i] = \langle x, q, s\rangle \wedge v[i] = \square)) \wedge \bigwedge_{\substack{z\in\mathbb{X}\setminus\{x\} \\ q,s\in Q_r}} (w[i] = \langle z, q, s\rangle \rightarrow (u[i] = \square \wedge v[i] = \langle z, q, s\rangle)) \Bigg) \tag{37}$$

$$expand^y_x(u, v) \triangleq \tag{38}$$
$$\bigwedge_{\substack{s,q\in Q_r, s\rightsquigarrow q \\ \xi'=\langle x,s,q\rangle}} \Big(\xi' \in v \rightarrow \bigvee_{\substack{p\in Q_r, p\rightsquigarrow s \\ \xi=\langle x,p,q\rangle \\ \xi''=\langle y,p,s\rangle}} \xi \in u \wedge \xi'' \in v\Big) \wedge \tag{39}$$
$$\bigwedge_{\substack{p,s\in Q_r, p\rightsquigarrow s \\ \xi''=\langle y,p,s\rangle}} \Big(\xi'' \in v \rightarrow \bigvee_{\substack{q\in Q_r, s\rightsquigarrow q \\ \xi=\langle x,p,q\rangle \\ \xi'=\langle x,s,q\rangle}} \xi \in u \wedge \xi' \in v\Big) \wedge \tag{40}$$
$$\bigwedge_{\substack{p,q\in Q_r \\ \xi=\langle x,p,q\rangle}} \Big(\xi \in u \rightarrow \bigvee_{\substack{s\in Q_r, p\rightsquigarrow s\rightsquigarrow q \\ \xi''=\langle y,p,s\rangle \\ \xi'=\langle x,s,q\rangle}} \xi'' \in u \wedge \xi' \in v\Big) \tag{41}$$

$$union(u, v, w) \triangleq \bigwedge_{\xi\in\Gamma_{\mathbb{X},\mathcal{A}_r}} \xi \in w \leftrightarrow (\xi \in u \vee \xi \in v) \tag{42}$$

Intuitively, in the definition of $filter_x(u, v, w)$, the first part picks from w symbols containing x and adds them into u and the second part picks from w the other symbols and adds them into v. On the other hand, in the definition of $expand^y_x(u, v)$, we use s to denote the *splitting* state on the path from state p to state q. Then, the first and the second parts of the formula denote that $\langle y, p, s\rangle$ and $\langle x, s, q\rangle$ are in v while the last part denotes that $\langle x, p, q\rangle$ is in u.

We further consider the relations $\tau_{+\mathsf{pad}} = \{(w, w') \mid w \in (\Gamma_{\mathbb{X},\mathcal{A}_r} \cup \{\square\})^*, w' \in w.\{\square\}^*\}$ and $\tau_{-\mathsf{pad}} = \{(w, w') \mid w' \in (\Gamma_{\mathbb{X},\mathcal{A}_r} \cup \{\square\})^*, w \in w'.\{\square\}^*\}$ appending and removing padding, respectively. These relations are rational. Then, observe that $\mathsf{pad}_\square(T^{reg}_{x\rightarrowtail yx}) = \tau_{+\mathsf{pad}} \circ \tau_{-\mathsf{pad}} \circ \mathcal{L}(\psi^{reg}_{x\rightarrowtail yx}) \circ \tau_{+\mathsf{pad}}$. Recall the relation $\mathsf{pad}_\square(T)$ appends to w' in $(w, w') \in T$ an arbitrary number of symbols $\square$ (cf. Section 3.4). From Propositions 1 and 2, we have that $\mathsf{pad}_\square(T^{reg}_{x\rightarrowtail yx})$ is rational (the same for $\mathsf{pad}_\square(T^{reg}_{x\rightarrowtail\epsilon})$). □

The last missing piece is a definition of the destination set containing all satisfiable regular constraints. For a variable $x \in \mathbb{X}$, we define the set of satisfiable x-constraints as $L^x = \{ w_{\preccurlyeq} \mid w = \langle x, q_1, r_1\rangle \cdots \langle x, q_n, r_n\rangle \in \Gamma^*_{\mathbb{X},\mathcal{A}_r}, \bigcap_{i=1}^n \mathcal{L}_{\mathcal{A}_r}(q_i, r_i) \neq \emptyset \}$. Then, the destination set for a set of variables $\mathbb{X} = \{x_1, \ldots, x_k\}$ is given as $\mathcal{D}_{reg} = \{ w_{\preccurlyeq} \mid w \in L^{x_1} \cdots L^{x_k} \}$. As in the case of $\mathcal{I}_{\varphi_r}$, the set $\mathcal{D}_{reg}$ is finite and hence regular as well.

Satisfiability of a quadratic word equation $eq\colon \mathsf{t}_\ell = \mathsf{t}_r$ with a regular constraint φ_r can be expressed in the RMC framework as $(\mathcal{I}^{reg}_{\varphi_r}, \mathcal{T}^{reg}_{x\rightarrowtail\alpha x} \cup \mathcal{T}^{reg}_{x\rightarrowtail\epsilon}, \mathcal{D}^{reg}_{\varphi_r})$ instantiating Algorithm 1 with items given in as follows (note that we use a fresh delimiter $\#_{reg}$):

$$\mathcal{I}^{reg}_{\varphi_r} = \mathcal{I}^{eq}.\{\#_{reg}\}.\mathsf{pad}_\square(\mathcal{I}_{\varphi_r}) \qquad \mathcal{D}^{reg}_{\varphi_r} = \mathcal{D}^{eq}.\{\#_{reg}\}.\mathsf{pad}_\square(\mathcal{D}_{reg})$$
$$\mathcal{T}^{reg}_{x\rightarrowtail\alpha x} = \bigcup_{y\in\mathbb{X}, z\in\mathbb{X}} T^{\leq 2}_{y\rightarrowtail zy}.\{\#_{reg} \mapsto \#_{reg}\}.\mathsf{pad}_\square(T^{reg}_{y\rightarrowtail zy}) \cup \bigcup_{y\in\mathbb{X}, a\in\Sigma} T^{\leq 2}_{y\rightarrowtail ay}.\{\#_{reg} \mapsto \#_{reg}\}.\mathsf{pad}_\square(T^{reg}_{y\rightarrowtail ay})$$
$$\mathcal{T}^{reg}_{x\rightarrowtail\epsilon} = \bigcup_{y\in\mathbb{X}} T^{\leq 2}_{y\rightarrowtail\epsilon}.\{\#_{reg} \mapsto \#_{reg}\}.\mathsf{pad}_\square(T^{reg}_{y\rightarrowtail\epsilon})$$

The rationality of $\mathcal{T}^{reg}_{x\rightarrowtail\alpha x}$ and $\mathcal{T}^{reg}_{x\rightarrowtail\epsilon}$ follows directly from Proposition 1. The soundness and completeness of our procedure is summarized by the following lemma.

Lemma 17. *Given a quadratic word equation $eq\colon \mathsf{t}_\ell = \mathsf{t}_r$ with a regular constraint φ_r, Algorithm 1 instantiated with $(\mathcal{I}^{reg}_{\varphi_r}, \mathcal{T}^{reg}_{x\rightarrowtail\alpha x} \cup \mathcal{T}^{reg}_{x\rightarrowtail\epsilon}, \mathcal{D}^{reg}_{\varphi_r})$ is sound, complete, and terminating.*

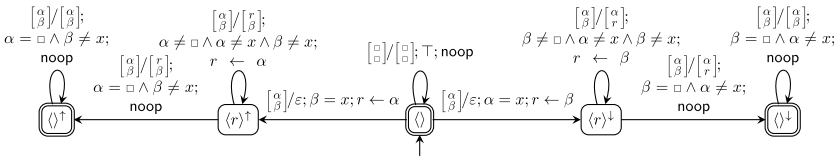
Proof. Similarly to proof of Lemma 15, we can generalize nodes of the Nielsen proof graph to pairs of the form $(\mathsf{t}'_\ell = \mathsf{t}'_r, S)$ where $S \subseteq \Gamma_{\mathbb{X},\mathcal{A}_r}$. The transformation rules can be straightforwardly generalized to take into account also the regular constraints represented by a subset of $\Gamma_{\mathbb{X},\mathcal{A}_r}$. Since $\Gamma_{\mathbb{X},\mathcal{A}_r}$ is finite and eq is quadratic, the generalized proof graph is finite. The initial nodes of the generalized proof graph are exactly encoded strings from $\mathcal{I}^{reg}_{\varphi_r}$, the final nodes correspond to $\mathcal{D}^{reg}_{\varphi_r}$, and the transformation rules correspond to the encoded relations $\mathcal{T}^{reg}_{x\rightarrowtail\alpha x}$ and $\mathcal{T}^{reg}_{x\rightarrowtail\epsilon}$. Since our RMC framework implements the BFS strategy, from the previous we get that our procedure is sound, complete, and terminating in proving satisfiability. □

Table 1
Summary of the proposed approach on various types of string constraints with extensions. S stands for *sound*, C stands for *complete*, and T stands for *terminating*.

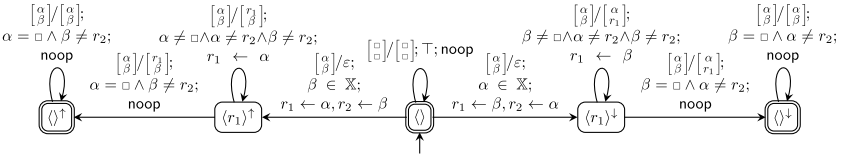
	Quadratic system	*General system*	*Boolean combination*
Pure	SCT (Section 4.1)	SC (Section 4.2)	SC (Section 5)
Length	SC (Section 6.1)	SC (Section 6.1)	SC (Section 6.1)
Regular	SCT (Section 6.2)	SC (Section 6.2)	SC (Section 6.2)

As in the case of length constraints, the satisfiability of a word equation eq with regular constraints can be generalized to a system of equations Φ with regular constraints. The languages/relations corresponding to eq are replaced by languages/relations corresponding to Φ. For a system of word equations with regular constraints our algorithm is still sound (and complete if we consider the transducers $\mathcal{T}^{i,\wedge eq\star}$ for Φ).

Discussion. In the previous sections, we elaborated the proposed RMC framework for various kinds of string constraints including length and regular extensions. In Table 1, we give a summary of the achieved results. If the system of equations is quadratic (and even enriched with regular extensions), then our RMC approach is sound, complete, and terminating. It is basically due to the fact that the language *processed* in Algorithm 1 for the transformations tailored for quadratic equations becomes saturated after a finite number of steps. In all other cases, our RMC approach is sound and complete (but not generally terminating). For suitable encoded transformations, we are able to reach a solution after a finite number of steps (if the system is satisfiable). But in general, the language *processed* in Algorithm 3 for these transformations is not guaranteed to eventually become saturated (see Fig. 11).



(a) An FRT for the relation $T^{\leq 1}_{x\mapsto\varepsilon}$ for an arbitrary set of variables $\mathbb{X}$ and alphabet Σ



(b) An FRT for the relation $\bigcup_{x\in\mathbb{X}} T^{\leq 1}_{x\mapsto\varepsilon}$ for an arbitrary set of variables $\mathbb{X}$ and alphabet Σ

Fig. 11. FRT (a) implementing the relation $T^{\leq 1}_{x\mapsto\epsilon}$ for an arbitrary set of variables $\mathbb{X}$ and alphabet Σ. The register r is used to store a shifted symbol. FRT (b) implementing the relation $\bigcup_{x\in\mathbb{X}} T^{\leq 1}_{x\mapsto\epsilon}$ for an arbitrary set of variables $\mathbb{X}$ and alphabet Σ. The register r_1 is used to store a shifted symbol and the register r_2 is used to store a variable that should be removed. In the figures, α and β are variables representing the input symbol, r, r_1, and r_2 are registers, and the transitions are of the form *action;condition;register update*.

7. Implementation

We created a prototype Python tool called Retro,[1] where we implemented the symbolic procedure for solving systems of word equations. Retro implements a modification of the RMC loop from Algorithm 1. In particular, instead of standard transducers defined in Section 2, it uses the so-called *finite-alphabet register transducers* (FRTs), which allow a more concise representation of a rational relation.

Informally, an FRT is a register automaton (in the sense of Kaminski and Francez (1994) and Demri and Lazic (2009)) where the alphabet is finite. The finiteness of the alphabet implies that the expressive power of FRTs coincides with the class of rational languages, but the advantage of using FRTs is that they allow a more concise representation than ordinary transducers. One can consider FRTs to be the restriction of symbolic transducers with registers of Veanes et al. (2012) to finite alphabets. Operations for dealing with these transducers are then straightforward restrictions of the operations considered by Veanes et al. (2012) and therefore do not elaborate on it here.

In particular, transducers (without registers) corresponding to the transformers $\mathcal{T}_{x\rightarrowtail\alpha x}$ and $\mathcal{T}_{x\rightarrowtail\epsilon}$ contain branching at the beginning for each choice of x and α. Especially in the case of huge alphabets, this yields huge transducers (consider for instance the Unicode alphabet with over 1 million symbols). The use of FRTs yields much smaller automata because the choice of x and α is stored into registers and then processed symbolically. To illustrate the effect of using registers, consider the transducer shown in Fig. 8 implementing the encoded relation $T^{\leq 1}_{x\mapsto\epsilon}$ for $\mathbb{X} = \{x\}$ and $\Sigma = \{a\}$. The full transducer for large alphabets would require a branching for each $\left[\begin{smallmatrix}u\\v\end{smallmatrix}\right]$, with $u, v \in \Sigma_{\mathbb{X}}$, and a lot of states to store the concrete shifted symbols. In particular, it requires at least one pair of states $\langle v\rangle^{\uparrow}$ and $\langle v\rangle^{\downarrow}$ for each $v \in \Sigma_{\mathbb{X}}$, which is unfeasible for very large $\Sigma_{\mathbb{X}}$. On the other hand, the FRT in Fig. 11(a) stores the shifted symbols in the register r, the branching is replaced by a symbolic transition, and hence it requires just a couple of states and transitions. Moreover, using an additional register r_2 to store the variable to replace, we are able to efficiently represent the relation $\bigcup_{x\in\mathbb{X}} T^{\leq 1}_{x\mapsto\epsilon}$, as shown in Fig. 11(b).

Concretely, Retro implements the decision procedure for a system of general word equations and length constraints (i.e., the procedures covered in Sections 3, 4 and 6.1). It does not implement (i) Boolean combinations of constraints (Section 5) and (ii) regular constraints (Section 6.2), which are quite inefficient and are provided in order for our approach to have a more robust theoretical formal basis for a large fragment of input constraints (cf. the discussion at the beginning of Section 5).

As another feature, Retro uses deterministic finite automata to represent configurations in Algorithm 1. It also uses eager automata minimization, since it has a big impact on the performance, especially on checking the termination condition of the RMC algorithm, which is done by testing language inclusion between the current configuration and all so-far processed configurations.

[1] available at https://github.com/VeriFIT/retro.

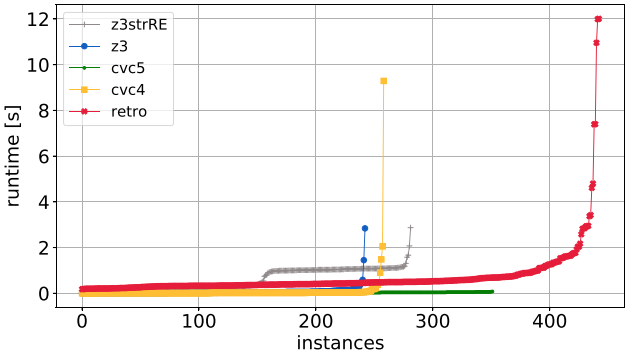
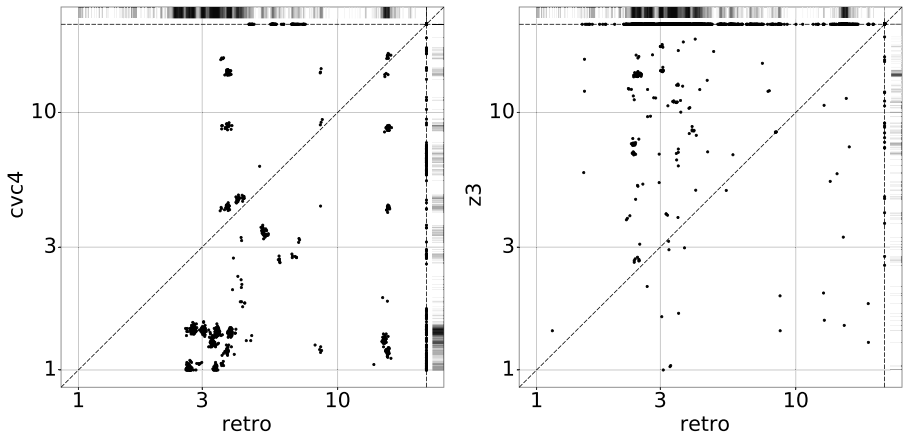


Fig. 12. A cactus plot comparing Retro, CVC4, CVC5, Z3, and Z3Str3RE on $\texttt{Kepler}_{22}$.

8. Experimental evaluation

We compared the performance of our approach (implemented in Retro) with four current state-of-the-art SMT solvers that support the string theory: Z3 4.8.14 (de Moura and Bjørner, 2008), Z3Str3RE (Berzish et al., 2021), CVC4 1.8 (Barrett et al., 2011), and CVC5 1.0.1 (Barbosa et al., 2022). Regarding other solvers that we are aware of, the performance of Norn from Abdulla et al. (2015) and Ostrich from Chen et al. (2019b) was much worse than the considered tools, the performance of Z3Str4 (Mora et al., 2021) was similar to that of Z3Str3RE, and Sloth of Holík et al. (2018) was unsound on the considered fragment (it supports only the so-called *straight-line fragment*) (see Fig. 13)

The first set of benchmarks is $\texttt{Kepler}_{22}$, obtained from Le and He (2018). $\texttt{Kepler}_{22}$ contains 600 hand-crafted string constraints composed of quadratic word equations with length constraints. In Fig. 12, we give a cactus plot of the results of the solvers



(a) Retro vs CVC4 (b) Retro vs Z3

Fig. 13. Comparison of Retro with CVC4 and Z3 on PyEx-Hard. We show difficult instances that took more than 1 s to finish. Times are given in seconds, axes are logarithmic.

on the $\texttt{Kepler}_{22}$ benchmark set with the timeout of 20 s. In cactus plots, the closer a solver's plot is to the right and bottom borders, the better is the corresponding solver. The total numbers of solved benchmarks within the timeout were: 243 for Z3, 282 for Z3Str3RE, 259 for CVC4, 352 for CVC5, and 443 for Retro. We can refine these numbers by comparing the cases solved by each pair of tools, as shown in Table 2. From the table we can see that Retro solves significantly more benchmarks than all other state-of-the-art tools, from 92 when compared with CVC5 to 201 with Z3. Only in one case Retro failed (as did Z3Str3RE) while CVC4, CVC5, and Z3 succeeded. Except for CVC4 vs. CVC5, the comparison between the other tools is inconclusive, as the entries (X, Y) and (Y, X) of the table both contain large values.

The other set of benchmarks that we tried is PyEx-Hard. Here we wanted to see the potential of integrating Retro with DPLL(T)-based string solvers, like Z3 or CVC4, as a specific string theory solver. The input of this component is a conjunction of atomic string formulae (e.g., $xy = zb \wedge z = ax$) that is a model of the Boolean structure of the top-level formula. The conjunction of atomic string formulae is then, in several layers, processed by various string theory solvers, which either add more conflict clauses or return a model. To evaluate whether Retro is suitable to be used as "one of the layers" of Z3 or CVC4's string solver, we analyzed the PyEx benchmarks from Reynolds et al. (2017) and extracted from it 967 difficult instances that neither CVC4 nor Z3 could solve in 10 s. From those instances, we obtained 20,020 conjunctions of word equations that Z3's DPLL(T) algorithm sent to its string theory solver when trying to solve them. We call those 20,020 conjunctions of word equations PyEx-Hard. We then evaluated the other solvers on PyEx-Hard with the timeout of 20 s. Out of these, Z3 could not solve 3001, Z3Str3RE 814, CVC4 152, CVC5 171, and Retro could not solve 3079 instances.

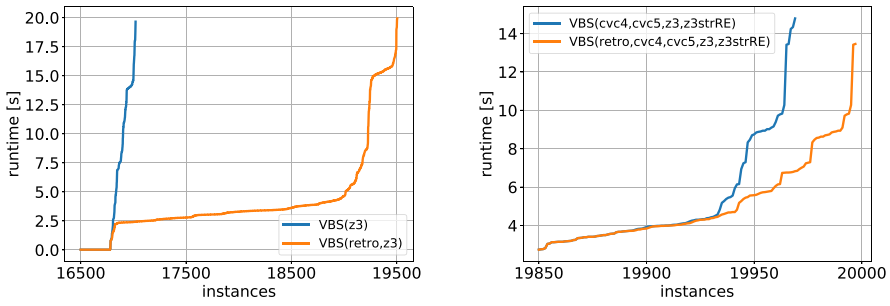
Let us now look closely at the hard instances in the PyEx-Hard benchmark set, in particular, on the instances that the other tools could not solve. These benchmarks cannot be handled by the (several layers of) fast heuristics implemented in these tools, which are sufficient to solve many benchmarks without the need to start applying the case-split rule.[2] In Fig. 13 we give a comparison of the running times of Retro with CVC4 and Z3 with a particular focus on the difficult instances (running time above 1 s). The plots about CVC5 and Z3Str3RE show similar trends. From the figure we can see that on many hard instances Retro can provide the answer much faster than the other tool, in particular for Z3. When we compare the solvers on the examples that the other tools failed to solve, we have that Retro could solve 86 examples (56.2%) out of those where CVC4 failed, 130 examples (76.02%) where CVC5 failed, 2484 examples (82.7%) where Z3 failed, and 519 examples (63.75%) where Z3Str3RE failed. Moreover, Retro solved 28 instances as the only tool. Lastly, we consider how Retro affects the *Virtual Best Solver*: given a set of solvers S, we use $VBS(S)$ to denote the solver that would be obtained by taking, for each benchmark, the fastest solver on the given benchmark. In Fig. 14, we provide cactus plots showing the impact of Retro on two instances of Virtual Best Solvers; the plots for Z3Str3RE, CVC4, and CVC5 are similar, with the one for Z3Str3RE being closer to Fig. 14(a) while the ones for CVC4/5 being closer to Fig. 14(b). As we can see, the plots show that our approach can significantly help solvers deal with hard equations.

Table 2
A break-down of solved cases on the $\texttt{Kepler}_{22}$ benchmark. A number on row X in column Y denotes the number of cases that solver X could solve and solver Y could not solve.

	Retro	CVC4	CVC5	Z3	Z3Str3RE
Retro	–	185	92	201	161
CVC4	1	–	2	104	63
CVC5	1	95	–	197	145
Z3	1	88	88	–	23
Z3Str3RE	0	86	75	62	–

Discussion. From the obtained results, we see that our approach works well in *difficult cases*, where the fast heuristics implemented in state-of-the-art solvers are not sufficient to quickly discharge a formula, which happens in particular when the (un)satisfiability proof is complex. Our approach can exploit the symbolic representation of the proof tree and use it to reduce the redundancy of performing transformations. Note that we can still beat the heavily optimized Z3, Z3Str3RE, CVC4, and CVC5 written in C++ by a Python prototype in those cases. We believe that

[2] For instance, when Z3 receives the word equation $xy = yax$, it infers the length constraint $|x| + |y| = |y| + 1 + |x|$, which implies unsatisfiability of the word equation without the need to start applying the case-split rule at all.



(a) VBS(Retro, Z3) (b) VBS(Retro, CVC4, CVC5, Z3, Z3Str3RE)

Fig. 14. A cactus plot comparing the Virtual Best Solver of (a) Z3 with and without Retro and (b) all tools with and without Retro on the PyEx-Hard benchmark. We show only the most difficult benchmarks (out of 20,020).

implementing our symbolic algorithm as a part of a state-of-the-art SMT solver would push the applicability of string solving even further, especially for cases of string constraints with a complex structure, which need to solve multiple DPLL(T) queries in order to establish the (un)satisfiability of a string formula.

9. Related work

The study of solving string constraint traces back to 1946, when (Quine, 1946) showed that the first-order theory of word equations is undecidable. Makanin (1977) achieved a milestone result by showing that the class of quantifier-free word equation is decidable. Since then, several works, e.g., Plandowski (1999, 2006), Matiyasevich (2008), Robson and Diekert (1999), Schulz (1990), Ganesh et al. (2012), Ganesh and Berzish (2016), Abdulla et al. (2014), Barceló et al. (2013), Lin and Barceló (2016), Chen et al. (2018, 2019b) and Abdulla et al. (2019), consider the decidability and complexity of different classes of string constraints. Efficient solving of satisfiability of string constraints is a challenging problem. Moreover, decidability of the problem of satisfiability of word equations combined with length constraints of the form $|x| = |y|$ has already been open for over 20 years (Büchi and Senger, 1990).

The strong practical motivation led to the rise of several string constraint solvers that concentrate on solving practical problem instances. The typical procedure implemented within *DPLL(T)-based* string solvers is to split the constraints into simpler subcases based on how the solutions are aligned, combining with powerful techniques for Boolean reasoning to efficiently explore the resulting exponentially-sized search space. The case-split rule is usually performed explicitly. Examples of solvers implementing this approach are Norn (Abdulla et al., 2014, 2015), Trau (Abdulla et al., 2018), Ostrich (Chen et al., 2019b), Sloth (Holík et al., 2018), CVC4 (Barrett et al., 2011), CVC5 (Barbosa et al., 2022), Z3str2 (Zheng et al., 2017), Z3str3 (Berzish et al., 2017), Z3str4 (Mora et al., 2021), Z3Str3RE (Berzish et al., 2021), S3 (Trinh et al., 2014), S3P (Trinh et al., 2016). In contrast, our approach performs case-splits symbolically.

Automata and transducers have been used in many approaches and tools for string solving, such as in Norn (Abdulla et al., 2014, 2015), Trau (Abdulla et al., 2018), Ostrich (Chen et al., 2019b), Sloth (Holík et al., 2018), Slog (Wang et al., 2016), Slent (Wang et al., 2018), or Z3Str3RE (Berzish et al., 2021), and also in string solvers for analyzing string-manipulating programs, such as ABC (Aydin et al., 2018) and Stranger (Yu et al., 2010), which soundly over-approximate string constraints using transducers (Yu et al., 2016). The main difference of these approaches to ours is that they use transducers to encode possible models (solutions) to the string constraints, while we use automata and transducers to encode the string constraint transformations. Other approaches for solving string constraints include reducing the constraints to the SMT theory of bit vectors (e.g., Zheng et al., 2017; Berzish et al., 2017; Mora et al., 2021; Kiezun et al., 2012), the theory of arrays (e.g., Li and Ghosh (2013)), or SAT-solving (e.g., Day et al., 2019; Amadini et al., 2017; Scott et al., 2017). Not so many approaches are based on algebraic approaches, such as the Nielsen transformation. In addition to our approach, it is also used as the basis of the work of Le and He (2018). On the other hand, the Nielsen transformation (Nielsen, 1917) is used by some tools that implement different approaches to discharge quadratic equations (e.g., Ostrich of Chen et al. (2019b)). Complex rewriting rules are used, e.g., when dealing with regular constraints in CVC5 (Nötzli et al., 2022).

CRediT authorship contribution statement

Yu-Fang Chen: Conceptualization, Methodology, Writing. **Vojtěch Havlena:** Conceptualization, Methodology, Writing, Software. **Ondřej Lengál:** Conceptualization, Methodology, Writing. **Andrea Turrini:** Conceptualization, Methodology, Writing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

We thank Mohamed Faouzi Atig for discussing the topic. This work has been partially supported by the National Natural Science Foundation of China (grant no. 61836005), the Chinese Academy of Sciences Project for Young Scientists in Basic Research (grant no. YSBR-040), the Czech Science Foundation project 19-24397S, the FIT BUT internal project FIT-S-20-6427, the Czech Ministry of Education, Youth and Sports project LL1908 of the ERC.CZ programme, and the project of Ministry of Science and Technology, Taiwan (grant no. MOST-109-2628-E-001-001-MY3). This work is part of the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant no. 101008233.

References

Abdulla, P.A., 2012. Regular model checking. STTT 14 (2), 109–118. http://dx.doi.org/10.1007/s10009-011-0216-8.

Abdulla, P.A., Atig, M.F., Chen, Y., Diep, B.P., Holík, L., Rezine, A., Rümmer, P., 2017. 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. ACM, pp. 602–617. http://dx.doi.org/10.1145/3062341.3062384.

Abdulla, P.A., Atig, M.F., Chen, Y.-F., Diep, B.P., Holík, L., Rezine, A., Rümmer, P., 2018. Trau: SMT solver for string constraints. In: 2018 Formal Methods in Computer Aided Design. FMCAD, IEEE, pp. 1–5.

Abdulla, P.A., Atig, M.F., Chen, Y.-F., Holík, L., Rezine, A., Rümmer, P., Stenman, J., 2014. String constraints for verification. In: International Conference on Computer Aided Verification. Springer, pp. 150–166.

Abdulla, P.A., Atig, M.F., Chen, Y.-F., Holík, L., Rezine, A., Rümmer, P., Stenman, J., 2015. Norn: An SMT solver for string constraints. In: International Conference on Computer Aided Verification. Springer, pp. 462–469.

Abdulla, P.A., Atig, M.F., Diep, B.P., Holík, L., Janků, P., 2019. Chain-free string constraints. In: ATVA. Springer, pp. 277–293.

Amadini, R., Gange, G., Stuckey, P.J., Tack, G., 2017. A novel approach to string constraint solving. In: Beck, J.C. (Ed.), Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings. In: Lecture Notes in Computer Science, vol. 10416, Springer, pp. 3–20. http://dx.doi.org/10.1007/978-3-319-66158-2_1.

Aydin, A., Eiers, W., Bang, L., Brennan, T., Gavrilov, M., Bultan, T., Yu, F., 2018. Parameterized model counting for string and numeric constraints. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM, pp. 400–410.

Barbosa, H., Barrett, C., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y., 2022. CVC5: A versatile and industrial-strength SMT solver. In: Fisman, D., Rosu, G. (Eds.), Tools and Algorithms for the Construction and Analysis of Systems. Springer International Publishing, Cham, pp. 415–442.

Barceló, P., Figueira, D., Libkin, L., 2013. Graph logics with rational relations. arXiv preprint arXiv:1304.4150.

Barrett, C.W., Conway, C.L., Deters, M., Hadarean, L., Jovanovic, D., King, T., Reynolds, A., Tinelli, C., 2011. CVC4. In: Gopalakrishnan, G., Qadeer, S. (Eds.), Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings. In: Lecture Notes in Computer Science, vol. 6806, Springer, pp. 171–177. http://dx.doi.org/10.1007/978-3-642-22110-1_14.

Berstel, J., 1979. Transductions and Context-Free Languages. In: Teubner Studienbücher : Informatik, vol. 38, Teubner, URL: http://www.worldcat.org/oclc/06364613.

Berzish, M., Ganesh, V., Zheng, Y., 2017. Z3str3: A string solver with theory-aware heuristics. In: 2017 Formal Methods in Computer Aided Design, FMCAD 2017. pp. 55–59. http://dx.doi.org/10.23919/FMCAD.2017.8102241.

Berzish, M., Kulczynski, M., Mora, F., Manea, F., Day, J.D., Nowotka, D., Ganesh, V., 2021. An SMT solver for regular expressions and linear arithmetic over string length. In: International Conference on Computer Aided Verification. Springer, pp. 289–312.

Bjørner, N., Tillmann, N., Voronkov, A., 2009. Path feasibility analysis for string-manipulating programs. In: Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference, TACAS 2009. In: Lecture Notes in Computer Science, vol. 5505, Springer, pp. 307–321. http://dx.doi.org/10.1007/978-3-642-00768-2_27.

Blotsky, D., Mora, F., Berzish, M., Zheng, Y., Kabir, I., Ganesh, V., 2018. Stringfuzz: A fuzzer for string solvers. In: International Conference on Computer Aided Verification. Springer, pp. 45–51.

Bouajjani, A., Habermehl, P., Rogalewicz, A., Vojnar, T., 2012. Abstract regular (tree) model checking. STTT 14 (2), 167–191. http://dx.doi.org/10.1007/s10009-011-0205-y.

Bouajjani, A., Jonsson, B., Nilsson, M., Touili, T., 2000. Regular model checking. In: Emerson, E.A., Sistla, A.P. (Eds.), Computer Aided Verification. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 403–418.

Büchi, J.R., 1960. Weak second-order arithmetic and finite automata. Math. Log. Q. 6 (1–6), 66–92. http://dx.doi.org/10.1002/malq.19600060105.

Büchi, J.R., Senger, S., 1990. Definability in the existential theory of concatenation and undecidable extensions of this theory. In: The Collected Works of J. Richard Büchi. Springer, pp. 671–683.

Cadar, C., Ganesh, V., Pawlowski, P.M., Dill, D.L., Engler, D.R., 2006. EXE: Automatically generating inputs of death. In: Proceedings of the 13th ACM Conference on Computer and Communications Security. CCS '06, Association for Computing Machinery, pp. 322–335. http://dx.doi.org/10.1145/1180405.1180445.

Chen, T., Chen, Y., Hague, M., Lin, A.W., Wu, Z., 2018. What is decidable about string constraints with the ReplaceAll function. PACMPL 2 (POPL), 3:1–3:29. http://dx.doi.org/10.1145/3158091.

Chen, T., Hague, M., Lin, A.W., Rümmer, P., Wu, Z., 2019a. Decision procedures for path feasibility of string-manipulating programs with complex operations. PACMPL 3 (POPL), 49:1–49:30. http://dx.doi.org/10.1145/3290362.

Chen, T., Hague, M., Lin, A.W., Rümmer, P., Wu, Z., 2019b. Decision procedures for path feasibility of string-manipulating programs with complex operations. Proc. ACM Program. Lang. 3 (POPL), 1–30.

Chen, Y., Havlena, V., Lengál, O., Turrini, A., 2020. A symbolic algorithm for the case-split rule in string constraint solving. In: Oliveira, B.C.d.S. (Ed.), Programming Languages and Systems - 18th Asian Symposium, APLAS 2020, Fukuoka, Japan, November 30 - December 2, 2020, Proceedings. In: Lecture Notes in Computer Science, vol. 12470, Springer, pp. 343–363. http://dx.doi.org/10.1007/978-3-030-64437-6_18.

Day, J.D., Ehlers, T., Kulczynski, M., Manea, F., Nowotka, D., Poulsen, D.B., 2019. On solving word equations using SAT. In: Filiot, E., Jungers, R.M., Potapov, I. (Eds.), Reachability Problems - 13th International Conference, RP 2019, Brussels, Belgium, September 11-13, 2019, Proceedings. In: Lecture Notes in Computer Science, vol. 11674, Springer, pp. 93–106. http://dx.doi.org/10.1007/978-3-030-30806-3_8.

Demri, S., Lazic, R., 2009. LTL with the freeze quantifier and register automata. ACM Trans. Comput. Log. 10 (3), 16:1–16:30. http://dx.doi.org/10.1145/1507244.1507246.

Diekert, V., 2002. Makanin's algorithm. In: Algebraic Combinatorics on Words. In: Encyclopedia of Mathematics and its Applications, Cambridge University Press, pp. 387–442. http://dx.doi.org/10.1017/CBO9781107326019.013.

Durnev, V.G., Zetkina, O.V., 2009. On equations in free semigroups with certain constraints on their solutions. J. Math. Sci. 158 (5), 671–676. http://dx.doi.org/10.1007/s10958-009-9409-z.

Ganesh, V., Berzish, M., 2016. Undecidability of a theory of strings, linear arithmetic over length, and string-number conversion. arXiv preprint arXiv:1605.09442.

Ganesh, V., Minnes, M., Solar-Lezama, A., Rinard, M., 2012. Word equations with length constraints: what's decidable? In: Haifa Verification Conference. Springer, pp. 209–226.

Glenn, J., Gasarch, W.I., 1996. Implementing WS1S via finite automata. In: Raymond, D.R., Wood, D., Yu, S. (Eds.), Automata Implementation, First International Workshop on Implementing Automata, WIA '96, London, Ontario, Canada, August 29-31, 1996, Revised Papers. In: Lecture Notes in Computer Science, vol. 1260, Springer, pp. 50–63. http://dx.doi.org/10.1007/3-540-63174-7_5.

Godefroid, P., Klarlund, N., Sen, K., 2005. DART: Directed automated random testing. SIGPLAN Not. 40 (6), 213–223. http://dx.doi.org/10.1145/1064978.1065036.

Gulwani, S., Jha, S., Tiwari, A., Venkatesan, R., 2011. Synthesis of loop-free programs. SIGPLAN Not. 46 (6), 62–73. http://dx.doi.org/10.1145/1993316.1993506.

Gulwani, S., Srivastava, S., Venkatesan, R., 2008. Program analysis as constraint solving. In: PLDI'08.

Holík, L., Janků, P., Lin, A.W., Rümmer, P., Vojnar, T., 2018. String constraints with concatenation and transducers solved efficiently. Proc. ACM Program. Lang. 2 (POPL), 4:1–4:32. http://dx.doi.org/10.1145/3158092.

Kaminski, M., Francez, N., 1994. Finite-memory automata. Theoret. Comput. Sci. 134 (2), 329–363. http://dx.doi.org/10.1016/0304-3975(94)90242-9.

Karp, R.M., 1972. Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (Eds.), Proceedings of a Symposium on the Complexity of Computer Computations, Held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA. In: The IBM Research Symposia Series, Plenum Press, New York, pp. 85–103. http://dx.doi.org/10.1007/978-1-4684-2001-2_9.

Kesten, Y., Maler, O., Marcus, M., Pnueli, A., Shahar, E., 2001. Symbolic model checking with rich assertional languages. Theoret. Comput. Sci. 256 (1–2), 93–112. http://dx.doi.org/10.1016/S0304-3975(00)00103-1.

Kiezun, A., Ganesh, V., Artzi, S., Guo, P.J., Hooimeijer, P., Ernst, M.D., 2012. HAMPI: a solver for word equations over strings, regular expressions, and context-free grammars. ACM Trans. Softw. Eng. Methodol. 21 (4), 25:1–25:28. http://dx.doi.org/10.1145/2377656.2377662.

King, J.C., 1976. Symbolic execution and program testing. Commun. ACM 19 (7), 385–394. http://dx.doi.org/10.1145/360248.360252.

Knoth, T., Wang, D., Polikarpova, N., Hoffmann, J., 2019. Resource-guided program synthesis. In: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. In: PLDI 2019, Association for Computing Machinery, New York, NY, USA, pp. 253–268. http://dx.doi.org/10.1145/3314221.3314602.

Kosovskii, N.K., 1976. Properties of the solutions of equations in a free semigroup. J. Math. Sci. 6 (4), http://dx.doi.org/10.1007/BF01084074.

Le, Q.L., He, M., 2018. A decision procedure for string logic with quadratic equations, regular expressions and length constraints. In: Ryu, S. (Ed.), Programming Languages and Systems - 16th Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2-6, 2018, Proceedings. In: Lecture Notes in Computer Science, vol. 11275, Springer, pp. 350–372. http://dx.doi.org/10.1007/978-3-030-02768-1_19.

Levi, F.W., 1944. On semigroups. Bull. Calcutta Math. Soc. 36, 141–146.

Li, G., Ghosh, I., 2013. PASS: String solving with parameterized array and interval automaton. In: Bertacco, V., Legay, A. (Eds.), Hardware and Software: Verification and Testing - 9th International Haifa Verification Conference, HVC 2013, Haifa, Israel, November 5-7, 2013, Proceedings. In: Lecture Notes in Computer Science, vol. 8244, Springer, pp. 15–31. http://dx.doi.org/10.1007/978-3-319-03077-7_2.

Liang, T., Reynolds, A., Tinelli, C., Barrett, C.W., Deters, M., 2014. A DPLL(T) theory solver for a theory of strings and regular expressions. In: Computer Aided Verification - 26th International Conference, CAV 2014. In: Lecture Notes in Computer Science, vol. 8559, Springer, pp. 646–662. http://dx.doi.org/10.1007/978-3-319-08867-9_43.

Lin, A.W., Barceló, P., 2016. String solving with word equations and transducers: towards a logic for analysing mutation XSS. In: ACM SIGPLAN Notices, Vol. 51. ACM, pp. 123–136.

Lin, A.W., Majumdar, R., 2018. Quadratic word equations with length constraints, counter systems, and Presburger arithmetic with divisibility. In: Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018. In: Lecture Notes in Computer Science, vol. 11138, Springer, pp. 352–369. http://dx.doi.org/10.1007/978-3-030-01090-4_21.

Loring, B., Mitchell, D., Kinder, J., 2019. Sound regular expression semantics for dynamic symbolic execution of JavaScript. In: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 425–438.

Makanin, G.S., 1977. The problem of solvability of equations in a free semigroup. Mat. Sb. 145 (2), 147–236.

Matiyasevich, Y.V., 1968. A connection between systems of word and length equations and Hilbert's tenth problem. Zap. Nauchnykh Semin. POMI 8, 132–144.

Matiyasevich, Y., 2008. Computation paradigms in light of Hilbert's tenth problem. In: New Computational Paradigms. Springer, pp. 59–85.

Mora, F., Berzish, M., Kulczynski, M., Nowotka, D., Ganesh, V., 2021. Z3str4: A multi-armed string solver. In: Huisman, M., Pasareanu, C.S., Zhan, N. (Eds.), Formal Methods - 24th International Symposium, FM 2021, Virtual Event, November 20-26, 2021, Proceedings. In: Lecture Notes in Computer Science, vol. 13047, Springer, pp. 389–406. http://dx.doi.org/10.1007/978-3-030-90870-6_21.

de Moura, L.M., Bjørner, N., 2008. Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings. In: Lecture Notes in Computer Science, vol. 4963, Springer, pp. 337–340. http://dx.doi.org/10.1007/978-3-540-78800-3_24.

Neider, D., Jansen, N., 2013. Regular model checking using solver technologies and automata learning. In: Brat, G., Rungta, N., Venet, A. (Eds.), NASA Formal Methods, 5th International Symposium, NFM 2013, Moffett Field, CA, USA, May 14-16, 2013. Proceedings. In: Lecture Notes in Computer Science, vol. 7871, Springer, pp. 16–31. http://dx.doi.org/10.1007/978-3-642-38088-4_2.

Nielsen, J., 1917. Die isomorphismen der allgemeinen, unendlichen Gruppe mit zwei Erzeugenden. Math. Ann. 78 (1), 385–397. http://dx.doi.org/10.1007/BF01457113.

Nötzli, A., Reynolds, A., Barbosa, H., Barrett, C.W., Tinelli, C., 2022. Even faster conflicts and lazier reductions for string solvers. In: Shoham, S., Vizel, Y. (Eds.), Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part II. In: Lecture Notes in Computer Science, vol. 13372, Springer, pp. 205–226. http://dx.doi.org/10.1007/978-3-031-13188-2_11.

Osera, P.-M., 2019. Constraint-based type-directed program synthesis. In: Proceedings of the 4th ACM SIGPLAN International Workshop on Type-Driven Development. In: TyDe 2019, Association for Computing Machinery, New York, NY, USA, pp. 64–76. http://dx.doi.org/10.1145/3331554.3342608.

Pin, J. (Ed.), 2021. Handbook of Automata Theory. European Mathematical Society Publishing House, Zürich, Switzerland, http://dx.doi.org/10.4171/Automata.

Plandowski, W., 1999. Satisfiability of word equations with constants is in PSPACE. In: 40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039). IEEE, pp. 495–500.

Plandowski, W., 2006. An efficient algorithm for solving word equations. In: Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing. ACM, pp. 467–476.

Presburger, M., 1929. About the completeness of a certain system of integer arithmetic in which addition is the only operation. In: I Congrès de Mathématiciens des Pays Slaves. pp. 92–101.

Quine, W.V., 1946. Concatenation as a basis for arithmetic. J. Symb. Log. 11 (4), 105–114.

Reynolds, A., Nötzli, A., Barrett, C., Tinelli, C., 2019. High-level abstractions for simplifying extended string constraints in SMT. In: International Conference on Computer Aided Verification. Springer, pp. 23–42.

Reynolds, A., Woo, M., Barrett, C., Brumley, D., Liang, T., Tinelli, C., 2017. Scaling up DPLL (T) string solvers using context-dependent simplification. In: International Conference on Computer Aided Verification. Springer, pp. 453–474.

Robson, J.M., Diekert, V., 1999. On quadratic word equations. In: Annual Symposium on Theoretical Aspects of Computer Science. Springer, pp. 217–226.

Schulz, K.U., 1990. Makanin's algorithm for word equations-two improvements and a generalization. In: International Workshop on Word Equations and Related Topics. Springer, pp. 85–150.

Scott, J.D., Flener, P., Pearson, J., Schulte, C., 2017. Design and implementation of bounded-length sequence variables. In: Salvagnin, D., Lombardi, M. (Eds.), Integration of AI and OR Techniques in Constraint Programming - 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings. In: Lecture Notes in Computer Science, vol. 10335, Springer, pp. 51–67. http://dx.doi.org/10.1007/978-3-319-59776-8_5.

Sen, K., Kalasapur, S., Brutch, T., Gibbs, S., 2013. Jalangi: A selective record-replay and dynamic analysis framework for JavaScript. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. In: ESEC/FSE 2013, Association for Computing Machinery, pp. 488–498. http://dx.doi.org/10.1145/2491411.2491447.

Stanford, C., Veanes, M., Bjørner, N., 2021. Symbolic boolean derivatives for efficiently solving extended regular expression constraints. In: Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. pp. 620–635.

Thatcher, J.W., Wright, J.B., 1968. Generalized finite automata theory with an application to a decision problem of second-order logic. Math. Syst. Theory 2 (1), 57–81. http://dx.doi.org/10.1007/BF01691346.

Trinh, M.-T., Chu, D.-H., Jaffar, J., 2014. S3: A symbolic string solver for vulnerability detection in web applications. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. ACM, pp. 1232–1243.

Trinh, M.-T., Chu, D.-H., Jaffar, J., 2016. Progressive reasoning over recursively-defined strings. In: International Conference on Computer Aided Verification. Springer, pp. 218–240.

Trinh, M.-T., Chu, D.-H., Jaffar, J., 2020. Inter-theory dependency analysis for SMT string solvers. Proc. ACM Program. Lang. 4 (OOPSLA), 1–27.

Tseitin, G.S., 1983. On the complexity of derivation in propositional calculus. In: Automation of Reasoning. Springer, pp. 466–483.

Veanes, M., Hooimeijer, P., Livshits, B., Molnar, D., Bjørner, N.S., 2012. Symbolic finite state transducers: algorithms and applications. In: Field, J., Hicks, M. (Eds.), Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012. ACM, pp. 137–150. http://dx.doi.org/10.1145/2103656.2103674.

Wang, H., Chen, S., Yu, F., Jiang, J.R., 2018. A symbolic model checking approach to the analysis of string and length constraints. In: Huchard, M., Kästner, C., Fraser, G. (Eds.), Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018. ACM, pp. 623–633. http://dx.doi.org/10.1145/3238147.3238189.

Wang, H., Tsai, T., Lin, C., Yu, F., Jiang, J.R., 2016. String analysis via automata manipulation with logic circuit representation. In: Computer Aided Verification - 28th International Conference, CAV 2016. In: Lecture Notes in Computer Science, vol. 9779, Springer, pp. 241–260. http://dx.doi.org/10.1007/978-3-319-41528-4_13.

Wang, Y., Zhou, M., Jiang, Y., Song, X., Gu, M., Sun, J., 2017. A static analysis tool with optimizations for reachability determination. In: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering. ASE, pp. 925–930. http://dx.doi.org/10.1109/ASE.2017.8115706.

Wolper, P., Boigelot, B., 1998. Verifying systems with infinite but regular state spaces. In: Hu, A.J., Vardi, M.Y. (Eds.), Computer Aided Verification, 10th International Conference, CAV '98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings. In: Lecture Notes in Computer Science, vol. 1427, Springer, pp. 88–97. http://dx.doi.org/10.1007/BFb0028736.

Wolper, P., Boigelot, B., 2000. On the construction of automata from linear arithmetic constraints. In: Graf, S., Schwartzbach, M.I. (Eds.), Tools and Algorithms for Construction and Analysis of Systems, 6th International Conference, TACAS 2000, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS 2000, Berlin, Germany, March 25 - April 2, 2000, Proceedings. In: Lecture Notes in Computer Science, vol. 1785, Springer, pp. 1–19. http://dx.doi.org/10.1007/3-540-46419-0_1.

Wolper, P., Boigelot, B., 2002. Representing arithmetic constraints with finite automata: An overview. In: Stuckey, P.J. (Ed.), Logic Programming, 18th International Conference, ICLP 2002, Copenhagen, Denmark, July 29 - August 1, 2002, Proceedings. In: Lecture Notes in Computer Science, vol. 2401, Springer, pp. 1–19. http://dx.doi.org/10.1007/3-540-45619-8_1.

Yu, F., Alkhalaf, M., Bultan, T., 2010. Stranger: An automata-based string analysis tool for php. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, pp. 154–157.

Yu, F., Alkhalaf, M., Bultan, T., Ibarra, O.H., 2014. Automata-based symbolic string analysis for vulnerability detection. Form. Methods Syst. Des. 44 (1), 44–70. http://dx.doi.org/10.1007/s10703-013-0189-1.

Yu, F., Shueh, C.-Y., Lin, C.-H., Chen, Y.-F., Wang, B.-Y., Bultan, T., 2016. Optimal sanitization synthesis for web application vulnerability repair. In: Proceedings of the 25th International Symposium on Software Testing and Analysis. ACM, pp. 189–200.

Zheng, Y., Ganesh, V., Subramanian, S., Tripp, O., Berzish, M., Dolby, J., Zhang, X., 2017. Z3str2: an efficient solver for strings, regular expressions, and length constraints. Form. Methods Syst. Des. 50 (2–3), 249–288.

Yu-Fang Chen is a research fellow at the Institute of Information Science, Academia Sinica. He has more than ten years of experience in the field of formal methods and verification, and has contributed to developing methods such as automata-based verification, algorithmic learning, and various types of decision procedures.

Vojtěch Havlena is a researcher at Faculty of Information Technology, Brno University of Technology, Czech Republic. His research interests include formal methods, automata theory, and logics. He particularly focuses on the development of automata-based techniques for the use in decision procedures of various theories and in security applications.

Ondřej Lengál is an assistant professor at Faculty of Information Technology, Brno University of Technology. His main interests lie in automata theory, logics, and their uses in developing efficient, safe, and secure computer systems. In particular, he works on analysis and verification of programs and techniques for efficient work with automata and their applications

Andrea Turrini received his Master and Ph.D. degree in computer science from University of Verona, Italy, in 2005 and 2009, respectively. He is currently an associate research professor at the Institute of Software, Chinese Academy of Sciences, Beijing. His research interests include the formal analysis of processes involving probability and nondeterminism, the development of efficient operations and transformations on omega regular automata, and the extension of automata-based frameworks to new scenarios.