A Decision Procedure for String Logic with Quadratic Equations, Regular Expressions and Length Constraints

Quang Loc Le and Mengda He

Teesside University, Middlesbrough, United Kingdom
Q.Le@tees.ac.uk

Abstract. In this work, we consider the satisfiability problem in a logic that combines word equations over string variables denoting words of unbounded lengths, regular languages to which words belong and Presburger constraints on the length of words. We present a novel decision procedure over two decidable fragments that include quadratic word equations (i.e., each string variable occurs at most twice). The proposed procedure reduces the problem to solving the satisfiability in the Presburger arithmetic. The procedure combines two main components: (i) an algorithm to derive a complete set of all solutions of conjunctions of word equations and regular expressions; and (ii) two methods to precisely compute relational constraints over string lengths implied by the set of all solutions. We have implemented a prototype tool and evaluated it over a set of satisfiability problems in the logic. The experimental results show that the tool is effective and efficient.

1 Introduction

The problem of solving word algebras has been studied since the early stage of mathematics and computer science [16]. Solving word equation (which includes concatenation operation, equalities and inequalities on string variables) was an intriguing problem and initially investigated due to its ties to Hilbert's 10th problem. The major result was obtained in 1977 by Makanin [37] who showed that the satisfiability of word equations with constants is, indeed, decidable. In recent years, due to considerable number of security threats over the Internet, there has been much renewed interest in the satisfiability problem involving the development of formal reasoning systems to either verify safety properties or to detect vulnerability for web and database applications. These applications often require a reasoning about string theories that combines word equations, regular languages and constraints on the length of words.

Providing a decision procedure for the satisfiability problem on a string logic including word equations and length constraints has been difficult to achieve. One main challenge is how to support an inductive reasoning about the combination of unbounded strings and the *infinite* integer domain. Indeed, the satisfiability of word equations combined with length constraints of the form $|x|{=}|y|$ is open [11,22] (where $|x|$ denotes the length of the string variable x). So far, very few decidability results in this logic are known; the most expressive result is restricted within the straight-line fragment (SL) which is based on *acyclic* word equations [22,7,36,12,23]. This SL fragment excludes constraints combining *quadratic* word equations, the equations in which each string variable occurs at most twice. For instance, the following constraint is beyond the SL fragment: $\mathtt{e}_c \equiv x \cdot a \cdot a \cdot y = y \cdot b \cdot a \cdot x$ where x and y are string variables, a and b are letters, and $\cdot$ is the string concatenation operation. Hence, one research goal is to identify decidable logics combining quadratic word equations (and beyond), based on which we can develop an efficient decision procedure.

There have been efforts to deal with the cyclic string constraints in Z3str2 [51,50], CVC4 [34] and S3P [48]. While Z3str2 presented a mechanism to detect overlapping variables to avoid non-termination, CVC4 proposed *refutation complete* procedure to generate a refutation for any unsatisfiable input problem and S3P [48] provided a method to identify and prune non-progressing scenarios. However, none is both complete and terminating over quadratic word equations. For instance, Z3str2, CVC4 and S3P (and all the state-of-the-art string solving techniques [7,8,6,9,12,23]) is not able to decide the satisfiability of the word equation $\mathtt{e}_c$ above.

In this work, we propose a novel cyclic proof system within a satisfiability procedure for the string theory combining word equations, regular memberships and Presburger constraints over the length functions. Moreover, we identify decidable fragments with quadratic word equations (e.g., the constraint $\mathtt{e}_c$ above) where the proposed procedure is complete and terminating. To the best of our knowledge, our proposal is the first decision procedure for string constraints beyond the straight-line word equations. Our proposal has two main components. First, we present a novel algorithm to construct a cyclic *reduction tree* which finitely represents all solutions of a conjunction of word equations and regular membership predicates. Secondly, we describe two procedures to infer the length constraints implied by the set of all solutions.

Contributions. We make the following technical contributions.

- We develop a algorithm, called ω-$\mathtt{SAT}$, to derive a cyclic reduction tree as a finite representation for all solutions of a conjunction of word equations and regular expressions. We show that if ω-$\mathtt{SAT}$ terminates with a reduction tree, the tree forms a finite-index $EDT0L$ system [41].
- We present a decision procedure, called $\mathtt{Kepler}_{22}$, with two decidable fragments and provide a complexity analysis of our approach. This is the first decidable result for the string theory combining *quadratic* word equations with length constraints.
- We have implemented a prototype solver and evaluated it over a set of hand-drafted benchmarks in the decidable fragments. The experimental results show that when compared with the state-of-the-art solvers, our proposal is both effective and efficient in solving string constraints with quadratic equations and length constraints.

Organization. The rest of the paper is organized as follows. Sect 2 presents relevant definitions. Sect 3 shows an overview of our approach through an example. We show how to compute a cyclic reduction tree to finitely represent all solutions of a conjunction of word equations and regular memberships in Sect 4. Sect 5 presents the proposed decision procedure. Sect 6 and Sect 7 describe the two decidable fragments. Sect 8 presents an implementation and evaluation. Sect 9 reviews related work and concludes.

2 Preliminaries

Concrete string models assume a finite alphabet Σ whose elements are called *letters*, set of finite words over Σ^* including ϵ - the empty word, and a set of integer numbers $\mathbb{Z}$. We

$$
\begin{array}{llll}
\text{disj formula} & \pi ::= \phi \mid \pi_1 \vee \pi_2 & \text{formula} & \phi \ ::= \ \mathsf{e} \mid \alpha \mid s{\in}\mathcal{R} \mid \neg\phi_1 \mid \phi_1 \wedge \phi_2 \\
\text{(dis)equality} & \mathsf{e} ::= s_1{=}s_2 & \text{term} & s \ ::= \ \epsilon \mid c \mid x \mid s_1 \cdot s_2 \\
\text{regex} & \multicolumn{3}{l}{\mathcal{R} ::= \emptyset \mid \epsilon \mid c \mid w \mid \mathcal{R}_1 \cdot \mathcal{R}_2 \mid \mathcal{R}_1 + \mathcal{R}_2 \mid \mathcal{R}_1 \cap \mathcal{R}_2 \mid \mathcal{R}_1^C \mid \mathcal{R}_1^*} \\
\text{Arithmetic} & \multicolumn{3}{l}{\alpha ::= a_1 = a_2 \mid a_1 > a_2 \mid \alpha_1 \wedge \alpha_2 \mid \alpha_1 \vee \alpha_2 \mid \exists v.\alpha_1} \\
 & \multicolumn{3}{l}{a ::= 0 \mid 1 \mid v \mid |u| \mid i \times a_1 \mid -a_1 \mid a_1 + a_2}
\end{array}
$$

Fig. 1: Syntax

work with a set U of string variables denoting words in Σ^*, and a set *I* of arithmetical variables. We use $|w|$ to denote the length of $w{\in}\Sigma^*$ and $\bar{v}$ a sequence of variables. A language L over the alphabet Σ is a set $L{\subseteq}\Sigma^*$. A language L is a set of words generated by a grammar system. We use $\mathcal{L}(L)$ to denote the class of all languages L.

Syntax The syntax of quantifier-free string formulas, called STR, is presented in Fig. 1. π is a disjunction formula where each disjunct ϕ is a conjunction of word equations e, arithmetic constraints α and regular memberships $s{\in}\mathcal{R}$. A word equation e is an equality of string terms s. (We use either s or tr to denote a string term.) A string term is a concatenation of the empty word ϵ, letters $c \in \Sigma$ and string variables x. We often write $s_1 s_2$ to denote $s_1 \cdot s_2$ if it is not ambiguous. Regular expression $\mathcal{R}$ over Σ is built over $c \in \Sigma$, $w \in \Sigma^*$, ϵ, and closing under union $+$, intersection $\cap$, complement C, concatenation $\cdot$, and the Kleene star operator $*$. Regular expressions $\mathcal{R}$ does not contain any string variables.

We use $\mathcal{E}$ to denote a conjunction (a.k.a system) of word equations. $\pi[t_1/t_2]$ denotes a substitution of all occurrences of t_2 in π to t_1. We use function $FV(\pi)$ to return all free variables of π. We inductively define length function of a string term s, denoted as $|s|$, as: $|\epsilon| = 0$, $|c| = 1$, and $|s_1 \cdot s_2| = |s_1| + |s_2|$. Notational length of the word equation e, denoted by $\mathsf{e}(N)$, is the number of its symbols.

A word equation is called *acyclic* if each variable occurs at most once. A word equation is called *quadratic* if each variable occurs at most twice. Similarly, a system of word equations is called quadratic if each variable occurs at most twice.

A word equation system is said to be straight-line [22,7,36] if it can be rewritten (by reordering the conjuncts) as the form $\bigwedge_{i=1}^{n} x_i = s_i$ such that: (i) x_1,...,x_n are different variables; and (ii) $FV(s_i) \subseteq \{x_1, x_2, .., x_{i-1}\}$. A formula $\pi \equiv \mathsf{e}_1{\wedge}\mathsf{e}_2{\wedge}...{\wedge}\mathsf{e}_n \wedge \Upsilon$ is called in straight-line fragment (SL) if $\mathsf{e}_1{\wedge}\mathsf{e}_2{\wedge}...{\wedge}\mathsf{e}_n$ is straight-line and the regular expression Υ is of the conjunction of regular memberships $x_j{\in}\mathcal{R}_j$ where $x_j{\in}\{x_1, ..., x_n\}$.

Semantics Every regular expression $\mathcal{R}$ is evaluated to the language $\mathcal{L}(\mathcal{R})$. We define:

$$
\mathit{SStacks} \stackrel{\text{def}}{=} (U{\cup}\Sigma){\rightarrow}\Sigma^* \qquad \mathit{ZStacks} \stackrel{\text{def}}{=} I \rightarrow \mathbb{Z}\,\cdot
$$

The semantics is given by a satisfaction relation: $\eta,\beta_\eta{\models}\pi$ that forces the interpretation on both string η and arithmetic β_η to satisfy the constraint π where $\eta \in \mathit{SStacks}$, $\beta_\eta{\in}\mathit{ZStacks}$, and π is a formula. We remark that $\forall\eta \in \mathit{SStacks}$: $\eta(c){=}c$ for all $c \in \Sigma$ and

$$
\begin{array}{lll}
\eta,\beta_\eta \models \pi_1 \vee \pi_2 & \texttt{iff} & \eta,\beta_\eta \models \pi_1 \text{ or } \eta,\beta_\eta \models \pi_2 \\
\eta,\beta_\eta \models \pi_1 \wedge \pi_2 & \texttt{iff} & \eta,\beta_\eta \models \pi_1 \text{ and } \eta,\beta_\eta \models \pi_2 \\
\eta,\beta_\eta \models \neg\pi_1 & \texttt{iff} & \eta,\beta_\eta \not\models \pi_1 \\
\eta,\beta_\eta \models s \in \mathcal{R} & \texttt{iff} & \exists w \in \mathcal{L}(\mathcal{R}) \cdot \eta,\beta_\eta \models s = w \\
\eta,\beta_\eta \models s_1 = s_2 & \texttt{iff} & \eta(s_1) = \eta(s_2) \text{ and } \beta_\eta(s_1) = \beta_\eta(s_2) \\
\eta,\beta_\eta \models s_1 \neq s_2 & \texttt{iff} & \eta,\beta_\eta \models \neg(s_1 = s_2) \\
\eta,\beta_\eta \models a_1 \oslash a_2 & \texttt{iff} & \eta(a_1) \oslash \eta(a_2), \text{ where } \oslash \in \{=, \leq\}
\end{array}
$$

Fig. 2: Semantics

$\eta(t_1 t_2)=\eta(t_1)\eta(t_2)$. The semantics of our language is formalized in Figure 2. If $\eta,\beta_\eta \models \pi$, we use the pair $\langle\eta,\beta_\eta\rangle$ to denote a solution of the formula π. Let $\mathtt{e} \equiv x_1 \cdot \ldots \cdot x_l = x_{l+1} \cdot \ldots \cdot x_n$ be a word equation. If $\mathtt{e}$ is satisfied with the solution $\langle\eta,\beta_\eta\rangle$, we also refer $\eta(x_1) \cdot \ldots \cdot \eta(x_l)$ as a solution word of $\mathtt{e}$. A solution word is minimal if the length of the solution word $(|\eta(x_1)| + \ldots + |\eta(x_l)|)$ is minimal. $\mathtt{e}_1$ is referred as a suffix of $\mathtt{e}_2$ if they are satisfied and the solution word of $\mathtt{e}_1$ is a suffix of the solution word of $\mathtt{e}_2$.

Formal Language A deterministic finite automaton (DFA) A is a tuple: $A=\langle Q,\Sigma,\delta,q_o,Q_F\rangle$, where Q is a finite set of states, $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$ is a finite set of transitions, $q_0 \in Q$ is the initial state and $Q_F \subseteq Q$ is a set of accepting states. We use $\mathcal{L}(A)$ to denote the (regular) language generated by a DFA A. It is known that the languages generated by regular expressions are also in the class of regular languages [26].

A context-free grammar (CFG) G is defined by the quadruple: $G=\langle V, \Sigma, P, S\rangle$ where V is a finite nonempty set of nonterminals, Σ is a finite set of terminals and disjoint from V, and $P \subseteq V \times (V \cup \Sigma)^*$ is a finite relation. For any strings $u, v \in (V \cup \Sigma)^*$, v is a result of applying the rule (α, β) to u $u \Rightarrow_G v$ if $\exists(\alpha,\beta) \in P$ $u_1, u_2 \in (V \cup \Sigma)^*$ such that $u = u_1 \alpha u_2$ and $v = u_1 \beta u_2$. $\mathcal{L}(G)=\{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$ to denote a language produced by the CFG G. Given a CFG $G=\langle V, \Sigma, P, S\rangle$, we use G_X (where $X \in V$) to denote a sub-language of $\mathcal{L}(G)$, defined by $\mathcal{L}(G_X)=\{w \in \Sigma^* \mid X \Rightarrow_G^* w\}$.

Normal Form $\pi \equiv \mathcal{E} \wedge \Upsilon \wedge \alpha$ is called in the normal form if it is of the form: $\mathcal{E}$ is a system of word equations, Υ is a conjunction of regular memberships (e.g., $X \in \mathcal{R}$) and α is a Presburger formula. (For the transformation of a formula presented in Fig. 1 into the normal form, [29,15] described how to eliminate negation over word equations, and disjunction of word equations and [7] showed how to remove the negation and the concatenation operator over regular expressions.)

Problem Definition Throughout this work, we consider the following problem.

PROBLEM:	SAT−STR.
INPUT:	A string constraint π in normal form over Σ.
QUESTION:	Is π satisfiable?

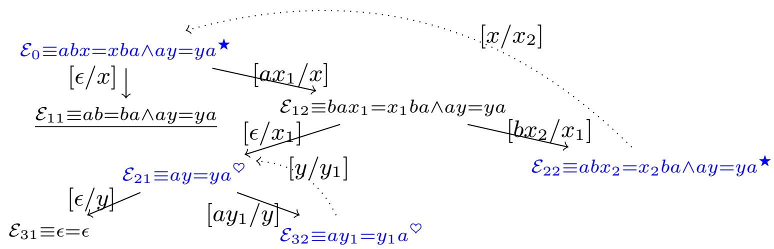

Fig. 3: Reduction Tree $\mathcal{T}_3$.

3 Overview and Illustration

The overall of our idea is an algorithm to reduce an input constraint to a set of solvable constraints. In this section, we first define the reduction tree (subsection 3.1). After that, we illustrate the proposed decision procedure through an example (subsection 3.2).

3.1 Cyclic Reduction Tree

Formally, a cyclic reduction tree $\mathcal{T}_i$ is a tuple $(V, E, \mathcal{C})$ where

- V is a finite set of nodes where each node represents a conjunction of word equations $\mathcal{E}$.
- E is a set of labeled and directed edges $(\mathcal{E}, \sigma, \mathcal{E}') \in E$ where $\mathcal{E}'$ is a child of $\mathcal{E}$. This edge means we can reduce $\mathcal{E}$ to $\mathcal{E}'$ via the label σ, a substitution, s.t.: $\mathcal{E}' \equiv \mathcal{E}\sigma$.
- And $\mathcal{C}$ is a back-link (partial) function which captures virtual cycles in the tree. A cycle, e.g. $\mathcal{C}(\mathcal{E}_c{\rightarrow}\mathcal{E}_b, \sigma)$, in $\mathcal{C}$ means the leaf $\mathcal{E}_b$ is linked back to its ancestor $\mathcal{E}_c$ and $\mathcal{E}_c \equiv \mathcal{E}_b\sigma$. In this back-link, $\mathcal{E}_b$ is referred as a *bud* and $\mathcal{E}_c$ is referred as a *companion*.

A path $(\mathtt{v}_s, \mathtt{v}_e)$ is a sequence of nodes and edges connecting node $\mathtt{v}_s$ with node $\mathtt{v}_e$. A leaf node is either unsatisfiable, or satisfiable or linked back to an interior node, or not-yet-reduced. If a leaf node is not-yet-reduced, it is marked as open. Otherwise, it is marked as closed. A trace of a tree is a sequence of edge labels of a path in the tree. We refer a trace as solution trace if it corresponds to a path $(\mathtt{v}_s, \mathtt{v}_e)$ where $\mathtt{v}_s$ is the root and $\mathtt{v}_e$ is a satisfiable leaf. This trace represents a (infinite) family solutions of the equation at the root.

3.2 Illustrative Example

We consider the following constraint: $\pi \equiv abx{=}xba{\wedge}ay{=}ya \wedge |x|{=}2|y|$ where x, y are string variables and a, b are letters. This constraint is beyond the straight-line fragment [22,7,36,12,23]. Moreover, as the length constraint $|x|{=}2|y|$ is not regular-based, the automata-based translation proposed in [12] cannot be applied.

The proposed solver $\mathtt{Kepler}_{22}$ could solve the constraint π above through the following three steps. First, it invokes procedure ω-$\mathtt{SAT}$ to construct a cyclic reduction tree to capture all solutions of the word equations $\mathcal{E}_0 \equiv abx{=}xba \wedge ay{=}ya$. Next, it infers a precise constraint α_{xy} implied by string lengths of all solutions. Lastly, it solves the conjunction: $\alpha_{xy}{\wedge}\alpha$ where α is the arithmetic constraint in the input π.

The representation of all solutions ω-SAT derives the reduction tree $\mathcal{T}_3$ (V, E, C), shown in Figure 3, as the finite presentation of all solutions for $\mathcal{E}_0$. In particular, the root of the tree is $\mathcal{E}_0$. $\mathcal{E}_0$ has two children $\mathcal{E}_{11}$ and $\mathcal{E}_{12}$, which are obtained by reducing x into two *complete* cases: $x{=}\epsilon$ and $x{=}ax_1$ where x_1 is fresh. Note that $\mathcal{E}_{12}$ is obtained by first applying the substitution: $\mathcal{E}'_{12} \equiv \mathcal{E}_0[ax_1/x] \equiv abax_1{=}ax_1ba{\wedge}ay{=}ya$ prior to subtracting the letter a at the heads of the two sides of the first word equation. Next, while $\mathcal{E}_{11}$ is classified as unsatisfiable, (underlined) and marked closed, $\mathcal{E}_{12}$ is further reduced into two children, $\mathcal{E}_{21}$ and $\mathcal{E}_{22}$. They are obtained by reducing x_1 at the head of the right-hand side (RHS) of $\mathcal{E}_{12}$ into two complete cases: $x_1{=}\epsilon$ to generate $\mathcal{E}'_{21} \equiv \mathcal{E}'_{12}[\epsilon/x_1] \equiv ab{=}ab{\wedge}ay{=}ya$ and $x_1{=}bx_2$ (where x_2 is a fresh variable) to generate $\mathcal{E}'_{22} \equiv \mathtt{e}'_{12}[bx_2/x_1] \equiv babx_2{=}bx_2ba$. Next, $\mathcal{E}'_{21}$ is further reduced into $\mathcal{E}_{21}$ by matching a, b letters; and $\mathcal{E}'_{22}$ is further reduced into $\mathcal{E}_{22}$ by matching b letters at the heads of its two sides. Lastly, $\mathcal{E}_{22}$ is linked back to $\mathcal{E}_0$ to form the back-link $\mathcal{C}(\mathcal{E}_0{\to}\mathcal{E}_{22}, [x/x_2])$. Similarly, $\mathcal{E}_{21}$ is reduced until all leaf nodes are marked closed.

A path $(\mathtt{v}_s, \mathtt{v}_e)$ with trace σ represents for $\mathtt{v}_e \equiv \mathtt{v}_s\sigma$. If $\mathtt{v}_e$ is satisfiable, then σ represents for a family of solutions (or valid assignments). For instance, in Fig. 3, the path $(\mathcal{E}_0, \mathcal{E}_{31})$ has the trace $\sigma_{31}{=}[ax_1/x, \epsilon/x_1, \epsilon/y]$. As $\mathcal{E}_{31}$ is satisfiable, we can derive a solution of $\mathcal{E}_0$ based on σ_{31} as: $x{=}a$ and $y{=}\epsilon$. Moreover, trace solution that is involved in cycles represents a set of infinite solutions, since we can construct infinitely many solution traces by iterating through the cycles an unbounded number of times. For example, all solution traces σ_{ij} obtained from the path $(\mathcal{E}_0, \mathcal{E}_{31})$ above is as:

$$\sigma_{ij} \equiv [ax_1/x] \circ [bx_2/x_1, x/x_2, ax_1/x]^i \circ [ay_1/y, y_1/y]^j \circ [\epsilon/x_1 \circ \epsilon/y]$$

where $\circ$ is the substitution composition operation, σ^k means σ is repeatedly composed zero, one or more times, and $i{\geq}0$, $j{\geq}0$.

Computing α_{xy} constraint Based on the solution trace σ_{ij} above, $\mathtt{Kepler}_{22}$ first generates a conjunctive set of constrained Horn clauses to define the relational assumptions over lengths of x and y in the set of all solutions. After that it infers the length constraint as: $\alpha_{xy} \equiv \exists i.|x|{=}2i{+}1{\wedge}i{\geq}0{\wedge}|y|{\geq}0$. Now, the satisfiability of π is equi-satisfiable to the following formula: $\pi' \equiv (\exists i.\ |x|{=}2i{+}1{\wedge}i{\geq}0{\wedge}|y|{\geq}0) \wedge |x|{=}2|y|$. As π' is unsatisfiable, so is π.

4 The Representation of All Solutions

In this section, we first present procedure ω-SAT which constructs a cyclic reduction tree for a conjunction of word equations $\mathcal{E}$ (subsection 4.1). After that, we describe how to combine the tree with regular membership predicates Υ (subsection 4.2). Finally, we discuss the correctness in subsection 4.3.

4.1 Constructing Cyclic Reduction Tree

ω-SAT transforms a conjunction of word equations $\mathcal{E}$ into a cyclic reduction tree $\mathcal{T}_n$ which represents all its solutions. This procedure starts with the tree $\mathcal{T}_0$ with only the input $\mathcal{E}$ at the root. After that, in each iteration it chooses one leaf node to reduce (using function `reduce`) or to make a back-link (using function `link_back`) until every leaf node is either irreducible or linked back. A leaf node is irreducible if it either trivially true (i.e., $w_1{=}w_1 \wedge ... \wedge w_i{=}w_i$ where $w_1, ..., w_i {\in} \Sigma^*$) or trivially false (i.e., either it is of the form: $c_1 tr_1{=}c_2 tr_2 \wedge \mathcal{E}$ where c_1, c_2 are different letters or its over-approximation over the length functions is unsatisfiable). Function `reduce` takes a leaf node $\mathcal{E}_i$ as input and produces a set L_i each element of which is a pair of a node $\mathcal{E}_{i_j}$ and a corresponding substitution σ_j such that $\mathcal{E}_{i_j}{=}\mathcal{E}_i\sigma_j$. For each pair $(\mathcal{E}_{i_j}, \sigma_j){\in}L_i$, it adds an new open node $\mathcal{E}_{i_j}$ and a new edge $(\mathcal{E}_i, \sigma_j, \mathcal{E}_{i_j})$. As a result, `reduce` extends the current tree with the new nodes and new edges. In particular, function `reduce` is implemented as: $L_i{=}\bigcup\{\texttt{matchs}(\mathcal{E}_{i_j}) \mid \mathcal{E}_{i_j}{\in}\texttt{complete}(\mathcal{E}_i)\}$ where function `matchs` exhaustively matches and subtracts identical letters and string variables at the heads of left-hand side (LHS) and right-hand side (RHS) of each word equation using function `match`. In the following, we describe the details of the functions used by ω-SAT.

Matching `match(e)` matches two terms at the heads of LHS and RHS of `e` as follows.

$$\texttt{match}(u_1{\cdot}tr_1{=}u_2{\cdot}tr_2) = \begin{cases} \texttt{match}(tr_1{=}tr_2) & \text{if } u_1, u_2 \text{ are identical} \\ u_1{\cdot}tr_1{=}u_2{\cdot}tr_2 & \text{otherwise} \end{cases}$$

where u_1, u_2 are either letters or string variables.

Procedure `complete` The overall goal of our reduction is to transform every word equation, say $\texttt{e}{\equiv}u_1tr_1{=}u_2tr_2$ where $\mathcal{E}_i{=}\texttt{e}{\wedge}\mathcal{E}$, into a set of "smaller" string equation $\texttt{e}_i$ such that if `e` is satisfied, $\texttt{e}_i$ is a suffix of `e`. Word equations in a node are reduced in a depth-first manner. Intuitively, our reduction over the word equation `e` is based on the possible arrangements of two carrier terms, the terms at the heads of LHS and RHS of `e`. Suppose that `e` is satisfied. Let l_1, r_1 be the starting and ending positions of u_1 in the solution word of `e`. Similarly, let l_2, r_2 be the starting and ending positions of u_1 in the solution word of `e`. Obviously, $l_1{=}l_2$. Our reduction, function `complete`, considers all possible arrangements based on these positions. For arrangements in one-side (LHS or RHS), it considers the cases: $l_1{=}r_1$ (i.e., $u_1{=}\epsilon$), $l_1{<}r_1$ and $l_2{=}r_2$ (i.e., $u_2{=}\epsilon$), $l_2{<}r_2$. For arrangements between the two sides, it considers the cases: $r_1{\geq}r_2$ and $r_2{\geq}r_1$. In particular, function `complete` considers the following two scenarios of the carrier terms.
Case 1: One term is a letter and another term is a string variable, e.g. $x_1tr_1{=}c_2tr_2$. `complete` generates the set L_i as $L_i{\equiv}\{(\mathcal{E}_{i_1}, \sigma_1); (\mathcal{E}_{i_2}, \sigma_2)\}$ where

- 1a) $\sigma_1{=}[\epsilon/x_1]$
- 1b) $\sigma_2{=}[c_2x_1'/x_1]$, x_1' is a fresh variable and referred as a subterm of x_1.

Case 2: These terms are two different string variables, e.g. $x_1tr_1{=}x_2tr_2$. `complete` generates the set L_i as : $L_i{\equiv}\{(\mathcal{E}_{i_1}, \sigma_1); (\mathcal{E}_{i_2}, \sigma_2); (\mathcal{E}_{i_3}, \sigma_3); (\mathcal{E}_{i_4}, \sigma_4)\}$ where

- 2a) $\sigma_1{=}[\epsilon/x_1]$,

– 2b) σ_3=$[x_2x_1'/x_1]$, x_1' is a fresh variable and referred as a subterm of x_1,
– 2c) σ_2=$[\epsilon/x_2]$
– 2d) σ_4=$[x_1x_2'/x_2]$, x_2' is a fresh variable and referred as a subterm of x_2.

As both Case 2b and Case 2d include the scenario where $x_1{=}x_2$, the reduction tree generated represents a *complete* but *not minimal* set of all solution.

Linking back `link_back` links a leaf node $\mathcal{E}_b$ to an interior node $\mathcal{E}_c$ if after some substitution σ_{cyc}, two nodes are identical: $\mathcal{E}_c{\equiv}\mathcal{E}_b\sigma_{cyc}$. In addition, for every entry $X/X' \in \sigma_{cyc}$ where X and X' are string variables, X' is a subterm of X. σ_{cyc} can be considered as a permutation function on both U and the alphabet Σ. We recap that we refer to this cycle as a triple $\mathcal{C}(\mathcal{E}_c{\rightarrow}\mathcal{E}_b, \sigma_{cyc})$ where $\mathcal{E}_c$ is called a companion, $\mathcal{E}_b$ is called a bud.

4.2 Combining with regular memberships

We propose to derive a finite representation of all solutions of a conjunction of word equations and regular expressions. using procedure `widentree`. Procedure `widentree` takes a pair of a reduction tree $\mathcal{T}_n$ of $\mathcal{E}_0$ (generated by ω-SAT) and a conjunction of regular expressions Υ as inputs and manipulates the reduction tree $\mathcal{T}_n$ through the following three steps. First, it constructs a DFA $A{=}\langle Q,\Sigma,\delta,q_o,Q_F\rangle$ which generates the same language with Υ. Let m be the number states in Q and $M{=}m!$. Intuitively, $m{+}1$ is the minimal times of a cycle to obtain the minimal solutions of $\mathcal{E}_0{\wedge}\Upsilon$. M is the periodic of the sets of all solutions. Secondly, it unfolds every cycles $\mathcal{C}(\mathcal{E}_c{\rightarrow}\mathcal{E}_b, \sigma)$ of $\mathcal{T}_n$ $m{+}M$ times. It updates `link_back` functions by eliminating the old back-link between $\mathcal{E}_b$ and $\mathcal{E}_c$ prior to generating a new back-link between $\mathcal{E}_{b_{m+M}}$ and $\mathcal{E}_{c_m}$ as well as marking $\mathcal{E}_{b_{m+M}}$ as closed. We note that a solution corresponding to a trace which visits the companion $\mathcal{E}_{c_m}$ $l{+}1$ times (i.e., including k new cycles above) has the form: $S \equiv u_1w^{m+1+lM}u_2$. Lastly, it collects label σ_j for every path $(\mathcal{E}_0, \mathcal{E}_j)$ in the new tree where $\mathcal{E}_0$ is the root, $\mathcal{E}_j$ is a leaf node that is neither unsatisfiable nor a bud prior to evaluating $\mathcal{E}_j$. From σ_j, it generates the following formula: $\pi_j{\equiv}\bigwedge\{X_i{=}s_i|(s_i/X_i){\in}\sigma_j\}{\wedge}\Upsilon$. π_j is in a *straight-line* fragment where the satisfiability problem SAT-STR is decidable [36].

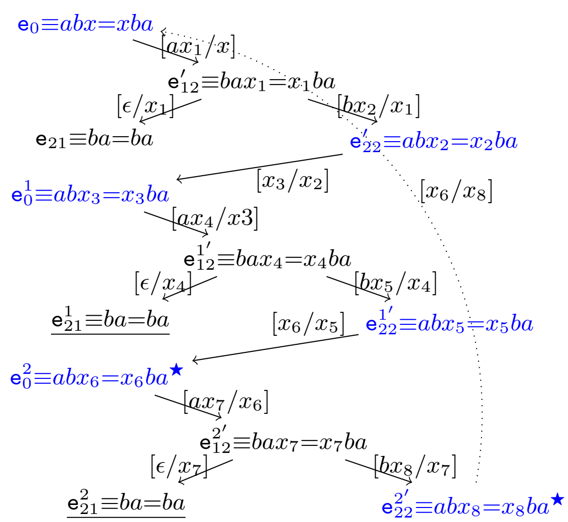

Fig. 4: Extending Tree $\mathcal{T}_2$ with $x \in a^*$.

Example 1. To illustrate our first decidable fragment, we use the following word equation as a running example: $abx{=}xba$ where x is string variable and a, b are letters. This

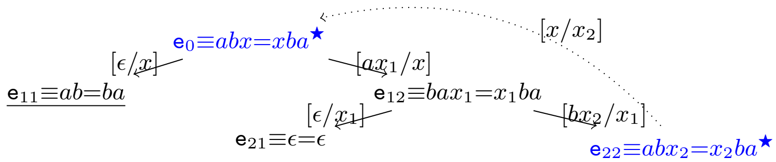

Fig. 5: Reduction Tree $\mathcal{T}_2$.

is the first equation in the motivating example (section 3.2). Its reduction tree $\mathcal{T}_2$ is presented in Fig. 5. We now illustrate how to use procedure `widentree` above to extend the tree to represent all solutions of $\pi_1{\equiv}abx{=}xba \wedge x{\in}a^*$. To do that, `widentree` first derives for the regular expression $x \in a^*$ a DFA as: $A = \langle\{q_0\},\{a\},\{((q_0,a),a)\},q_0,\{q_0\}\rangle$, and then identifies $m{=}1$ and $M{=}m!{=}1$. Secondly, it clones the cycle of $\mathcal{T}_2$ $m + M = 1 + 1 = 2$ more times. The resulting tree is described in Fig. 4. Lastly, it discharges the satisfiability of solutions corresponding to the paths which start from the root and end at leaf nodes $\mathtt{e}_{21}$, $\mathtt{e}^1_{21}$ or $\mathtt{e}^2_{21}$. The evaluation is as follows.

path	formula	outcome
$(\mathtt{e}_0, \mathtt{e}_{21})$	$x{=}ax_1{\wedge}x_1{=}\epsilon \wedge x{\in}a^*$	SAT
$(\mathtt{e}_0, \mathtt{e}^1_{21})$	$x = ax_1{\wedge}x_1{=}bx_2{\wedge}x_2{=}x_3 \wedge x_3{=}ax_4{\wedge}x_4{=}\epsilon \wedge x{\in}a^*$	UNSAT
$(\mathtt{e}_0, \mathtt{e}^2_{21})$	$x = ax_1{\wedge}x_1{=}bx_2{\wedge}x_2{=}x_3 \wedge x_3{=}ax_4{\wedge}x_4{=}bx_5{\wedge}$ $x_5{=}x_6{\wedge}x_6{=}ax_7{\wedge}x_7{=}\epsilon \wedge x{\in}a^*$	UNSAT

4.3 Correctness

In the following, we formalize the correctness of the proposed procedures and show the relationship between the derived reduction tree with $EDT0L$ system [41].

Proposition 1. *Suppose that ω-*SAT *takes a conjunction $\mathcal{E}$ as input, and produces a cyclic reduction graph $\mathcal{T}_n$ in a finite time. Then, $\mathcal{T}_n$ represents all solutions of $\mathcal{E}$.*

Proposition 2. *Suppose that $\Upsilon \equiv X_1{\in}\mathcal{R}_1{\wedge}...{\wedge}X_n{\in}\mathcal{R}_n$ $(X_i{\in}FV(\mathcal{E}_0), \forall 1 \leq i \leq n)$ is a conjunction of regular memberships and $\mathcal{T}_n$ be the reduction tree derived for $\mathcal{E}_0$. Then,* `widentree`$(\mathcal{T}_n, \Upsilon)$ *produces a reduction tree representing all solutions of $\mathcal{E}_0{\wedge}\Upsilon$.*

An *interactionless Lindenmayer system* (0L system) [41] is a parallel rewriting system which was introduced in 1968 to model the development of multicellular system. The class of $EDT0L$ languages forms perhaps the central class in the theory of L systems. The acronym EDT0L refers to **E**xtended, **D**eterministic, **T**able, **0** interaction, and **L**indenmayer. In the following, we give a formal definition of $EDT0L$ system.

Definition 1 *An $ET0L$ system is a quadruple $G{=}\langle V,\ \Sigma,\ \mathcal{P},\ S\rangle$ where V is a finite nonempty set of nonterminals (or variables), Σ is a finite set of terminals and disjoint from V, $S{\in}V$ is the start variable (or start symbol), $\mathcal{P}$ is a finite set each element of which (called a* table*) is a finite binary relation included in $V \times (V{\cup}\Sigma)^*$. It is assumed that $\forall P \in \mathcal{P}, \forall x{\in}V, \exists tr{\in}(V \cup \Sigma)^*$ such that $(x,tr) \in P$. An $EDT0L$ system is a* deterministic *$ET0L$ system in which $\forall P{\in}\mathcal{P}, \forall x \in V, \exists! tr{\in}(V \cup \Sigma)^*$ s.t. $(x,tr) \in P$.*

For a production (x,tr) of P in $\mathcal{P}$, we often write: $x \rightarrow tr$. We also write $x \rightarrow_P tr$ for "$x \rightarrow tr$ is in $\mathcal{P}$". Let G=$\langle V,\, \Sigma,\, \mathcal{P},\, S\rangle$ be an $ET0L$ system.

1. Let x,$y \in (V \cup \Sigma)^*$, and x contains k nonterminals v_1,..., v_k in V. We say that x directly derives y (in G), denoted as $x \Rightarrow_G y$, if there is a $P \in \mathcal{P}$ such that y is obtained by substituting v_i by s_i, respectively for all $i \in \{1, ..., k\}$, where $v_1 \rightarrow_P s_1$, ..., $v_k \rightarrow_P s_k$. In this case, we also write $x \Rightarrow_P y$.
2. Let $\Rightarrow_G^*$ be the reflexive transitive closure of the relation $\Rightarrow$. If $x \Rightarrow_G^* y$ then we say that x derives y (in G).
3. The language of G, denoted by $\mathcal{L}(G)$, defined by $\mathcal{L}(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$.

A grammar system that is *k-index* is restricted so that, for every word generated by the grammar, there is some successful derivation where at most *k* nonterminals appear in every sentential form of the derivation [42]. A system is finite-index if it is k-index for some k. We use $\mathcal{L}(L)_{FIN}$ to denote the class of all L languages of finite-index.

Corollary 4.1 *A reduction tree derived by* ω-SAT *forms a finite-index* $EDT0L$ *system.*

Example 2. The tree in the Fig. 5 above forms the following finite-index $EDT0L$. G=$\langle\{S, x, x_1, x_2\},\ \Sigma,\ \{P_1, P_2\},\ S\rangle$ where $P_1 = \{(S, abx), (x, ax_1), (x_1, \epsilon)\}$ and $P_2 = \{(S, abx), (x, ax_1), (x_1, bx_2), (x_2, x)\}$.

5 Decision Procedure

We present decision procedure Kepler$_{22}$ to handle SAT−STR. Kepler$_{22}$ takes a constraint, say $\mathcal{E} \wedge \Upsilon \wedge \alpha$, as input and returns SAT or UNSAT. It works as follows. First, it invokes ω-SAT to construct a reduction tree $\mathcal{T}_n$ as a finite representation of all solutions of $\mathcal{E}$. After that, $\mathcal{T}_n$ is post-processed using procedure postpro as below to explicate all free variables. This step is critical to the next step. Secondly, it uses procedure widentree to extend $\mathcal{T}_n$ with membership predicates Υ and obtains $\mathcal{T}_{n+1}$. Note that unsatisfiable nodes in the reduction tree are eliminated. Thirdly, it computes the length constraints which are precisely implied by all solutions generated through procedure $extract\_pres(\mathcal{T}_{n+1})$.These length constrains, say α_w, are computed as an existentially quantified Presburger formula. Lastly, Kepler$_{22}$ solves that satisfiability of the conjunction $\alpha_w \wedge \alpha$ which is in the Presburger arithmetic and decidable [21].

Decision Procedure: Kepler$_{22}$($\mathcal{E} \wedge \Upsilon \wedge \alpha$)

$\mathcal{T}_n \leftarrow$ postprotrim(ω−SAT($\mathcal{E}$));
if (is_false($\mathcal{T}_n$)) **return** UNSAT;
$\mathcal{T}_{n+1} \leftarrow$ widentree($\mathcal{T}_n, \Upsilon$)
if (is_false($\mathcal{T}_{n+1}$)) **return** UNSAT;
$\alpha_w \leftarrow extract\_pres(\mathcal{T}_{n+1})$;
return SAT$_{\texttt{pres}}$($\alpha_w \wedge \alpha$);

Fig. 6: Satisfiability Solving.

Post-Processing Given a path from the root e_0 to a satisfiable leaf node e_i, a variable x appearing in this path is called *free* if it has not been reduced yet. This means x can be assigned any value in Σ^* in a solution. Procedure postpro aims to replace a free variable by a sub-tree which represents for arbitrary values in Σ^*. The sub-tree is presented in Fig. 7. This tree has a *base* leaf node (with substitution $[\epsilon/x]$) and k

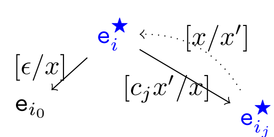

Fig. 7: Free Variable x.

cycles (k is the size of the alphabet Σ) one of which represents for a letter $c_i \in \Sigma$. If a satisfiable leaf node has more than one free variable, each variable is replaced by such sub-tree and these sub-trees are connected together at base nodes.

Correctness The correctness of step 1 and step 2 have been shown in the previous section. Thus, the remaining tasks to show $\texttt{Kepler}_{22}$ is a decision procedure in a fragment are the termination of ω-SAT as well as the decidability of $extract\_pres(\mathcal{T}_{n+1})$.

6 $\texttt{STR}_{\texttt{EDT0L}}$ Decidable Fragment

Computing length constraint in this fragment is based on Parikh's Theorem [38], one of the most celebrated theorem in automata theory. The Parikh image (a.k.a. letter-counts) of a word over a given alphabet counts the number of occurrences of each symbol in the word without regard to their order. The Parikh image of a language is the set of Parikh images of the words in the language. A language is Parikh-definable if its Parikh image precisely coincides with semilinear sets which, in turn, can be computed as a Presburger formula. In particular, Parikh's Theorem [38] states that context-free languages (and regular languages, of course) are Parikh-definable. In fact, given a context-free grammar, we can compute its Parikh image in polynomial time [49,19]. Moreover, the authors in [42] show that finite-index EDT0L languages [41] are also Parikh-definable. In our work, we use $Par(L)$ to denote the Parikh images computed for the language L.

A given constraint, say $\mathcal{E}{\wedge}\Upsilon{\wedge}\pi$, is said to be in the fragment if the following two conditions hold. First, ω-SAT terminates on $\mathcal{E}$. Secondly, $\pi \equiv \alpha_1{\wedge}..{\wedge}\alpha_n$ where $FV(\alpha_i)$ contains at most one string length $\forall i \in \{1...n\}$. By the first condition, $\texttt{Kepler}_{22}$ can derive for $\mathcal{E}$ a finite-index $EDT0L$ system (Corollary 4.1). Moreover, finite-index $EDT0L$ can be translated into a Parikh-equivalent DFA (by Parikh's Theorem [38,42]). This means length of each string variable in the set of all solutions can be computed as a DFA. By the second condition, each constraint α_1 is based on the length of one string variable. Hence, this constraint can be translated into another DFA. As regular languages are closed under intersection. Therefore, the satisfiability of π is decidable.

$\texttt{Kepler}_{22}$ uses $extract\_pres(\mathcal{T}_{n+1})$ to compute the length constraints represented for all solutions of $\mathcal{E}{\wedge}\Upsilon$ as follows. Firstly, it transforms $\mathcal{T}_{n+1}$ into a finite-index $EDT0L$ system. Secondly, it transforms the $EDT0L$ grammar into a Parikh-equivalent CFG G (see [42]). Lastly, it computes the length constraints α_w for every string variables as: $\alpha_w {\equiv} \bigwedge\{Par(\mathcal{L}(G_x)) \mid x \in FV(\mathcal{E}{\wedge}\Upsilon)\}$.

6.1 Parikh Image of CFG

In order to infer the Parikh image for a given CFG, we first transform the CFG into a Parikh equivalent communication-free Petri net and then compute the Parikh image of the communication-free Petri net [49]. The correctness was presented in [18,45,49]. Procedure Par takes a CFG $G{=}\langle V,\ \Sigma,\ P,\ s_0\rangle$ as input and produces a Presburger formula to represents the Parikh image of all words derived from the start symbol s_0. In particular, it first transforms the CFG into a communication-free Petri net and then generates a Presburger formula α_G for this net.

A net N is a quadruple $N{=}\langle S,\, T,\, W,\, s_0\rangle$ where S is a set of places, T is a set of transitions, W is a weight function: $(S \times T) \cup (T \times S) \rightarrow \mathbb{N}$, and s_0 is the start place in the net. If $W(x,y){>}0$, there is an edge from x to y of weight $W(x,y)$. A net is communication-free if for each transition t there is at most one place s with $W(s,t) > 0$ and furthermore $W(s,t) = 1$. A marking M, a function $S \rightarrow \mathbb{N}$, associates a number of tokens with each place. A communication-free Petri net is a pair (N, M) where N is a communication-free net and M is a marking.

The CFG G is transformed into a communication-free Petri net (N_G, M_G) as: $N_G{=}\langle V{\cup}\Sigma,\, P,\, W,\, s_0\rangle$. If $A{\rightarrow}s$ is a production $p \in P$ then $W(A,p){=}1$ and $W(B,p)$ is the number occurrences of B in s, for each $B{\in}V{\cup}\Sigma$. Finally, $M_G(s_0){=}1$ and $M_G(X){=}0$ for all other $X{\in}V{\cup}\Sigma$ and $X{\neq}s_0$. Let x_c be a new integer variable for each letter $c{\in}\Sigma$, y_p be a new integer variable for each rule $p{\in}P$, and z_s be a new integer variable for each symbol $s{\in}V \cup \Sigma$. We assume that we have m variables $y_{p_1},..,y_{p_m}$ and n variables $z_{s_1},..,z_{s_n}$. We note that x_c is used to count the number occurrences of the letter $c{\in}\Sigma$ in a word derived by the grammar G. The output α_G is generated through the following two steps. Firstly, the procedure generates a quantifier-free Presburger formula α_{count} which constrains the occurrences of letters in words derived by the grammar G. In particular, α_{count} is a conjunction of the four following kinds of subformulas.

- $x_c{\geq}0$ for all $c{\in}\Sigma$.
- For each $X{\in}V$, let p_1, ..., p_k be all productions which X is on the left-hand side. And we recap $W(X,p)$ denotes the number occurrences of X on the right-hand side of the production rule p. Then, α_{count} contains the following conjunct:

$$M_G(X) + \Sigma_{p\in P} W(X,p) y_p - \Sigma_{i=1}^{k} y_{p_i} = 0$$

- For each $c \in \Sigma$, α_{count} contains the following conjuncts:

$$x_c = \Sigma_{p\in P} W(c,p) y_p \wedge (x_c = 0 \vee z_c > 0)$$

- For each $s \in V \cup \Sigma$, let p_1,...,p_l be the productions where s is on the right-hand side and X_1,...X_l are their corresponding left-hand sides. Then, α_{count} contains the following conjunct: $(z_s{=}0 \vee \bigvee_{i=1}^{l}(z_s = z_{X_i}{+}1{\wedge}y_{p_i}{>}0{\wedge}z_{X_i}{>}0)$. If one of the X_i is the start symbol s_0, the corresponding disjunct is replaced by $z_s{=}1{\wedge}y_{p_i}{>}0$.

Secondly, α_G is generated as: $\alpha_G \equiv \exists y_{p_1},..,y_{p_m}, z_{s_1},..,z_{s_n}.|s_0|{=}\Sigma_{c\in\Sigma} x_c {\wedge} \alpha_{count}$.

Example 3. For the $EDT0L$ in Ex. 2, we generate the following Parikh-equivalent CFG G_1 $\langle V_1,\, \Sigma,\, P_1,\, S_1\rangle$ where the start symbol S_1 is fresh, $V_1{=}\{S_1, x, x_1, x_2, x_3\}$ and $P_1{\equiv}\{(S_1, abx), (x, ax_1), (x_1, bx_2), (x_2, x), (x, x_3), (x_3, ax_1), (x_1, \epsilon)\}$.

Next, we show how to compute $Par(\mathcal{L}(G_{1_x}))$, Parikh image of CFG G_{1_x}. Let x_a and x_b be integer variables which count the occurrences of letters a and b, resp., of every word. Let y_1, y_2,..., y_7 be integer variables representing for the each production in P_1 following the left-right order. And let z_a, z_b, z_{S_1}, z_x, z_{x_1}, z_{x_2} and z_{x_3} be integer variables which reflect the distance of the corresponding symbols to the start symbol x in a spanning tree on the subgraph of the transformed net induced by those p with $y_p{>}0$. The first kind of conjuncts in α_{count} is: $x_a{\geq}0{\wedge}x_b{\geq}0$. The second is:

Variable	$conjunct$
x	$1+(y_4+y_1)-(y_2+y_5)=0$
S_1	$0+0-y_1=0$
x_1	$0+(y_2+y_6)-(y_3+y_7)=0$

Variable	$conjunct$
x_2	$0+y_3-y_4=0$
x_3	$0+y_5-y_6=0$

The third kind of conjuncts in α_{count} corresponding to letter a and b is: $x_a{=}y_1{+}y_2{+}y_6\wedge(x_a{=}0\vee z_a{>}0)$ and $x_b{=}y_1{+}y_3\wedge(x_b{=}0\vee z_b{>}0)$, respectively. The fourth is as follows.

$x\quad z_x=0\vee(z_x=z_{x_2}+1\wedge y_4>0\wedge z_{x_2}>0)\vee(z_x=z_{S_1}+1\wedge y_1>0\wedge z_{S_1}>0)$
$S_1\ z_{S_1}=0$
$x_1\ z_{x_1}>0\vee(z_{x_1}=1\wedge y_2>0)\vee(z_{x_1}=z_{x_3}+1\wedge y_6>0\wedge z_{x_3}>0)$
$x_2\ z_{x_2}>0\vee(z_{x_2}=z_{x_1}+1\wedge y_3>0\wedge z_{x_1}>0)$
$x_3\ z_{x_3}>0\vee(z_{x_3}=1\wedge y_5>0)$
$a\quad z_a{>}0\vee(z_a{=}z_{S_1}{+}1\wedge y_1{>}0\wedge z_{S_1}{>}0)\vee(z_a{=}1{\wedge}y_2{>}0)\vee(z_a{=}z_{x_3}{+}1\wedge y_6{>}0\wedge z_a{>}0)$
$b\quad z_b>0\vee(z_b=z_S+1\wedge y_1>0\wedge z_{S_1}>0)\vee(z_a=z_{x_1}+1\wedge y_3>0\wedge z_a>0)$

Then, the length constraint of x is inferred as:

$$\begin{aligned}\alpha_{G_{1x}} &\equiv \exists y_1,..,y_7,z_a,z_b,z_x,z_{S_1},z_{x_1},z_{x_2},z_{x_3}.|x|{=}x_a{+}x_b\wedge\alpha_{count}\\ &\equiv \exists y_1,..,y_7,z_a,z_b,z_x,z_{S_1},z_{x_1},z_{x_2},z_{x_3}.|x|{=}2y_3{+}1\wedge x_a{=}y_3{+}1\wedge x_b{=}y_3\wedge\alpha_{count}\end{aligned}$$

6.2 $\mathtt{STR}_{\mathtt{EDT0L}}$: A Syntactic Decidable Fragment

Definition 2 ($\mathtt{STR}_{\mathtt{EDT0L}}$ Formulas) *$\mathcal{E}\wedge\Upsilon\wedge\alpha_1\wedge..\wedge\alpha_n$ is called in fragment $\mathtt{STR}_{\mathtt{EDT0L}}$ if $\mathcal{E}$ is a quadratic system and FV(α_i) contains at most one string length $\forall i\in\{1...n\}$.*

For example, $\mathtt{e}_c{\equiv}xaby{=}ybax$ is in $\mathtt{STR}_{\mathtt{EDT0L}}$. But $\pi\equiv abx{=}xba\wedge ay{=}ya\wedge|x|{=}2|y|$ (Sect. 3.2) is *not* in $\mathtt{STR}_{\mathtt{EDT0L}}$ as the arithmetic constraint includes two string lengths.

The decidability relies on the termination of ω-SAT over quadratic systems.

Proposition 3. *ω-SAT runs in factorial time in the worst case for quadratic systems.*

Let SAT-STR[$\mathtt{STR}_{\mathtt{EDT0L}}$] be the satisfiability problem in this fragment. The following theorem immediately follows from Proposition 3, Corollary 4.1, Parikh image of finite-index $EDT0L$ systems [42].

Theorem 1. *SAT-STR[$\mathtt{STR}_{\mathtt{EDT0L}}$] is decidable.*

7 $\mathtt{STR}_{\mathtt{flat}}$ Decidable Fragment

We first describe $\mathtt{STR}^{\mathtt{dec}}_{\mathtt{flat}}$ fragment through a semantic restriction and then show the computation of the length constraints. After that, we syntactically define $\mathtt{STR}_{\mathtt{flat}}$.

Definition 3 *The normalized formula $\mathcal{E}\wedge\Upsilon\wedge\alpha$ is called in the $\mathtt{STR}^{\mathtt{dec}}_{\mathtt{flat}}$ fragment if ω-SAT takes $\mathcal{E}$ as input, and produces a tree $\mathcal{T}_n$ in a finite time. Furthermore, for every cycle $\mathcal{C}(\mathcal{E}_c{\to}\mathcal{E}_b,\sigma_{cyc})$ of $\mathcal{T}_n$, every label along the path ($\mathcal{E}_c$, $\mathcal{E}_b$) is of the form: $[cY/X]$ where X, Y are string variables and c is a letter.*

This restriction implies that every node in a $\mathcal{T}_n$ belongs to *at most one cycle* and $\mathcal{T}_n$ does not contain any nested cycles. We refer such $\mathcal{T}_n$ as a *flat(able)* tree. It further implies that σ_{cyc} is of the form $\sigma_{cyc}\equiv[X_1/X_1',...,X_k/X_k']$ and X_j' is a (direct or indirect) subterm of X_j for all $j\in\{1...k\}$. We refer the variables X_j for all $j\in\{1...k\}$ as extensible variables and such cycle as $\mathcal{C}(\mathcal{E}_c{\to}\mathcal{E}_b,\sigma_{cyc})^{[X_1,...,X_k]}$.

Procedure $extract\_pres$ From a reduction tree, we propose to extract a system of inductive predicates which precisely capture the length constraints of string variables.

First, we extend the syntax of arithmetical constraints in Fig. 1 with inductive definitions as: $\alpha ::= a_1 = a_2 \mid a_1 > a_2 \mid \alpha_1 \wedge \alpha_2 \mid \alpha_1 \vee \alpha_2 \mid \exists v.\alpha_1 \mid \mathtt{P}(\bar{\mathbf{v}})$. In intuition, α may contain occurrences of predicates $\mathtt{P}(\bar{v})$ whose definitions are inductively defined. Inductive predicate is interpreted as a least fixed-point of values [46]. We notice that inductive predicates are restricted within arithmetic domain only. We assume that the system $\mathcal{P}$ includes n *unknown* (a.k.a. uninterpreted) predicates and $\mathcal{P}$ is defined by a set of constrained Horn clauses. Every clause is of the form: $\phi_{i_j} \Rightarrow \mathtt{P_i}(\bar{v}_i)$ where $\mathtt{P_i}(\bar{v}_i)$ is the head and ϕ_{i_j} is the body. A clause without head is called a query. A formula without any inductive predicate is referred as a *base* formula and denoted as ϕ^b. We now introduce Γ to denote an interpretation over unknown predicates such that for every $\mathtt{P_i} \in \mathcal{P}$, $\Gamma(\mathtt{P_i}(\bar{v}_i)) \equiv {\phi^b}_i$. We use $\phi(\Gamma)$ to denote a formula obtained by replacing all unknown predicates in ϕ with their definitions in Γ. We say a clause $\phi_b \Rightarrow \phi_h$ satisfies if there exists Γ and for all stacks $\eta{\in}\mathit{Stacks}$, we have $\eta \models \phi_b(\Gamma)$ implies $\eta \models \phi_h(\Gamma)$. A conjunctive set of Horn clauses (CHC for short), denoted by $\mathcal{R}$, is satisfied if every constraints in $\mathcal{R}$ is satisfied under the same interpretation of unknown predicates.

We maintain a one to one function that maps every string variable $x{\in}U$ to its respective length variable $n_x{\in}I$. We further distinguish *U* into two disjoint sets: *G* a set of global variables and *E* a set of local (existential) variables. While *G* includes those variables from the root of a reduction tree, *E* includes those fresh variables generated by ω-SAT. Given a tree $\mathcal{T}_{n+1}$ $(V, E, \mathcal{C})$ (where $\mathcal{E}_0{\in}V$ be the root of the tree) deduced from an input $\mathcal{E}_0{\wedge}\Upsilon$, we generate a system of inductive predicates and CHC $\mathcal{R}$ as follows.

1. For every node $\mathcal{E}_i{\in}V$ s.t. $\bar{v}_i{=}FV(\mathcal{E}_i){\neq}\emptyset$, we generate an inductive predicate $\mathtt{P_i}(\bar{v}_i)$.
2. For every edge $(\mathcal{E}_i, \sigma, \mathcal{E}_j){\in}E$, $\bar{v}_i{=}FV(\mathcal{E}_i){\neq}\emptyset$, $\bar{v}_j{=}FV(\mathcal{E}_j)$, $\bar{w}_j{=}FV(\mathcal{E}_j) \cap E$, we generate the clause: $\exists \bar{w}_j.\ \mathtt{gen}(\sigma) \wedge \mathtt{P_j}(\bar{v}_j) \Rightarrow \mathtt{P_i}(\bar{v}_i)$ where $\mathtt{gen}(\sigma)$ is defined as:

$$\mathtt{gen}(\sigma) == \begin{cases} n_x{=}0 & \text{if } \sigma{\equiv}[\epsilon/x] \\ n_x{=}n_y{+}1 & \text{if } \sigma{\equiv}[cy/x] \\ n_x{=}n_y{+}n_z & \text{if } \sigma{\equiv}[yz/x] \end{cases}$$

3. For every cycle $\mathcal{C}(\mathcal{E}_c{\to}\mathcal{E}_b, \sigma_{cyc}){\in}\mathcal{C}$, we generate the following clause:

$$\bigwedge\{v_{b_i}{=}v_{c_i} \mid [v_{c_i}/v_{b_i}] \in \sigma_{cyc}\} \wedge \mathtt{P_c}(\bar{v}_c) \Rightarrow \mathtt{P_b}(\bar{v}_b)$$

The length constraint of all solutions of $\mathcal{E}_0{\wedge}\Upsilon$ is captured by the query: $\mathtt{P_0}(FV(\mathcal{E}_0))$.

In the following, we show that if $\mathcal{T}_n$ is a flat tree, the satisfiability of the generated CHC is decidable. This decidability relies on the decidability of inductive predicates in DPI fragment which is presented in [46]. In particular, a system of inductive predicates is in DPI fragment if every predicate $\mathtt{P}$ is defined as follows. Either it is constrained by one base clause as: $\phi^b \Rightarrow \mathtt{P}(\bar{v})$ or it is defined by two clauses as:

$${\phi^b}_1{\wedge}..{\wedge}{\phi^b}_m \Rightarrow \mathtt{P}(\bar{v}) \qquad \exists \bar{w}.\ \bigwedge\{\bar{v}_i{\pm}\bar{t}_i{=}k\}{\wedge}\mathtt{P}(\bar{t}) \Rightarrow \mathtt{P}(\bar{v})$$

where $FV({\phi^b}_j) \in \bar{v}$ (for all $i \in 1..m$) and has at most one variable; $\bar{t} \subseteq \bar{v} \cup \bar{w}$, $\bar{v}_i$ is the variable at i^{th} position of the sequence $\bar{v}$, and $k \in \mathbb{Z}$.

To solve the generated clauses $\mathcal{R}$, we infer definitions for the unknown predicates in a bottom-up manner. Under assumption that $\mathcal{T}_n$ does not contain any mutual cycles, all mutual recursions can be eliminated and predicates are in the DPI fragment.

Proposition 4. *The length constraint implied by a flat tree is Presburger-definable.*

Example 4 (Motivating Example Revisited). We generate the following CHC for the tree $\mathcal{T}_3$ in Fig. 3.

$$\begin{array}{ll}
\exists n_{x_1}.n_x{=}n_{x_1}{+}1 \wedge \mathtt{P}_{12}(n_{x_1},n_y) & \Rightarrow \mathtt{P}_0(n_x,n_y) \\
n_{x_1}{=}0 \wedge \mathtt{P}_{21}(n_y) & \Rightarrow \mathtt{P}_{12}(n_{x_1},n_y) \\
\exists n_{x_2}.n_{x_1}{=}n_{x_2}{+}1 \wedge \mathtt{P}_{22}(n_{x_2},n_y) & \Rightarrow \mathtt{P}_{12}(n_{x_1},n_y) \\
n_{x_2}{=}n_x \wedge \mathtt{P}_0(n_x,n_y) & \Rightarrow \mathtt{P}_{22}(n_{x_2},n_y) \\
n_y{=}0 & \Rightarrow \mathtt{P}_{21}(n_y) \\
\exists n_{y_1}.n_y{=}n_{y_1}{+}1 \wedge \mathtt{P}_{32}(n_{y_1}) & \Rightarrow \mathtt{P}_{21}(n_y) \\
n_{y_1}{=}n_y \wedge \mathtt{P}_{21}(n_y) & \Rightarrow \mathtt{P}_{32}(n_{y_1}) \\
\mathtt{P}_0(n_x,n_y) \wedge (\exists k.n_x{=}4k{+}3) \wedge n_x{=}2n_y &
\end{array}$$

After eliminating the mutual recursion, predicate $\mathtt{P}_{21}$ is in the DPI fragment and generated a definitions as: $\mathtt{P}_{21}(n_y) \equiv n_y{\geq}0$. Similarly, after substituting the definition of $\mathtt{P}_{21}$ into the remaining clauses and eliminating the mutual recursion, predicate $\mathtt{P}_0$ is in the DPI fragment and generated a definitions as: $\mathtt{P}_0(n_x,n_y) \equiv \exists i.n_x{=}2i{+}1{\wedge}n_y{\geq}0$.

$\mathtt{STR}_{\mathtt{flat}}$ *Decidable Fragment* A quadratic word equation is called *regular* if it is either acyclic or of the form $Xw_1 = w_2X$ where X is a string variable and $w_1, w_2 \in \Sigma^*$. A quadratic word equation is called *phased-regular* if it is of the form: $s_1{\cdot}...{\cdot}s_n = t_1{\cdot}...{\cdot}t_n$ where $s_i{=}t_i$ is a regular equation for all $i \in \{1...n\}$.

Definition 4 ($\mathtt{STR}_{\mathtt{flat}}$ Formulas) *$\pi{\equiv}\mathcal{E}{\wedge}\Upsilon{\wedge}\alpha$ is called in the $\mathtt{STR}_{\mathtt{flat}}$ fragment if either $\mathcal{E}$ is both quadratic and phased-regular or $\mathcal{E}$ is in SL fragment.*

For example, $\pi \equiv abx{=}xba{\wedge}ay{=}ya \wedge |x|{=}2|y|$ is in $\mathtt{STR}_{\mathtt{flat}}$. But $\mathtt{e}_c{\equiv}xaby{=}ybax$ is *not* in $\mathtt{STR}_{\mathtt{flat}}$.

Proposition 5. *ω-SAT constructs a flat tree for a $\mathtt{STR}_{\mathtt{flat}}$ constraint in linear time.*

Let SAT-STR[$\mathtt{STR}_{\mathtt{flat}}$] be the satisfiability problem in this fragment.

Theorem 2. SAT-STR*[$\mathtt{STR}_{\mathtt{flat}}$] is decidable.*

8 Implementation and Evaluation

We have implemented a prototype for $\mathtt{Kepler}_{22}$, using OCaml, to handle the satisfiability problem in theory of word equations and length constraints over the Presburger arithmetic. It takes a formula in SMT-LIB format version as input and produces `SAT` or `UNSAT` as output. For the problem beyond the decidable fragments, ω-SAT may not terminate and $\mathtt{Kepler}_{22}$ may return `UNKNOWN`. We made use of Z3 [14] as a back-end SMT solver for the linear arithmetic.

Table 1: Experimental Results

	#$\surd$SAT	#$\surd$UNSAT	#✗SAT	#✗UNSAT	#UNKNOWN	#timeout	ERR	Time
Trau [4]	8	73	8	0	354	117	40	713m33s
S3P [3]	55	110	1	0	100	253	81	801m55s
CVC4 [1]	120	143	0	69	0	268	0	795m49s
Norn [2]	67	98	0	3	432	0	0	336m20s
Z3str3 [5]	69	102	0	0	292	24	113	77m4s
Z3str2 [51]	136	66	0	0	380	18	0	54m35s
$\mathtt{Kepler}_{22}$	298	302	0	0	0	0	0	18m58s

Evaluation As noted in [22,12], all constraints in the standard Kaluza benchmarks [43] with 50,000+ test cases generated by symbolic execution on JavaScript applications satisfy the straight-line conditions. Therefore, these benchmarks are not be suitable to evaluate our proposal that focuses on the cyclic constraints. We have generated and experimented $\mathtt{Kepler}_{22}$ over a new set of 600 hand-drafted benchmarks each of which is in the the proposed decidable fragments. The set of benchmarks includes 298 satisfiable queries and 302 unsatisfiable queries. For every benchmark which is a *phased-regular* constraint in $\mathtt{STR}_{\mathtt{flat}}$, it has from one to three phases. We have also compared $\mathtt{Kepler}_{22}$ against the existing state-of-the-art string solvers: Z3-str2 [52,51], Z3str3 [9], CVC4 [34], S3P [48], Norn [7,8] and Trau [6]. All experiments were performed on an Intel Core i7 3.6Gh with 12GB RAM. Experiments on Trau were performed in the VirtualBox image provided by the Trau's authors.

The experiments are shown in Table 1. The first column shows the solvers. The column #$\surd$SAT (resp., #$\surd$UNSAT) indicates the number of benchmarks for which the solvers decided SAT (resp., UNSAT) correctly. The column #✗SAT (resp., #✗UNSAT) indicates the number of benchmarks for which the solvers decided UNSAT on satisfiable queries (resp., SAT on unsatisfiable queries). The column #UNKNOWN indicates the number of benchmarks for which the solvers returned unknown, *timeout* for which the solvers were unable to decide within 180 seconds, *ERR* for internal errors. The column *Time* gives CPU running time (*m* for minutes and *s* for seconds) taken by the solvers.

The experimental results show that among the existing techniques that deal with cyclic scenarios, the method presented by Z3-str2 performed the most effectively and efficiently. It could detect the overlapping variables in 380 problems (63.3%) without any wrong outcomes in a short running time. Moreover, it could decide 202 problems (33.7%) correctly. CVC4 produced very high number of correct outcome (43.8% - 263/600). However, it returned both false positives and false negatives. Finally, non-progressing detection method in S3P worked not very well. It detected non-progressing reasoning in only 98 problems (16.3%) but produced false negatives and high number of timeouts and internal errors (crashes). Surprisingly, Norn performed really well. It could detect the highest number of the cyclic reasoning (432 problems - 72%). Trau was able to solve a small number of problems with 8 false negatives. The results also show that $\mathtt{Kepler}_{22}$ was both effective and efficient on these benchmarks. It decided correctly all queries within a short running time. These results are encouraging us to extend the proposed cyclic proof system to support inductive reasoning over other string operations (like `replaceAll`).

To highlight our contribution, we revisit the problem $\mathtt{e}_c \equiv xaay{=}ybax$ (highlighted in Sect. 1) which is contained in file quad−004−2−unsat of the benchmarks. $\mathtt{Kepler}_{22}$ generates a cyclic proof for $\mathtt{e}_c$ with the base case $\mathtt{e}_c^1 \vee \mathtt{e}_c^2$ where $\mathtt{e}_c^1 \equiv \mathtt{e}_c[\epsilon/x] \equiv aay{=}yba$ and $\mathtt{e}_c^2 \equiv \mathtt{e}_c[\epsilon/y] \equiv xaa{=}bax$. It is known that for certain words w_1, w_2 and a variable z the word equation $z{\cdot}w_1{=}w_2{\cdot}z$ is satisfied if there exist words A, B and a natural number i such that $w_1{=}A{\cdot}B$, $w_2{=}B{\cdot}A$ and $z{=}(A{\cdot}B)^i{\cdot}A$. Therefore, both $\mathtt{e}_c^1$ and $\mathtt{e}_c^2$ are unsatisfiable. The soundness of the cyclic proof implies that $\mathtt{e}_c$ is unsatisfiable. For this problem, while $\mathtt{Kepler}_{22}$ returned UNSAT within 1 second, Z3str2 and Z3str3 returned UNKNOWN, S3P, Norn and CVC4 were unable to decide within 180 seconds.

9 Related Work and Conclusion

Makanin notably provides a mathematical proof for the satisfiability problem of word equation [37]. In the sequence of papers, Plandowski *et.al.* showed that the complexity of this problem is PSPACE [39]. The proposed procedure ω-SAT is closed to the (more general) problem in computing the set of all solutions for a word equation [27,40,20,28,13]. The algorithm presented in [27] which is based on Makanin's algorithm does not terminate if the set is infinite. Moreover, the length constraints derived by [40,28] may not be in a finite form. In comparison, due to the consideration of cyclic solutions, ω-SAT terminates even for infinite sets of all solutions. ω-SAT is relevant to the Nielsen transform [44,17] and cyclic proof systems [10,30,32,31]. Our work extends the Nielsen transform to the set of all solution to handle the string constraints beyond the word equations. Furthermore, in contrast to the cyclic systems our soundness proof is based on the fact that solutions of a word equation must be finite. The description of the sets of all solutions as EDT0L languages was known [20,13]. For instance, authors in [20] show that the languages of quadratic word equations can be recognized by some pushdown automaton of level 2. Although [28] did not aim at giving such a structural result, it provided *recompression* method which is the foundation for the remarkable procedure in [13] which prove that languages of solution sets of arbitrary word equations are EDT0L. In this work, we propose a decision procedure which is based on the description of solution sets as *finite-index* EDT0L languages. Like [20], we also show that sets of all solutions of quadratic word equation are EDT0L languages. In contrast to [20], we give a concrete procedure to construct such languages for a solvable equation such that an implementation of the decision procedure for string constraints is feasible. As shown in this work, finite-index feature is the key to obtain a decidability result when handling a theory combining word equations with length constraints over words. It is unclear whether the description derived by the procedure in [13] is the language of finite index. Furthermore, node of the graph derived by [13] is an extended equation which is an element in a free partially commutative monoid rather than a word equation.

Decision procedures for quadratic word equations are presented in [44,17]. Moreover, Schulz [44] also extends Makanin's algorithm to a theory of word equations and regular memberships. Recently, [24,25] presents a decision procedure for subset constraints over regular expressions. [35] presents a decision procedure for regular memberships and length constraints. [22,7] presents a decidable fragment of *acyclic* word equations, regular expressions and constraints over length functions. It can be implied that this fragment is subsumed by ours. [36,12,23] presents a straight-line fragment including word equations and transducer-based functions (e.g., `replaceAll`) which is incomparable to our decidable fragments. Z3str [52] implements string theory as an extension of Z3 SMT solver through string plug-in. It supports unbounded string constraints with a wide range of string operations. Intuitively, it solves string constraints and generates string lemmas to control with Z3's congruence closure core. Z3str2 [51] improves Z3str by proposing a detection of those constraints beyond the tractable fragment, i.e. overlapping arrangement, and pruning the search space for efficiency. Similar to Z3str, CVC4-based string solver [33] communicates with CVC4's equality solver to exchange information over string. S3P [47,48] enhances Z3str to incrementally interchange information between string and arithmetic constraints. S3P also presented some heuristics to detect and prune non-minimal subproblems while searching for a proof. While the technique in S3P was able to detect non-progressing scenarios of satisfiable formulas, it would not terminate for unsatisfiable formulas due to presence of multiple occurrences of each string variable. Our solver can support well for both classes of queries in case of less than or equal to two occurrences of each string variable.

Conclusion We have presented the solver $\texttt{Kepler}_{22}$ for the satisfiability of string constraints combining word equations, regular expressions and length functions. We have identified two decidable fragments including quadratic word equations. Finally, we have implemented and evaluated $\texttt{Kepler}_{22}$. Although our solver is only a prototype, the results are encouraging for their coverage as well as their performance. For future work, we plan to support other string operations (e.g., replaceAll). Deriving the length constraint implied by more expressive word equations would be another future work.

Acknowledgments. Anthony W. Lin and Vijay Ganesh for the helpful discussions. Cesare Tinelli and Andrew Reynolds for useful comments and testing on the benchmarks over CVC4. We thank Bui Phi Diep for his generous help on Trau experiments. We are grateful for the constructive feedback from the anonymous reviewers.

References

1. CVC4-1.5. http://cvc4.cs.stanford.edu/web/. Online; accessed: 14-Jun-2018.
2. Norn. http://user.it.uu.se/ jarst116/norn/. accessed: 14-Jun-2018.
3. S3P. http://www.comp.nus.edu.sg/ trinhmt/S3/S3P-bin-090817.zip. accessed: 20-Jan-2018.
4. TRAU. https://github.com/diepbp/fat. Online; accessed: 10-Jun-2018.
5. Z3str3. https://sites.google.com/site/z3strsolver/getting-started. accessed: 14-Jun-2018.
6. P. A. Abdulla, M. F. Atig, Y.-F. Chen, B. P. Diep, L. Holik, A. Rezine, and P. Rummer. Flatten and conquer: A framework for efficient analysis of string constraints. In *PLDI*, 2017.
7. P. A. Abdulla, M. F. Atig, Y.-F. Chen, L. Holik, A. Rezine, P. Rummer, and J. Stenman. *CAV*, chapter String Constraints for Verification, pages 150–166. Cham, 2014.
8. P. A. Abdulla, M. F. Atig, Y.-F. Chen, L. Holik, A. Rezine, P. Rummer, and J. Stenman. *CAV*, chapter Norn: An SMT Solver for String Constraints, pages 462–469. Cham, 2015.
9. M. Berzish, V. Ganesh, and Y. Zheng. Zsstrs: A string solver with theory-aware heuristics. In *2017 Formal Methods in Computer Aided Design (FMCAD)*, pages 55–59, Oct 2017.
10. J. Brotherston. Cyclic proofs for first-order logic with inductive definitions. In *Proceedings of TABLEAUX-14*, volume 3702 of *LNAI*, pages 78–92. Springer-Verlag, 2005.

11. J. R. Büchi and S. Senger. *Definability in the Existential Theory of Concatenation and Undecidable Extensions of this Theory*, pages 671–683. Springer New York, 1990.
12. T. Chen, Y. Chen, M. Hague, A. W. Lin, and Z. Wu. What is decidable about string constraints with the replaceall function. *POPL*, 2018.
13. L. Ciobanu, V. Diekert, and M. Elder. *Solution Sets for Equations over Free Groups are EDT0L Languages*, pages 134–145. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.
14. L. M. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *TACAS*, 2008.
15. V. Diekert. *Makanin's Algorithm*. Cambridge University Press, 2002.
16. V. Diekert. *More Than 1700 Years of Word Equations*, pages 22–28. Springer International Publishing, Cham, 2015.
17. V. Diekert and J. M. Robson. *Quadratic Word Equations*, pages 314–326. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.
18. J. Esparza. *Petri nets, commutative context-free grammars, and basic parallel processes*, pages 221–232. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.
19. J. Esparza, P. Ganty, S. Kiefer, and M. Luttenberger. Parikh's theorem: A simple and direct automaton construction. *Inf. Process. Lett.*, 111(12):614–619, June 2011.
20. J. Ferté, N. Marin, and G. Sénizergues. Word-mappings of level 2. *Theory of Computing Systems*, 54(1):111–148, 2014.
21. M. J. Fischer and M. O. Rabin. Super-exponential complexity of presburger arithmetic. Technical report, Cambridge, MA, USA, 1974.
22. V. Ganesh, M. Minnes, A. Solar-Lezama, and M. Rinard. Word equations with length constraints: What's decidable? In *HVC*, pages 209–226. Springer-Verlag, 2013.
23. L. Holik, P. Janku, A. W. Lin, P. Ruemmer, and T. Vojnar. String constraints with concatenation and transducers solved efficiently. *POPL*, 2018.
24. P. Hooimeijer and W. Weimer. A decision procedure for subset constraints over regular languages. In *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '09, pages 188–198, New York, NY, USA, 2009. ACM.
25. P. Hooimeijer and W. Weimer. Solving string constraints lazily. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ASE '10, pages 377–386, New York, NY, USA, 2010. ACM.
26. J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., 2006.
27. J. Jaffar. Minimal and complete word unification. *J. ACM*, 37(1):47–85, Jan. 1990.
28. A. Jez. Recompression: A simple and powerful technique for word equations. *J. ACM*, 63(1):4:1–4:51, 2016.
29. I. Khmelevskii. Equations in free semigroups. Number 107. Issue 107 of Proceedings of the Steklov Institute of Mathematics, 1971. English Translation in Proceedings of American Mathematical Society, 1976.
30. Q. L. Le, S. Jun, and W.-N. Chin. Satisfiability modulo heap-based programs. In *CAV*, 2016.
31. Q. L. Le, J. Sun, and S. Qin. Frame inference for inductive entailment proofs in separation logic. In D. Beyer and M. Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 41–60, Cham, 2018. Springer International Publishing.
32. Q. L. Le, M. Tatsuta, J. Sun, and W.-N. Chin. A decidable fragment in separation logic with inductive predicates and arithmetic. In R. Majumdar and V. Kuncak, editors, *Computer Aided Verification*, pages 495–517, Cham, 2017. Springer International Publishing.
33. T. Liang, A. Reynolds, C. Tinelli, C. Barrett, and M. Deters. *CAV*, chapter A DPLL(T) Theory Solver for a Theory of Strings and Regular Expressions, pages 646–662. Cham, 2014.
34. T. Liang, A. Reynolds, N. Tsiskaridze, C. Tinelli, C. Barrett, and M. Deters. An efficient smt solver for string constraints. *Form. Methods Syst. Des.*, 48(3):206–234, June 2016.

35. T. Liang, N. Tsiskaridze, A. Reynolds, C. Tinelli, and C. Barrett. *FroCoS*, chapter A Decision Procedure for Regular Membership and Length Constraints over Unbounded Strings, pages 135–150. Cham, 2015.
36. A. W. Lin and P. Barceló. String solving with word equations and transducers: Towards a logic for analysing mutation xss. In *POPL*, pages 123–136. ACM, 2016.
37. G. Makanin. The problem of solvability of equations in a free semigroup. *Mathematics of the USSR-Sbornik*, 32(2):129–198, 1977.
38. R. J. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, Oct. 1966.
39. W. Plandowski. Satisfiability of word equations with constants is in pspace. *J. ACM*, 51(3):483–496, May 2004.
40. W. Plandowski. An efficient algorithm for solving word equations. In *STOC*, pages 467–476, New York, NY, USA, 2006. ACM.
41. G. Rozenberg and A. Salomaa. *Handbook of Formal Lanuages. Volume 1: Word, Language, Grammar*. Springer, 1997.
42. G. Rozenberg and D. Vermeir. On etol systems of finite index. *Information and Control*, 38(1):103 – 133, 1978.
43. P. Saxena, D. Akhawe, S. Hanna, F. Mao, S. McCamant, and D. Song. A symbolic execution framework for javascript. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, SP '10, pages 513–528, Washington, DC, USA, 2010. IEEE Computer Society.
44. K. U. Schulz. Makanin's algorithm for word equations - two improvements and a generalization. In *Proceedings of the First International Workshop on Word Equations and Related Topics*, IWWERT '90, pages 85–150, London, UK, UK, 1992. Springer-Verlag.
45. H. Seidl, T. Schwentick, A. Muscholl, and P. Habermehl. *Counting in Trees for Free*, pages 1136–1149. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.
46. M. Tatsuta, Q. L. Le, and W.-N. Chin. *Decision Procedure for Separation Logic with Inductive Definitions and Presburger Arithmetic*, pages 423–443. APLAS, Cham, 2016.
47. M.-T. Trinh, D.-H. Chu, and J. Jaffar. S3: A symbolic string solver for vulnerability detection in web applications. In *CCS*, pages 1232–1243, New York, NY, USA, 2014. ACM.
48. M.-T. Trinh, D.-H. Chu, and J. Jaffar. Progressive reasoning over recursively-defined strings. In *CAV*, 2016.
49. K. N. Verma, H. Seidl, and T. Schwentick. *On the Complexity of Equational Horn Clauses*, pages 337–352. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.
50. Y. Zheng, V. Ganesh, S. Subramanian, O. Tripp, M. Berzish, J. Dolby, and X. Zhang. Z3str2: An efficient solver for strings, regular expressions, and length constraints. *Form. Methods Syst. Des.*, 50(2-3):249–288, June 2017.
51. Y. Zheng, V. Ganesh, S. Subramanian, O. Tripp, J. Dolby, and X. Zhang. *CAV*, chapter Effective Search-Space Pruning for Solvers of String Equations, Regular Expressions and Length Constraints, pages 235–254. Cham, 2015.
52. Y. Zheng, X. Zhang, and V. Ganesh. Z3-str: A z3-based string solver for web application analysis. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 114–124, New York, NY, USA, 2013. ACM.