Decidable Logics Combining Word Equations, Regular Expressions and Length Constraints

Quang Loc Le
Teesside University, United Kingdom

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.

Keywords: String Solver · Word Equations · Decidability · Cyclic Proofs.

1 Introduction

The problem of solving word algebras has been studied since the early stage in 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 [36] 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 [2,7,35,12,23]. 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: \( e_c \equiv x \cdot a \cdot x = y \cdot b \cdot a \cdot y \) 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 [50,49], CVC4 [33] and S3P [47]. 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 [47] 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 [7,8,6,10,12,23]) is not able to decide the satisfiability of the word equation \( e_c \) 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 \( e_c \) 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 novel algorithm, called \( \omega\text{-SAT} \), to derive a finite representation for all solutions of a conjunction of word equations and regular expressions.
- We present a decision procedure, called \( \text{Kepler}_{22} \), 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 our proposal is both effective and efficient in solving string constraints with quadratic word 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. For the space reason, all missing proofs are presented in Appendix.

## 2 Preliminaries

Concrete string models assume a finite alphabet \( \Sigma \) whose elements are called *letters*, set of finite words over \( \Sigma^* \) including \( \epsilon \) - the empty word, and a set of integer numbers \( \mathbb{Z} \). We
A deterministic finite automaton (DFA) $A$ is a tuple: $A=⟨Q,Σ,δ,q_0,Q_F⟩$, where $Q$ is a finite set of states, $δ ⊆ Q×(Σ∪{ε})×Q$ is a finite set of transitions, $q_0∈Q$ is the initial state, $Σ$ is the set of alphabet, $Q_F ⊆ Q$ is the set of accepting states, and $δ(q,x,q')=1$ if the transition of state $q$ with input $x$ is $q'$. The semantics of this language is formalized in App. A. Inductive predicate is interpreted as a least fixed-point of values [46]. If $η,β_η ⊨ π$, we use the pair $⟨η,β_η⟩$ to denote a solution of the formula $π$. Let $e≡x_1...x_{i+1}...x_n$ be a word equation. If $e$ is satisfied with the solution $⟨η,β_η⟩$, we also refer $η(x_1)...η(x_i)$ as a solution word of $e$. A solution word is minimal if the length of the solution word $⟨|η(x_1)|+...+|η(x_i)|⟩$ is minimal. $e_1$ is referred as a suffix of $e_2$ if they are satisfied and the solution word of $e_1$ is a suffix of the solution word of $e_2$.

|_disj formula $π ::= φ | π_1 ∨ π_2$
|_formula $φ ::= e | α | s∈R | ¬φ_1 | φ_1 ∧ φ_2$
|_(dis)equality $e ::= s_1=s_2$
|_term $s ::= ε | c | x | s_1·s_2$
|_regex $R ::= \emptyset | ε | c | w | R_1 · R_2 | R_1 + R_2 | R_1 ∩ R_2 | R_1^C \ C R_2^* | R_1^*$
|_Arithmetic $α ::= a \cdot a | a > a | α_1 ∧ α_2 | α_1 ∨ α_2 | ¬α | α_1 | α_2 | P(\bar{v})$
|_a ::= 0 | 1 | v | |u| | i · x_1 | a_1 ∨ a_2$

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 $w∈Σ^*$ and $\bar{v}$ a sequence of variables. A language $L$ over the alphabet $Σ$ is a set $L⊆Σ^*$. A language $L$ is a set of words generated by a grammar system. We use $L(L)$ to denote the class of all languages $L$.

**Syntax** The syntax of quantifier-free string formulas, called $\text{STR}$, is presented in Fig. 1. $π$ is a conjunction formula where each disjunct $φ$ is a conjunction of word equations $e$, regular memberships $s∈R$ and arithmetic constraints $α$. Especially, $α$ may contain predicates $P(\bar{v})$ whose definitions are inductively defined. We use either $s$ or $tr$ to denote a string term. We often write $s_1s_2$ to denote $s_1 · s_2$ if it is not ambiguous. Regular expression $R$ over $Σ$ is built over $c∈Σ$, $w∈Σ^*$, $ε$, and closing under union $+$, intersection $∩$, complement $C$, concatenation $·$, and the Kleene star operator $*$. Regular expressions $R$ does not contain any string variables.

We use $E$ to denote a conjunction (a.k.a system) of word equations. $π[t_1/t_2]$ denotes a substitution of all occurrences of $t_2$ in $π$ to $t_1$. We use function $FV(π)$ to return all free variables of $π$. We inductively define length function of a string term $s$, denoted as $|s|$, as: $|ε|=0$, $|c|=1$, and $|s_1·s_2|=|s_1|+|s_2|$. Notational length of the word equation $e$, denoted by $o(N)$, 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 [22735] if it can be rewritten (by reordering the conjuncts) as the form $\bigwedge_{i=1}^n x_i = s_1$ such that: (i) $x_1,...,x_n$ are different variables; and (ii) $FV(s_1) ⊆ \{x_1,x_2,...,x_{i-1}\}$. A formula $π ≡ e_1 · e_2 · ... · e_n ∨ T$ is called in straight-line fragment (SL) if $e_1 · e_2 · ... · e_n$ is straight-line and the regular expression $T$ is of the conjunction of regular memberships $x_j∈R_j$ where $x_j∈\{x_1,...,x_n\}$.

**Semantics** Every regular expression $R$ is evaluated to the language $L(R)$. We define:

$SSStacks \overset{\text{def}}{=} (U∪Σ)→Σ^*$

$ZStacks \overset{\text{def}}{=} I→\mathbb{Z}$.

The semantics is given by a satisfaction relation: $η,β_η ⊨ π$ that forces the interpretation on both string $η$ and arithmetic $β_η$ to satisfy the constraint $π$ where $η ∈ SStacks$, $β_η ∈ ZStacks$, and $π$ is a formula. We remark that $η ∈ SStacks$: $η(c)=c$ for all $c∈Σ$ and $η(t_1τ_2)=η(t_1)η(τ_2)$. The semantics of our language is formalized in App. A. Inductive predicate is interpreted as a least fixed-point of values [46]. If $η,β_η ⊨ π$, we use the pair $⟨η,β_η⟩$ to denote a solution of the formula $π$. Let $e≡x_1...x_{i+1}...x_n$ be a word equation. If $e$ is satisfied with the solution $⟨η_1,β_1⟩$, we also refer $η_1(x_1)...η_1(x_i)$ as a solution word of $e$. A solution word is minimal if the length of the solution word $⟨|η(x_1)|+...+|η(x_i)|⟩$ is minimal. $e_1$ is referred as a suffix of $e_2$ if they are satisfied and the solution word of $e_1$ is a suffix of the solution word of $e_2$.
is the initial state and $Q_F \subseteq Q$ is a set of accepting states. We use $L(A)$ 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 [20].

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, $\Sigma$ 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 $(\alpha, \beta)$ to $u$ if $\exists (\alpha, \beta) \in P$ such that $u = u_1 \alpha u_2$ and $v = u_1 \beta u_2$. $L(G) = \{ w \in \Sigma^* | S \Rightarrow^* G w \}$ 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 $L(G)$, defined by $L(G_X) = \{ w \in \Sigma^* | X \Rightarrow^*_G w \}$.

**Problem Definition**

Throughout this work, we consider the following problem.

**PROBLEM:** SAT–STR.
**INPUT:** A string constraint $\pi$ in normal form over $\Sigma$.
**QUESTION:** Is $\pi$ satisfiable?

Authors in [22,7,35] show that this problem in straight-line fragment is decidable.

## 3 Overview and Illustration

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 $T_i$ is a tuple $(V, E, C)$ where $V$ is a finite set of nodes where each node represents a conjunction of word equations $E$. $E$ is a set of labeled and directed edges $(\epsilon, \sigma, \epsilon') \in E$ where $\epsilon'$ is a child of $\epsilon$. This edge means we can reduce $\epsilon$ to $\epsilon'$ via the label $\sigma$, a substitution, s.t.: $\epsilon' \equiv \epsilon \sigma$. And $C$ is a back-link (partial) function which captures virtual cycles in the tree. A cycle, e.g. $C(\epsilon_c \rightarrow \epsilon_b, \sigma)$, in $C$ means the leaf $\epsilon_b$ is linked back to its ancestor $\epsilon_c$ and $\epsilon_c \equiv \epsilon_b \sigma$. In this back-link, $\epsilon_b$ is referred as a bud and $\epsilon_c$ is referred as a companion. A path $(v_s, v_e)$ is a sequence of nodes and edges connecting node $v_s$ with node $v_e$. 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 $(v_s, v_e)$ where $v_s$ is the root and $v_e$ 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:
The representation of all solutions \( \omega\text{-SAT} \) derives the reduction tree \( T_3 \) \((V, E, C)\), shown in Figure 2 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: \( \mathcal{E}_{12}^{'}=\mathcal{E}_0[ax_1/x] \equiv abax_1=abx_1a \) 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, \( \mathcal{E}_{11} \) is 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 \( \mathcal{E}_{21}^{'}=\mathcal{E}_{12}^{'}[\epsilon/x_1] \equiv ab=ba \) and \( x_1=bx_2 \) (where \( x_2 \) is a fresh variable) to generate \( \mathcal{E}_{22}^{'}=\mathcal{E}_{12}^{'}[bx_2/x_1] \equiv abx_2=bx_2b \). 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 \( \mathcal{C}(\mathcal{E}_0 \rightarrow \mathcal{E}_{22}, [x/x_2]) \). Similarly, \( \mathcal{E}_{21} \) is reduced until all leaf nodes are marked closed.

A path \((v_s, v_e)\) with trace \( \sigma \) represents for \( v_e \equiv v_s \sigma \). If \( v_e \) is satisfiable, then \( \sigma \) represents for a family of solutions (or valid assignments). For instance, in Fig. 2 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 \( \sigma_{31} \) 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 \( \sigma_{ij} \) obtained from the path \((\mathcal{E}_0, \mathcal{E}_{31})\) above is as:

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

where \( \circ \) is the substitution composition operation, \( \sigma^k \) means \( \sigma \) is repeatedly composed zero, one or more times, and \( i \geq 0, j \geq 0 \).
Computing \( \alpha_{xy} \) constraint Based on the solution trace \( \sigma_{ij} \) above, \texttt{Kepler}\textsubscript{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: \( \alpha_{xy} \equiv (\exists i. |x| = 2i + 1 \land i \geq 0 \land |y| \geq 0) \). Now, the satisfiability of \( \pi \) is equi-satisfiable to the following formula: \( \pi' \equiv (\exists i. |x| = 2i + 1 \land i \geq 0 \land |y| \geq 0) \land (\exists k. |x| = 4k + 3) \land |x| = 2|y| \). As \( \pi' \) is unsatisfiable, so is \( \pi \).

4 The Representation of All Solutions

In this section, we first present procedure \( \omega\text{-SAT} \) which constructs a cyclic reduction tree for a conjunction of word equations \( \mathcal{E} \) (subsection 4.1). We presents a fairly complicated cyclic reduction tree of \( e_i \equiv xaby = ybax \) in subsection 4.2. After that, we describe how to combine the tree with regular membership predicates \( T \) (subsection 4.3). Finally, we discuss the correctness in subsection 4.4.

4.1 Constructing Cyclic Reduction Tree

\( \omega\text{-SAT} \) transforms a conjunction of word equations \( \mathcal{E} \) into a cyclic reduction tree \( T_n \) which represents all its solutions. This procedure starts with the tree \( T_0 \) with only the input \( \mathcal{E} \) at the root. After that, in each iteration it chooses one leaf node to reduce (using function \texttt{reduce}) or to make a back-link (using function \texttt{link back}) until every leaf node is either irreducible or linked back. A leaf node is irreducible if it either trivially true (i.e., \( w_1 = u_1 \land \ldots \land w_i = u_i \) where \( w_1, \ldots, w_i \in \Sigma^* \)) or trivially false (i.e., either it is of the form: \( c_1 t_1 = c_2 t_2 \land \mathcal{E} \) where \( c_1, c_2 \) are different letters or its over-approximation over the length functions is unsatisfiable). Function \texttt{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 \( \mathcal{E}_{ij} \) and a corresponding substitution \( \sigma_j \) such that \( \mathcal{E}_{ij} = \mathcal{E}_i \sigma_j \). For each pair \( (\mathcal{E}_{ij}, \sigma_j) \in L_i \), it adds an new open node \( \hat{\mathcal{E}}_i \) and a new edge \( (\mathcal{E}_i, \sigma_j, \hat{\mathcal{E}}_i) \). As a result, \texttt{reduce} extends the current tree with the new nodes and new edges. In particular, function \texttt{reduce} is implemented as: \( L_i = \{ \text{matchs}(\mathcal{E}_{ij}) \mid \mathcal{E}_{ij} \in \text{complete}(\mathcal{E}_i) \} \) where function \texttt{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 \texttt{match}. In the following, we describe the details of the functions used by \( \omega\text{-SAT} \).

Matching \texttt{match(e)} matches two terms at the heads of LHS and RHS of \( e \) as follows.

\[
\text{match}(u_1 t_1 = u_2 t_2) = \begin{cases} 
\text{match}(tr_1 = tr_2) & \text{if } u_1, u_2 \text{ are identical} \\
u_1 t_1 = u_2 t_2 & \text{otherwise}
\end{cases}
\]

where \( u_1, u_2 \) are either letters or string variables.

Procedure \texttt{complete} The overall goal of our reduction is to transform every word equation, say \( e \equiv u_1 t_1 = u_2 t_2 \) where \( \mathcal{E}_i = e \land \mathcal{E} \), into a set of “smaller” string equation \( e_i \) such that if \( e \) is satisfied, \( e_i \) 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 \( l_1, r_1 \) are the starting and ending positions of \( u_1 \) in the solution word of \( e \). Similarly, let \( l_2, r_2 \) are the starting and ending positions of \( u_1 \) in the
solution word of \( e \). Obviously, \( l_1 = l_2 \). Our reduction, function \texttt{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., \( u_1 = e \), \( l_1 < r_1 \) and \( l_2 = r_2 \) (i.e., \( u_2 = e \), \( l_2 < r_2 \). For arrangements between the two sides, it considers the cases: \( r_1 \geq r_2 \) and \( r_2 \geq r_1 \). In particular, function \texttt{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. \( x_1 t r_1 = c_2 t r_2 \).

**Case 2:** These terms are two different string variables, e.g. \( \text{Linking back are as follows.} \)

- 1a) \( \sigma_1 = [e/x_1] \)
- 1b) \( \sigma_2 = [c_2 x_1'/x_1], x_1' \) is a fresh variable and referred as a subterm of \( x_1 \).

**Case 2:** These terms are two different string variables, e.g. \( x_1 t r_1 = c_2 t r_2 \).

As both Case 2b and Case 2d include the scenario where \( \equiv \) is not minimal, function \texttt{complete} generates the set \( L_i \) as \( L_i \equiv \{ (\mathcal{E}_1, \sigma_1); (\mathcal{E}_2, \sigma_2); (\mathcal{E}_3, \sigma_3); (\mathcal{E}_4, \sigma_4) \} \) where

- 2a) \( \sigma_1 = [e/x_1] \),
- 2b) \( \sigma_3 = [x_2 x_1'/x_1] - x_1' \) is a fresh variable and referred as a subterm of \( x_1 \),
- 2c) \( \sigma_2 = [e/x_2] \),
- 2d) \( \sigma_4 = [x_1 x_2'/x_2], x_2' \) 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 \texttt{complete} but \texttt{not minimal} set of all solution.

**Linking back** \texttt{linkback} links a leaf node \( \mathcal{E}_b \) to an interior node \( \mathcal{E}_c \) if after some substitution \( \sigma_{cyc} \), 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 \( X \). \( \sigma_{cyc} \) can be considered as a permutation function on both \( U \) and the alphabet \( \Sigma \). We recap that we refer to this cycle as a triple \( C(\mathcal{E}_c \rightarrow \mathcal{E}_b, \sigma_{cyc}) \) where \( \mathcal{E}_c \) is called a companion, \( \mathcal{E}_b \) is called a bud.

### 4.2 Cyclic Reduction Tree for \( \omega \cong x a b y = y b a x \)

We describe how \( \omega \cdot \text{SAT} \) can derive a reduction tree for the word equation: \( \omega \equiv x a b y = y b a x \).

As mentioned before, although the work presented in [40] can derive a graph to finitely represent all solutions of the word equation \( \omega_c \), the length constraints implied for variables \( x \) and \( y \) by all solutions of this equation can not be represented with finitely many equations in numeric solvable form. Our decision procedure can decide that \( \pi_c \) is satisfiable. Indeed, it derives for \( \omega_c \) a reduction tree as presented in Fig. 3 where its nodes are as follows.

\[
\begin{align*}
\omega_1 \equiv & ab y = y b a \\
\omega_7 \equiv & a b y = b a y \\
\omega_{21} \equiv & b y = y b \\
\omega_{13} \equiv & e = e \\
\omega_{19} \equiv & x a = a x \\
\end{align*}
\]

\[
\begin{align*}
\omega_2 \equiv & x_1 a b y = b a y x_1 \\
\omega_8 \equiv & x_2 a b y = a y b x_2 \\
\omega_{22} \equiv & b y_2 = y b \\
\omega_{14} \equiv & a b y_3 = y_3 b a \\
\omega_{18} \equiv & x_5 a b = b a x_5 \\
\omega_{20} \equiv & b x y_6 = y b a x \\
\omega_{23} \equiv & e = e \\
\end{align*}
\]

\[
\begin{align*}
\omega_3 \equiv & x a b = b a x \\
\omega_{15} \equiv & b y = y b \\
\omega_5 \equiv & a b = b a \\
\omega_{10} \equiv & x_1 a b = a b x \\
\omega_{11} \equiv & a b x = b a x \\
\omega_{12} \equiv & b x a y_5 = y b a x \\
\omega_{24} \equiv & x a = a x \\
\end{align*}
\]

### 4.3 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 \texttt{widetree}. Procedure \texttt{widetree} takes a pair of a reduction tree \( \mathcal{T}_n \) of \( \mathcal{E}_0 \) (generated by \( \omega \cdot \text{SAT} \)) and a conjunction of regular expressions \( \mathcal{T} \) as inputs and manipulates the reduction tree \( \mathcal{T}_n \) through the following
three steps. First, it constructs a DFA \( A = (Q, \Sigma, \delta, q_0, Q_F) \) which generates the same language with \( \mathcal{T} \). 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 \( \mathcal{E}_0 \land \mathcal{T} \). \( M \) is the periodic of the sets of all solutions. Secondly, it unfolds every cycles \( \mathcal{C}(E_c \rightarrow E_b, \sigma) \) of \( \mathcal{T}_n \) \( m+M \) times. It updates \( 1 \) link back functions by eliminating the old back-link between \( E_b \) and \( E_c \) 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: \( S = u_1 w^{n+1+i} M w_2 \). Lastly, it collects label \( \sigma_j \) 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 \( \sigma_j \), it

---

**Fig. 3:** Cyclic Reduction Tree \( \mathcal{T}_{11} \) for \( xaby = ybax \).

---

**Fig. 4:** Extending Tree \( \mathcal{T}_{2} \) with \( x \in a^{*} \).
The class of systems which was introduced in 1968 to model the development of multicellular systems. EDT\_0L refers to interactionless Lindenmayer systems. (More discussion on EDT\_0L language is left in App. D.1.) In the following, we give a formal definition of EDT\_0L system.

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 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 \texttt{widentree} above to extend the tree to represent all solutions of \(\pi_1 \equiv abx=xba \land x \in a^*\). To do that, \texttt{widentree} first derives for the regular expression \(x \in a^*\) a DFA as: \(A = (\{q_0\}, \{q_0\}, \{(q_0, q_0), a\}, q_0, \{q_0\})\), 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 \(e_{21}, e_{21}^1\) or \(e_{21}^2\). The evaluation is as follows.

| path   | formula                                           | outcome |
|--------|---------------------------------------------------|---------|
| \((e_0, e_{21})\) | \(x=ax_1 \land x_1=\epsilon \land x \in a^*\)   | SAT     |
| \((e_0, e_{21}^1)\) | \(x = ax_1 \land x_1=bx_2 \land x_2=x_3 \land x_3=bx_4 \land x_4=x_5 \land x_5=\epsilon \land x \in a^*\) | UNSAT   |
| \((e_0, e_{21}^2)\) | \(x=ax_1 \land x_1=bx_2 \land x_2=x_3 \land x_3=bx_4 \land x_4=bx_5 \land x_5=\epsilon \land x \in a^*\) | UNSAT   |

4.4 Correctness

In the following, we formalize the correctness of the proposed procedures and show the relationship between the derived reduction tree with EDT\_0L system.

**Proposition 1.** Suppose that \(\omega:\text{SAT}\) 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 \(\mathcal{Y} \equiv X_1 \in \mathcal{R}_1 \land \ldots \land X_n \in \mathcal{R}_n\) \((X_i \in \text{FV}(\mathcal{E}_0), 1 \leq i \leq n)\) be a conjunction of regular memberships and \(\mathcal{T}_n\) be the reduction tree derived for \(\mathcal{E}_0\). Then, \(\text{widentree}(\mathcal{T}_n, \mathcal{Y})\) produces a reduction tree representing all solutions of \(\mathcal{E}_0 \land \mathcal{Y}\).

An interactionless Lindenmayer system (0L system) is a parallel rewriting system which was introduced in 1968 to model the development of multicellular system. The class of EDT\_0L languages forms perhaps the central class in the theory of L systems. The acronym EDT\_0L refers to Extended, Deterministic, Table, 0 interaction, and Lindenmayer. (More discussion on EDT\_0L language is left in App. D.1.) In the following, we give a formal definition of EDT\_0L 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), $\Sigma$ 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 $ET0L$ 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 $P$". 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 $x \Rightarrow y$.
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 $\mathcal{L}(G) = \{ w \in \Sigma^* | S \Rightarrow^*_G w \}$.

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 $\omega$-SAT forms a finite-index $ET0L$ system.

Example 2. The tree in the Fig. 5 above forms the following finite-index $ET0L$ system.

$G=\langle \{ S, x, x_1, x_2 \}, \Sigma, \{ P_1, P_2 \}, S \rangle$ 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 Kepler22 to handle SAT-STR. Kepler22 takes a constraint, say $\mathcal{E} \land \forall \alpha$, as input and returns SAT or UNSAT. It works as follows.

1. First, it invokes $\omega$-SAT to construct a reduction tree $T_n$ as a finite representation of all solutions of $\mathcal{E}$. After that, $T_n$ is post-processed using procedure postpro as below to explicate all free variables. This step is critical to the next step.
2. Secondly, it uses procedure identree to extend $T_n$ with membership predicates $\forall$ and obtains $T_{n+1}$. Note that unsatisfiable nodes in the reduction tree are eliminated.
3. Thirdly, it computes the length constraints which are precisely implied by all solutions generated through procedure extract_pres$(T_{n+1})$. These length constraints, say $\alpha_w$, are computed as an existentially quantified Presburger formula.
4. Lastly, Kepler22 solves that satisfiability of the conjunction $\alpha_w \land \alpha$ which is in the Presburger arithmetic and decidable $[21]$. 

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 $\Sigma^*$ in a solution. Procedure postpro aims to replace a free variable by a sub-tree which represents for arbitrary values in $\Sigma^*$. The sub-tree is presented in Fig. 6. This tree has a base leaf node (with substitution $[\epsilon/x]$) and $k$ cycles ($k$ is the size of the alphabet $\Sigma$) 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 Kepler22 is a decision procedure in a fragment are the termination of $\omega$-SAT as well as the decidability of extract pres$(T_{n+1})$.

6 STREDTL Decidable Fragment

Computing length constraint in this fragment is based on Parikh’s Theorem [37], 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 [37] 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 [48,19]. 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 $L$.

Given a constraint, say $E \land T \land \pi$, is said to be in the fragment if the following two conditions hold. First, $\omega$-SAT terminates on $E$. Secondly, $\pi \equiv \alpha_1 \wedge \ldots \wedge \alpha_n$ where $FV(\alpha_i)$ contains at most one string length $\forall i \in \{1...n\}$. By the first condition, Kepler22 can derive for $E$ a finite-index EDT0L system (Corollary 4.1). Moreover, finite-index EDT0L can be translated into a Parikh-equivalent DFA (by Parikh’s Theorem [37,42]). 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 $\alpha_1$ 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 $\pi$ is decidable.

Kepler22 uses extract pres$(T_{n+1})$ to compute the length constraints represented for all solutions of $E \land T$ as follows. Firstly, it transforms $T_{n+1}$ 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 $\alpha_w$ for every string variables as: $\alpha_w \equiv \bigwedge \{Par(G_x) \mid x \in FV(E \land T)\}$. 

11
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 \([43,48]\). The correctness was presented in \([13,45,48]\).

Procedure \(\text{Par}\) takes a CFG \(G=(V, \Sigma, P, s_0)\) 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 \(\alpha_G\) for this net.

A net \(N\) is a quadruple \(N=(S, T, W, s_0)\) where \(S\) is a set of places, \(T\) is a set of transitions, \(W\) is a weight function: \((S \times T) \cup (T \times S) \to \mathbb{N}\), 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 \(S \to \mathbb{N}\), 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: \(N_G=(V \cup \Sigma, P, W, s_0)\). If \(A \to s\) is a production \(p \in P\) then \(W(A, p)=1\) and \(W(B, p)=0\) 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 \(y_{p_1}, \ldots, y_{p_m}\) and \(n\) variables \(z_{s_1}, \ldots, z_{s_n}\). 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 \(\alpha_G\) is generated through the following two steps. Firstly, the procedure generates a quantifier-free Presburger formula \(\alpha_{\text{count}}\) which constrains the occurrences of letters in words derived by the grammar \(G\).

In particular, \(\alpha_{\text{count}}\) is a conjunction of the following four kinds of subformulas.

- \(x_c \geq 0\) for all \(c \in \Sigma\).
- For each \(X \in V\), let \(p_1, \ldots, p_k\) 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, \(\alpha_{\text{count}}\) contains the following conjunct:
\[M_G(X) + \sum_{p \in P} W(X, p)y_p - \sum_{i=1}^k y_{p_i} = 0\]
- For each \(c \in \Sigma\), \(\alpha_{\text{count}}\) contains the following conjuncts:
\[x_c = \sum_{p \in P} W(c, p)y_p \land (x_c = 0 \lor z_c > 0)\]
- For each \(s \in V \cup \Sigma\), let \(p_1, \ldots, p_l\) be the productions where \(s\) is on the right-hand side and \(X_1, \ldots, X_l\) are their corresponding left-hand sides. Then, \(\alpha_{\text{count}}\) contains the following conjunct:
\[(z_s = 0 \lor \bigvee_{i=1}^{l} (z_s = z_{X_i} + 1 \land y_{p_i} > 0 \land z_{X_i} > 0)\]. If one of the \(X_i\) contains the start symbol \(s_0\), the corresponding disjunct is replaced by \(z_s = 1 \land y_{p_i} > 0\).

Secondly, \(\alpha_G\) is generated as: \(\alpha_G \equiv \exists y_{p_1}, \ldots, y_{p_m}, z_{s_1}, \ldots, z_{s_n}, s_0 \alpha_{\text{count}}\). The correctness of \(\text{Par}\) immediately follows the following theorem.

**Theorem 1** (\([43,48]\)). Given a CFG \(G\), one can compute an existential Presburger formula \(\alpha \equiv \exists y_{p_1}, \ldots, y_{p_m}, z_{s_1}, \ldots, z_{s_n}, s_0 \alpha_{\text{count}}\) for the Parikh image of \(L(G)\) in linear time.

**Example 3.** For the \(EDT0L\) in Ex. 2 we generate the following Parikh-equivalent CFG \(G_1 = (V_1, \Sigma, P_1, S_1)\) where the start symbol \(S_1\) is fresh, \(V_1 = \{S_1, x, x_1, x_2, x_3\}\) and \(P_1 = \{ ((S_1, ax), (x, ax_1), (x_1, bxz), (x_2, x), (x, x_3), (x_3, ax_1), (x_1, \epsilon) \} \).
Next, we show how to compute \( \text{Par}(L(G_{1_v})) \), Parikh image of CFG \( G_{1_v} \). Let \( x_1 \) and \( x_2 \) be integer variables which count the occurrences of letters \( a \) and \( b \), resp., of every word. Let \( y_1, y_2, \ldots, y_7 \) be integer variables representing for each production in \( P_1 \) following the left-right order. And let \( z_a, z_b, z_{S_1}, z_y, z_z, z_x, z_w \) 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 \( \alpha_{\text{count}} \) is: \( x_1 \geq 0 \land x_2 \geq 0 \). The second is:

\[
\begin{align*}
\text{Variable conjunct} & \quad \text{Variable conjunct} \\
x & 1 + (y_4 + y_1) - (y_2 + y_5) = 0 \\
S_1 & 0 + 0 - y_1 = 0 \\
x_1 & 0 + (y_2 + y_6) - (y_3 + y_7) = 0
\end{align*}
\]

The third kind of conjuncts in \( \alpha_{\text{count}} \) corresponding to letter \( a \) and \( b \) is: \( x_1 = y_1 + y_2 + y_6 \land (x_2 = 0 \lor z_2 > 0) \) and \( x_2 = y_1 + y_3 \land (x_2 = 0 \lor z_2 > 0) \), respectively. The fourth is as follows.

\[
\begin{align*}
& x \quad z_x = 0 \lor (z_x = z_{x_2} + 1 \land y_4 > 0 \land z_{x_2} > 0) \lor (z_x = z_{S_1} + 1 \land y_1 > 0 \land z_{S_1} > 0) \\
& S_1 \quad z_{S_1} = 0 \\
& x_1 \quad z_{x_1} > 0 \lor (z_{x_1} = 1 \land y_2 > 0) \lor (z_{x_1} = z_{x_3} + 1 \land y_6 > 0 \land z_{x_3} > 0) \\
& x_2 \quad z_{x_2} > 0 \lor (z_{x_2} = z_{x_1} + 1 \land y_3 > 0 \land z_{x_1} > 0) \\
& x_3 \quad z_{x_3} > 0 \lor (z_{x_3} = 1 \land y_5 > 0) \\
& a \quad z_a > 0 \lor (z_a = z_{S_1} + 1 \land y_1 > 0 \land z_{S_1} > 0) \lor (z_a = 1 \land y_2 > 0) \lor (z_a = z_{x_3} + 1 \land y_6 > 0 \land z_a > 0) \\
& b \quad z_b > 0 \lor (z_b = z_S + 1 \land y_1 > 0 \land z_S > 0) \lor (z_b = z_{x_1} + 1 \land y_3 > 0 \land z_a > 0)
\end{align*}
\]

Then, the length constraint of \( x \) is inferred as:

\[
\begin{align*}
\alpha_{G_{1_v}} & \equiv \exists y_1, \ldots, y_7, z_a, z_b, z_y, z_z, z_x, z_w, |x| = y_3 + 1 \land x_a + x_b \land \alpha_{\text{count}} \\
& \equiv \exists y_1, \ldots, y_7, z_a, z_b, z_y, z_z, z_x, z_w, |x| = 2y_3 + 1 \land x_a = y_3 + 1 \land x_b = y_3 \land \alpha_{\text{count}}
\end{align*}
\]

6.2 STR\text{EDTOL}: A Syntactic Decidable Fragment

**Definition 2** (STR\text{EDTOL} Formulas) \( E \land Y \land \alpha_1 \land \ldots \land \alpha_n \) is called in fragment STR\text{EDTOL} if \( E \) is a quadratic system and \( FV(\alpha_i) \) contains at most one string length \( \forall i \in \{1 \ldots n\} \).

The decidability relies on the termination of \( \omega\)-SAT over quadratic systems.

**Proposition 3.** \( \omega\)-SAT runs in factorial time in the worst case for quadratic systems.

Let \( \text{SAT-STR}[\text{STR\text{EDTOL}}] \) be the satisfiability problem in this fragment. The following theorem immediately follows from Proposition 3 Corollary 4.7 Parikh image of finite-index EDT0L systems 42.

**Theorem 2.** \( \text{SAT-STR}[\text{STR\text{EDTOL}}] \) is decidable.

7 STR\text{flat} Decidable Fragment

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

**Definition 3** The normalized formula \( E \land Y \land \alpha \) is called in the STR\text{flat} fragment if \( \omega\)-SAT takes \( E \) as input, and produces a tree \( T_\alpha \) in a finite time. Furthermore, for every cycle \( C(E_c \rightarrow E_b, \sigma_{\text{cycle}}) \) of \( T_\alpha \), every label along the path \((E_c, E_b)\) is of the form \( [c Y / X] \) where \( X, Y \) are string variables and \( c \) is a letter.

13
This restriction implies that $T_n$ does not contain any nested cycles. We refer such $T_n$ as a flat(able) tree. It further implies that $\sigma_{cyc}$ is of the form $\sigma_{cyc} \equiv [X_1/X, ..., X_k/X']$ and $X'$ 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 $C(E_c \rightarrow E_b, \sigma_{cyc})[X_1, ..., X_k]$.

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.

We assume that the system $P$ includes $n$ unknown (a.k.a. uninterpreted) predicates and $P$ is defined by a set of constrained Horn clauses. We notice that, as shown in Fig. 1, inductive predicates are restricted within arithmetic domain. Every clause is of the form: $\phi_i \Rightarrow P_1(\bar{v}_i)$ where $P_1(\bar{v}_i)$ is the head and $\phi_i$ 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 $\phi^0$. We now introduce $\Gamma$ to denote an interpretation over unknown predicates such that for every $P_i \in P$, $\Gamma(P_i(\bar{v}_i)) \equiv \phi^i$. We use $\phi(\Gamma)$ to denote a formula obtained by replacing all unknown predicates in $\phi$ with their definitions in $\Gamma$. We say a clause $\phi_h \Rightarrow \phi_n$ satisfies if there exists $\Gamma$ and for all stacks $\eta \in \text{Stacks}$, we have $\eta \models \phi_h(\Gamma)$ implies $\eta \models \phi_n(\Gamma)$. A conjunctive set of Horn clauses (CHC for short), denoted by $R$, is satisfied if every constraints in $R$ 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 $\omega$-SAT. Given a tree $T_{n+1} (V, E, C)$ (where $E_0 \in V$ be the root of the tree) deduced from an input $E_0 \cap T$, we generate a system of inductive predicates and CHC $R$ as follows.

1. For every node $E_c \in V$ s.t. $\bar{v}_i = FV(E_c) \neq \emptyset$, we generate an inductive predicate $P_1(\bar{v}_i)$.
2. For every edge $(E_i, \sigma, E_j) \in E, \bar{v}_i = FV(E_i) \neq \emptyset, \bar{v}_j = FV(E_j)$, $\bar{v}_j \cap E$, we generate the clause: $\exists \bar{w}_j, \text{gen}(\sigma) \wedge P_1(\bar{v}_j) \Rightarrow P_1(\bar{v}_j)$ where gen$(\sigma)$ is defined as:

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

3. For every cycle $C(E_c \rightarrow E_b, \sigma_{cyc}) \in C$, we generate the following clause:

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

The length constraint of all solutions of $E_0 \cap T$ is captured by the query: $P_0(FV(E_0))$.

In the following, we show that if $T_n$ 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 [49]. In particular, a system of inductive predicates is in DPI fragment if every predicate $P$ is defined as follows. Either it is constrained by one base clause as: $\phi^b \Rightarrow P(\bar{v})$ or it is defined by two clauses as:

$$\phi^h_1 \wedge ..., \phi^h_m \Rightarrow P(\bar{v}) \quad \exists \bar{w}, \bigwedge \{v_{i} \pm \bar{t}_i = k\} \wedge P(\bar{f}) \Rightarrow P(\bar{v})$$

where $FV(\phi^h_i) \in \bar{v}$ (for all $i \in 1..m$) and has at most one variable; $\bar{f} \subseteq \bar{v} \cup \bar{w}, \bar{v}_i$ 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. 2.

$$
\begin{align*}
\exists n_x, n_x = & n_{x_1} + 1 \land P_{12}(n_x, n_y) \Rightarrow P_0(n_x, n_y) & n_y = 0 \Rightarrow P_{21}(n_y) \\
& n_{x_1} = 0 \land P_{21}(n_y) \Rightarrow P_{12}(n_{x_1}, n_y) & \exists n_{y_1}, n_y = n_{y_1} + 1 \land P_{32}(n_{y_1}) \Rightarrow P_{21}(n_y) \\
\exists n_{x_2}, n_{x_1} = & n_{x_2} + 1 \land P_{22}(n_{x_2}, n_y) \Rightarrow P_{12}(n_{x_1}, n_y) & n_{y_1} = n_y \land P_{21}(n_y) \Rightarrow P_{32}(n_{y_1}) \\
& n_{x_2} = n_x \land P_0(n_x, n_y) \Rightarrow P_{22}(n_{x_2}, n_y) & P_0(n_x, n_y) \land (\exists k. n_x = 4k + 3) \land n_x = 2n_y
\end{align*}
$$

After eliminating the mutual recursion, predicate $P_{21}$ is in the DPI fragment and generated a definitions as: $P_{21}(n_y) \equiv n_y \geq 0$. Similarly, after substituting the definition of $P_{21}$ into the remaining clauses and eliminating the mutual recursion, predicate $P_0$ is in the DPI fragment and generated a definitions as: $P_0(n_x, n_y) \equiv \exists i. n_x = 2i + 1 \land n_y \geq 0$.

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

**Definition 4 (STRflat Formulas)** $\pi = \mathcal{E} \land \mathcal{Y} \land \alpha$ is called in the STRflat fragment if either $\mathcal{E}$ is both quadratic and phased-regular or $\mathcal{E}$ is in SL fragment.

**Proposition 5.** $\omega$-SAT constructs a flat tree for a STRflat constraint in linear time.

Let $\text{SAT-STR}[\text{STRflat}]$ be the satisfiability problem in this fragment.

**Theorem 3.** $\text{SAT-STR}[\text{STRflat}]$ is decidable.

8 Implementation and Evaluation

We have implemented a prototype for Kepler22, 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, $\omega$-SAT may not terminate and Kepler22 may return UNKNOWN. Our SMT-LIB parser is based on the open source [38]. We made use of Z3 [14] as a back-end SMT solver for the linear arithmetic.

**Evaluation** As noted in [22][12], all constraints in the standard Kaluza benchmarks [33] with 50,000+ test cases generated by symbolic execution on JavaScript applications satisfy the straight-line conditions. Therefore, it could not be used to evaluate our proposal that focuses on cyclic constraints. We have generated and experimented Kepler22 over a new set of 600 hand-drafted benchmarks each of which is a phased-regular constraint
Table 1: Experimental Results

| Solver   | #\text{\#SAT} | #\text{\#UNSAT} | #\text{\#XSAT} | #\text{\#XUNSAT} | #\text{\#UNKNOWN} | #timeout | ERR | Time |
|----------|----------------|-----------------|----------------|------------------|--------------------|-----------|-----|------|
| Trau     | 4              | 0               | 302            | 236              | 0                  | 0         | 62  | 37s  |
| S3P      | 5              | 55              | 110            | 1                | 0                  | 100       | 253 | 81   | 801m55s |
| CVC4     | 11             | 120             | 143            | 0                | 69                 | 0         | 268 | 0    | 795m49s |
| Norm     | 23             | 67              | 98             | 0                | 0                  | 342       | 0   | 0    | 336m20s |
| Z3str3   | 5              | 69              | 102            | 0                | 0                  | 292       | 24  | 113  | 77m4s  |
| Z3str2   | 50             | 136             | 66             | 0                | 0                  | 380       | 18  | 0    | 54m35s |
| Kepler22 |                | 298             | 302            | 0                | 0                  | 0         | 0   | 0    | 18m58s |

in the proposed decidable fragment \text{STR}_{\text{flat}}. The set of benchmarks includes 298 satisfiable queries and 302 unsatisfiable queries. Each benchmark has from one to three phases. Each phase is in the form of either $xaby = ybax$ or $xab = bax$ in the case of satisfiable constraints (files quad-*odd*number-*), and $xaay = ybax$ or $xaa = bax$ in the case of unsatisfiable constraints (files quad-*even*number-*), where $x, y$ are string variables and $a, b$ are letters. We have also compared Kepler22 against existing state-of-the-art string solvers: Z3-str2 [51, 50], Z3str3 [10], CVC4 [1], S3P [47], Norn [7, 8], and Trau [6]. All experiments were performed on an Intel Core i7 3.6Gh with 12GB RAM.

The experiments are shown in Table 1. The first column shows the solvers. The column $\#\text{\#SAT}$ (resp., $\#\text{\#UNSAT}$) indicates the number of benchmarks for which the solvers decided $\text{SAT}$ (resp., $\text{UNSAT}$) correctly. The column $\#\text{\#XSAT}$ (resp., $\#\text{\#XUNSAT}$) indicates the number of benchmarks for which the solvers decided $\text{UNSAT}$ on satisfiable queries (resp., $\text{SAT}$ on unsatisfiable queries). The column $\#\text{\#UNKNOWN}$ indicates the number of benchmarks for which the solvers returned unknown, $\text{timeout}$ for which the solvers were unable to decide within 180 seconds, $\text{ERR}$ for internal errors. The column $\text{Time}$ 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 eventually returned either crashes or $\text{UNSAT}$ for all benchmarks. The results also show that Kepler22 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 $e_c \equiv xaay = ybax$ (highlighted in Sect. 1) which is contained in file quad-004-2-unsat of the benchmarks. Kepler22 generates a cyclic proof for $e_c$ with the base case $e_c^1 \lor e_c^2$ where $e_c^1 \equiv e_c[\epsilon/x] \equiv aay = yba$ and $e_c^2 \equiv e_c[\epsilon/y] \equiv xaa = bax$. It is known that for certain words $w_1, w_2$ and a variable
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 \( e_1 \) and \( e_2 \) are unsatisfiable. The soundness of the cyclic proof implies that \( e_c \) is unsatisfiable. For this problem, while \texttt{Kepler}\textsubscript{22} returned \texttt{UNSAT} within 1 second, \texttt{Z3str2} and \texttt{Z3str3} returned \texttt{UNKNOWN}, \texttt{S3P}, \texttt{Norn} and \texttt{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 \cite{Makanin}. In the sequence of papers, Plandowski \textit{et.al.} showed that the complexity of this problem is \textsc{PSPACE} \cite{Plandowski}. The proposed procedure \( \omega \)-\textsc{SAT} is closed to the (more general) problem in computing the set of all solutions for a word equation \cite{Makanin,Plante1,Plante2,Plante3,Plante4}. The algorithm presented in \cite{Makanin} which is based on Makanin’s algorithm does not terminate if the set is infinite. Moreover, the length constraints derived by \cite{Plante1,Plante2} may not be in a finite form. In comparison, due to the consideration of cyclic solutions, \( \omega \)-\textsc{SAT} terminates even for infinite sets of all solutions. The description of the sets of all solutions as EDTOL languages was known \cite{Makanin,Plante1}. For instance, authors in \cite{Makanin} show that the languages of quadratic word equations can be recognized by some pushdown automaton of level 2. Although \cite{Makanin} did not aim at giving such a structural result, it provided \textit{recompression} method which is the foundation for the remarkable procedure in \cite{Makanin} which prove that languages of solution sets of arbitrary word equations are EDTOL. In this work, we propose a decision procedure which is based on the description of solution sets as finite-index EDTOL languages. Like \cite{Makanin}, we also show that sets of all solutions of quadratic word equation are EDTOL languages. In contrast to \cite{Makanin}, 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 \cite{Makanin} is the language of finite index. Furthermore, node of the graph derived by \cite{Makanin} 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 \cite{Schulz,Makanin}. Moreover, Schulz \cite{Schulz} also extends Makanin’s algorithm to a theory of word equations and regular memberships. Recently, \cite{Schulz,Makanin} presents a decision procedure for subset constraints over regular expressions. \cite{Plante1} presents a decision procedure for regular memberships and length constraints. \cite{Plante1} 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. \cite{Plante1} presents a straight-line fragment including word equations and transducer-based functions (e.g., \texttt{replaceAll}) which is incomparable to our decidable fragments. \texttt{Z3str} \cite{Z3str} implements string theory as an extension of \texttt{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 \texttt{Z3’s congruence closure core}. \texttt{Z3str2} \cite{Z3str2} improves \texttt{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 \[32\] communicates with CVC4’s equality solver to exchange information over string. S3P \[47\] 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 Kepler 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 Kepler. 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).

**Acknowledgements.** Anthony W. Lin for the helpful discussion while the author was visiting Oxford University in December 2017. We are grateful for the constructive feedback from the reviewers of POPL’18 and CAV’18.

**References**

1. CVC4-1.5. http://cvc4.cs.stanford.edu/web/. Online; accessed: 20-Jan-2018.
2. Norn. http://user.it.uu.se/~jarst116/norn/. Online; accessed: 20-Dec-2017.
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: Aug-2017.
5. Z3str3. https://sites.google.com/site/z3strsolver/getting-started. accessed: 24-Jan-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. A. V. Aho. Indexed grammars—an extension of context-free grammars. *J. ACM*, 15(4):647–671, Oct. 1968.
10. 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.
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, 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.
32. 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.
33. 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.
34. 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.
35. 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.
36. G. Makanin. The problem of solvability of equations in a free semigroup. *Mathematics of the USSR-Sbornik*, 32(2):129–198, 1977.
37. R. J. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, Oct. 1966.
38. S. Parser. [http://homepage.cs.uiowa.edu/~astump/](http://homepage.cs.uiowa.edu/~astump/).
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 Languages. 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. Makarin’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. Progressive reasoning over recursively-defined strings. In CAV, 2016.
48. K. N. Verma, H. Seidl, and T. Schwentick. On the Complexity of Equational Horn Clauses, pages 337–352. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.
49. 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.
50. 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.
51. 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.

A String Constraints (Cont)

\[
\begin{align*}
\eta, \beta_\eta &\models \pi_1 \lor \pi_2 \iff \eta, \beta_\eta \models \pi_1 \text{ or } \eta, \beta_\eta \models \pi_2 \\
\eta, \beta_\eta &\models \pi_1 \land \pi_2 \iff \eta, \beta_\eta \models \pi_1 \text{ and } \eta, \beta_\eta \models \pi_2 \\
\eta, \beta_\eta &\models \neg \pi_1 \iff \eta, \beta_\eta \not\models \pi_1 \\
\eta, \beta_\eta &\models s \in \mathcal{R} \iff \exists w \in \mathcal{L}(\mathcal{R}) \cdot \eta, \beta_\eta \models s = w \\
\eta, \beta_\eta &\models s_1 = s_2 \iff \eta(s_1) = \eta(s_2) \text{ and } \beta_\eta(s_1) = \beta_\eta(s_2) \\
\eta, \beta_\eta &\models s_1 \neq s_2 \iff \eta(s_1) \neq \eta(s_2) \\
\eta, \beta_\eta &\models a_1 \odot a_2 \iff \eta(a_1) \odot \eta(a_2), \text{ where } \odot \in \{=, \leq\}
\end{align*}
\]

Fig. 7: Semantics

Semantics of String Constraint We show the details of the semantics in Fig. 7. We notice that an equation of string terms is satisfied if there exists an assignment that satisfies word equation over string variables as well as equation over their lengths.
A.1 Normalized Formulas

We show how to normalize word equations and regular expressions in a formula. First, we show how to transform negation over word equations, and disjunction of word equations into an equivalent single word equation. By doing so, it is safe to consider only single word equation in the proposed algorithms. The reader is referred to [15] for the correctness of the transformation. Word disequalities can be eliminated using the following proposition [15].

Proposition 6. A disequality \( s_1 \not= s_2 \) is equivalent with the following formula:

\[
\bigvee_{a \in \Sigma} (s_1 = s_2 \cdot a \cdot x \lor s_2 = s_1 \cdot a \cdot x) \lor \\
\bigvee_{a,b \in \Sigma, a \not= b} (s_1 = x \cdot a \cdot y \land s_2 = x \cdot b \cdot z)
\]

where \( x, y \) and \( z \) are fresh variables.

Intuitively, two string terms \( s_1 \) and \( s_2 \) are different if there exists an interpretation \( \eta \) and a non-negative number \( i \) such that the letters at position \( i \) in \( \eta(s_1) \) and \( \eta(s_2) \) are different. This elimination is utilized Norn solver [7].

A disjunction of word equations can be replaced by a single word equation as follows.

Proposition 7. Let \( a,b \in \Sigma \) be distinct letters and \( a \not= b \). A disjunction of two word equations is equivalent with a single word equation in two extra unknowns.

Next, we show how to remove the negation and the concatenation operator over regular expression. It is easy to show that the negation of a membership predicate in a regular expression is equivalent \( R \) with a membership predicate in its complement \( R^C \).

Lemma 1. Let \( s \) be a string term and \( R \) a regular expression. Then, \( \neg(s \in R) \equiv (s \not\in R^C) \).

We note that either the expression \( w \in R \) or the expression \( \neg(w \in R) \) where \( w \in \Sigma^* \) is trivially evaluated and replaced by true or false. Removing the concatenation operator in the expression \( s_1 \cdot s_2 \in R \) relies on the following function. Let \( L \) be a regular language and \( f(L) \) is a set of pairs of DFAs \((D_1, D_2)\) which represent for regular languages \((L_1, L_2)\) such that \( L = L_1 \cdot L_2 \). To compute the set \( f(L) \) of a given regular language \( L \), represent \( L \) by some fixed automaton \((Q, \Sigma, \delta, s_0, Q_F)\). For any state \( q_i \in Q \), we generate two automata \( D_1 \) form \((Q, \Sigma, \delta, s_0, \{s_i\})\) and \( D_2 \) \((Q, \Sigma, \delta, s_i, Q_F)\), respectively. Then, \( \forall w \in L \) and \( w = w_1 \cdot w_2 \), there exists a state \( s_i \) to form such two automata which, in turn, generate two corresponding languages \( L_1, L_2 \) such that \( w_1 \in L_1 \) and \( w_2 \in L_2 \). Let \( DFA2RE \) be the function to convert a DFA to a regular expression. Then, the following lemma is straightforward.

Lemma 2. Let \( s_1, s_2 \) be string terms and \( R \) a regular expression. Then,

\[
(s_1 \cdot s_2 \in R) \equiv \bigvee \{ s_1 \in DFA2RE(D_1) \land s_2 \in DFA2RE(D_2) \mid (D_1, D_2) \in f(L(R)) \}
\]
B Correctness of $\omega$-SAT - Proposition\([1]\)

The correctness of procedure $\omega$-SAT replies on the correctness of $\text{match}$, $\text{complete}$ and the soundness of the cyclic proofs where all leaf nodes are marked as closed.

Procedure $\text{match}$ First, we show that $\text{match}$ produces an equi-satisfiable word equation.

**Lemma 3 (Matching).** Suppose that $e$ is a word equation, and $e'=\text{match}(e)$. Then, a) if $e$ is satisfiable, so is $e'$. b) if $e'$ is satisfiable, so is $e$. c) in both cases a) and b), $e'$ is a suffix of $e$.

Function $\text{match}(e)$ also has the following property.

**Lemma 4.** Let $e'(N')=\text{match}(e(N))$. Then, $N' \leq N$.

Procedure $\text{complete}$ Next, we show that $\text{complete}$ produces an equi-satisfiable set of word equations. We remark that procedure $\text{complete}$ also produces substitutions labeled along path traces which help to construct a model (assignments to string variables) for satisfiable inputs.

**Lemma 5 (Complete).** Suppose that $e$ is a word equation, and $L$ is set of pairs of word equation and substitutions such that $L=\text{complete}(e)$. Then,

- C1) if $e$ is satisfiable then there exists a pair $(e', \sigma) \in L$ such that $e'$ is satisfiable.
- C2) if there exists a pair $(e', \sigma) \in L$ such that $e'$ is satisfiable, then $e$ is satisfiable.

Cyclic Proofs Finally, we consider the case where the input is unsatisfiable. Suppose that $\omega$-SAT takes a word equation $e$ as input, and produces a cyclic reduction tree $T_n$ as output in a finite time. If all leaf nodes of $T_n$ is unsatisfiable, then following Lemma 3 and Lemma 5 we can conclude that $e$ is unsatisfiable. Now, we study the scenarios where some leaf nodes of $T_n$ is unsatisfiable and the remaining leaf nodes are linked back. We refer to such reduction tree $T_n$ as cyclic proofs. In the following, we show that if $\omega$-SAT can derive sound cyclic proofs for a word equation $e$, then $e$ is unsatisfiable. The following formalism is based on the generic framework S2SAT \([30,31]\). In contrast to \([30,31]\), our soundness proof is based on the fact that solutions of a word equation must be finite.

**Definition 5 (Pre-proof)** A pre-proof derived for an equation $e$ is a pair $(T_i, L)$ where $T_i$ is an unfolding tree whose root labelled by $e$ and $L$ is a back-link function assigning some leaf nodes $e_c$ of $T_i$ to interior nodes $e_c = L(e_b)$ such that there exists some substitution $\theta$ i.e., $e_c = e_b[\theta]$.

We recap that in the above definition $e_b$ is referred as a bud and $e_c$ is referred as its companion.

A cycle path in a pre-proof is a sequence of nodes $(e_i)_{i \geq 0}$.

**Definition 6 (Cycle Trace)** Let $(e_i)_{i \geq 0}$ be a cycle path in a pre-proof $PP$. A cycle trace following $(e_i)_{i \geq 0}$ is a sequence $(\alpha_i)_{i \geq 0}$ such that, for all $i \geq 0$, $\alpha_i$ is a string variable $x$ in the formula $e_i$, and either:
1. $\alpha_{i+1}$ is the variable $x$ occurrence in $e_{i+1}$, or
2. $\alpha_{i+1}$ is the subformula $c \cdot x'$ (where $c$ is a letter) according to $x$ in $e_{i+1}$ (i.e., $e_{i+1} = e_i[c \cdot x'/x]$) and $i$ is a progressing point of the trace, or
3. $\alpha_{i+1}$ is the subformula $y \cdot x'$ (where $y$ is a string variable) according to $x$ in $e_{i+1}$ (i.e., $e_{i+1} = e_i[y \cdot x'/x]$) and $i$ is a progressing point of the trace.

To ensure that pre-proofs correspond to sound proofs, a global soundness condition must be imposed on such pre-proofs as follows.

**Definition 7 (Cyclic proof)** A pre-proof is a cyclic proof if, for every infinite path $(e_i)_{i \geq 0}$, there is a tail of the cycle path $p=(e_i)_{i \geq n}$ s.t. there is an infinitely progressing trace following $p$.

**Lemma 6 (Soundness).** If there is a cyclic proof of $e$, $e$ is unsatisfiable.

The correctness of Proposition 1 immediately follows the following Lemma 3, Lemma 5 and Lemma 6.

**B.1 Proof of Lemma 3**

**Proof** We prove this lemma through the following three cases.

1. Case 1: $e \equiv c \cdot tr_1 = c \cdot tr_2$ where $c$ is a letter, then $e' \equiv tr_1 = tr_2$. The proof is as follows.

   - $e$ is satisfiable
   - $\Leftrightarrow$ there exists an assignment $\eta \in SStacks$ such that $\eta \models c \cdot tr_1 = c \cdot tr_2$
   - $\Leftrightarrow$ $\eta(c \cdot tr_1) = \eta(c \cdot tr_2)$ meaning of $\eta$ relation
   - $\Leftrightarrow$ $\eta(c) \cdot \eta(tr_1) = \eta(c) \cdot \eta(tr_2)$ meaning of $\eta$ on concatenation
   - $\Leftrightarrow$ $\eta(tr_1) = \eta(tr_2)$ meaning of concatenation
   - $\Leftrightarrow$ there exists an assignment $\eta \in SStacks$ such that $\eta \models tr_1 = tr_2$ meaning of $\eta$
   - $\Leftrightarrow$ $e'$ is satisfiable

   Furthermore, if $S$ and $S'$ are solution words of $e$ and $e'$, respectively, we can imply that $S = c \cdot S'$. Hence, $e'$ is a suffix of $e$.

2. Case 2: $e \equiv utr_1 = u tr_2$ where $u$ is a string variable, then $e' \equiv tr_1 = tr_2$. We consider two sub-cases.

   - Sub-case 2.1: $u \in (FV(tr_1) \cup FV(tr_2))$. The proof for this sub-case is similar to the proof in Case 1.
   - Sub-case 2.2: $u \notin (FV(tr_1) \cup FV(tr_2))$. The proof for Case a) of this sub-case is similar to the proof in Case 1. In the following, we show the proof for Case b).

   - $e'$ is satisfiable
   - $\Leftrightarrow$ there exists an assignment $\eta \in SStacks$ such that $\eta \models tr_1 = tr_2$
   - $\Leftrightarrow$ $\eta(tr_1) = \eta(tr_2)$ meaning of $\eta$

23
We create a new assignment $\eta'$ such that i) for all $v \in (FV(tr_1) \cup FV(tr_2))$, $\eta'(v) = \eta(v)$ and ii) $\eta'(u) = e$. From i), we have:

\[
\eta'(tr_1) = \eta'(tr_2) \Rightarrow \eta'(u) \cdot \eta'(tr_1) = \eta'(u) \cdot \eta'(tr_2) \quad \text{from ii)}
\]

\[
\eta'(u \cdot tr_1) = \eta'(u \cdot tr_2) \quad \text{meaning of $\eta'$ on concatenation}
\]

\[
\Rightarrow \text{there exists an relation $\eta' \in SStacks$
\]

\[
\text{such that } \eta'|=u \cdot tr_1=u \cdot tr_2 \quad \text{meaning of $\eta'$}
\]

\[
\Rightarrow \text{e is satisfiable}
\]

Furthermore, if $S'$ and $S$ are solution words of $e'$ and $e$, respectively, we can imply that $S=S'$. Hence, $e'$ is a suffix of $e$.

3. Case 3: $e\equiv u_1tr_1=u_2tr_2$ and $e'\equiv e$ where $u_1$, $u_2$ are either letters or string variables and $u_1$, $u_2$ are different. The proof for this case is straightforward.

\[
\square
\]

### B.2 Proof of Lemma 4

**Proof** Based on the definition of match, we consider two following cases.

1. M1) $e \equiv utr_1 = utr_2$ (where $u$ is a string variable or a letter), then $e' \equiv tr_1 = tr_2$.
   It is easy to show that $N' = N - 2$.
2. M2) $e \equiv u_1tr_1 = u_2tr_2$ (where $u_1$ and $u_2$ are string variables or letters), then $e' \equiv e$. Hence, $N' = N$.

\[
\square
\]

### B.3 Proof of Lemma 5

**Proof** Based on the definition of complete procedure, we prove this lemma (both C1) and C2) through following two cases.

1. $e\equiv x_1tr_1 = c_2tr_2$, then $L_i = \{(e_i, \sigma_1); (e_i, \sigma_2)\}$ where $\rho_1|=/\{x_1\}, e_i\equiv (x_1tr_1=c_2tr_2)\rho_1, \rho_2|=c_2x'_1/x_1$ (where $x'_1$ is a fresh variable) and $e_i\equiv (x_1tr_1=c_2tr_2)\rho_2$.
   We start with $e$ is satisfiable.

   \[
   e \text{ is satisfiable}
   \]

   \[
   \Leftrightarrow \text{there exists an assignment } \eta \in SStacks
   \]

   \[
   \text{such that } \eta|x_1\cdot tr_1=c_2\cdot tr_2
   \]

   \[
   \Leftrightarrow \eta(x_1\cdot tr_1) = \eta(c_2\cdot tr_2) \quad \text{meaning of $\eta$ relation}
   \]

   \[
   \Leftrightarrow \eta(x_1) \cdot \eta(tr_1) = \eta(c_2) \cdot \eta(tr_2) \quad \text{meaning of $\eta$ on concatenation}
   \]

   \[
   \Leftrightarrow \eta(x_1) \cdot \eta(tr_1) = c_2 \cdot \eta(tr_2) \quad \text{meaning of $\eta$ on letters (a1)}
   \]

   Now, we do case split on $\eta(x_1)$: $\eta(x_1)=\epsilon$ or $\eta(x_1)=u_1$ and $u_1\neq\epsilon$.

   **Sub-case 1.1:** $\eta(x_1)=\epsilon$. We create new assignment $\eta'$ such that: $\eta'(x_1)$ is not defined and $\eta'(v)=\eta(v) \forall v \in (FV(tr_1) \cup FV(tr_1) \setminus \{x_1\})$. It is easy to show that
\[ \eta(tr_1) \equiv \eta'(tr_1 \sigma_1) \text{ and } \eta(tr_2) \equiv \eta'(tr_2 \sigma_1) (a_2). \] From \((a_1)\) and \((a_2)\), we obtain:

\[
\begin{align*}
\eta'(tr_1 \sigma_1) &= c_2 \cdot \eta'(tr_2 \sigma_1) \\
\iff \eta'(tr_1 \sigma_1) &= \eta'(c_2 \cdot tr_2 \sigma_1) & \text{meaning of } \eta' \text{ on letters} \\
\iff \eta'((x_1 \cdot tr_1)\sigma_1) &= \eta'((c_2 \cdot tr_2)\sigma_1) & \text{substitution does not affect constants} \\
\iff \eta'((x_1 \cdot tr_1)\sigma_1) &= \eta'((c_2 \cdot tr_2)\sigma_1) \\
\iff \eta'|(x_1 \cdot tr_1)\sigma_1| &= (c_2 \cdot tr_2)\sigma_1 \\
\iff \eta'|(x_1 \cdot tr_1|=c_2 \cdot tr_2)\sigma_1 \\
e_i \text{ is satisfiable}
\end{align*}
\]

We can conclude that there exists \((e_i, \sigma_1) \in L_i\) such that \(e_i\) is satisfiable.

**Sub-case 1.2:** \(\eta(x_1) = w\) and \(w \neq e\). Substituting into \((a_1)\) to obtain: \(w_1 \cdot \eta(tr_1) = c_2 \cdot \eta(tr_2)\). This implies that \(w_1 = c_2 \cdot w_1'\). Hence, \(w_1 = c_2 \cdot w_1' \cdot \eta(tr_1) = c_2 \cdot \eta(tr_2)\) \((a_3)\).

We create new assignment \(\eta'\) such that: \(\eta'(x_1)\) is not defined, \(\eta'(v) = \eta(v) \forall v \in (FV(tr_1) \cup FV(tr_2) \setminus \{x_1\})\), and \(\eta'(x_1') = w_1'\). It is easy to show that \(\eta(tr_1) \equiv \eta'(tr_1 \sigma_2)\) and \(\eta(tr_2) \equiv \eta'(tr_2 \sigma_2)\) \((a_4)\). From \((a_3)\) and \((a_4)\), we have:

\[
\begin{align*}
&c_2 \cdot w_1' \cdot \eta'(tr_1 \sigma_2) = c_2 \cdot \eta'(tr_2 \sigma_2) \\
\iff &c_2 \cdot \eta(x_1') \cdot \eta'(tr_1 \sigma_2) = c_2 \cdot \eta'(tr_2 \sigma_2) \\
\iff &\eta'(c_2 \cdot x_1' \cdot tr_1 \sigma_2) = \eta'(c_2 \cdot tr_2 \sigma_2) \\
\iff &\eta'|(c_2 \cdot x_1' \cdot tr_1)\sigma_2| = (c_2 \cdot tr_2)\sigma_2 \\
e_i \text{ is satisfiable}
\end{align*}
\]

We can conclude that there exists \((e_i, \sigma_2) \in L_i\) such that \(e_i\) is satisfiable.

2. \(e \equiv x_1tr_1 = x_2tr_2\), then \(L_i = \{(e_i, \sigma_1); (e_i, \sigma_2); (e_i, \sigma_3); (e_i, \sigma_4)\}\). The proof is similar to the case above.

\[ \square \]

**B.4 Proof of Lemma 6**

**Proof** We prove this lemma by contradiction. Assume there is a cyclic proof \(\mathcal{P}\) of \(\varepsilon\) and \(\varepsilon\) is satisfiable. As the lengths of the solution words are finite, the lengths of paths starting from the root to satisfiable leaf nodes must be finite.

By following Lemma \[5\] and Lemma \[5\] we would be able to construct an infinite path \((e_i)_{i \geq 0}\) in \(\mathcal{P}\) such that \(e_0 \equiv \varepsilon\) and \(e_i\) is satisfiable for all \(i \geq 0\). Since \(\mathcal{P}\) is a cyclic proof, there exists an \(n \geq 0\) and a tail of the path, \(p = (e_i)_{i \geq n}\), such that there is an infinitely progressing trace following \(p\) (Definition \[7\]). This contradicts the fact of finite path for solutions of a word equation above.

\[ \square \]

**C Correctness of widetree(\(\mathcal{T}_n, \mathcal{Y}\)) - Proposition**

We show correctness of extended tree representing for all solutions of the conjunction of a word equation, say \(\varepsilon\), and regular expressions, say \(\mathcal{Y}\). The computation of \(m\) and
$M$ is based on the proof presented by Schulz in [44] to find the minimal solutions of the constraint $e \land T$. First, we transform the constraint on regular expressions into constraints over DFA. It is well known that there exists a DFA that accepts the same language with a conjunction of regular expressions [25].

Suppose that we are given a DFA: $A = \langle Q, \Sigma, \delta, q_0, Q_f \rangle$. For any pair $(q_i, q_j)$ in $Q$, $L(A^* q_i)$ denotes the language which is accepted by the automaton: $A^* q_i = \langle Q, \Sigma, \delta, q_i, \{q_j\} \rangle$. Further, we define $L(A, \emptyset) = \Sigma^*$ and $L(A, \Gamma) = \bigcap_{(q_i, q_j) \in Q \times Q} L(A^* q_i)$ where $\Gamma \neq \emptyset$ and $\Gamma \subseteq Q \times Q$. An $A - constraint$ is a finite set $\Gamma$ of pairs $(q_i, q_j) \in Q \times Q$. Given a constraint $\pi \equiv e \land X_1 \in \mathcal{R}_1 \land \ldots \land X_n \in \mathcal{R}_n$ over $\Sigma$ where $X_1, \ldots, X_n \in FV(e)$, It is obvious that we can find a DFA $A$ and $A - constraints \Gamma_1, \ldots, \Gamma_n$ such that a word $w_i \in \Sigma^*$ is an assignment for $X_i$ of a solution of $\pi$ if $w_i \in L(A, \Gamma_i)$ ($1 \leq i \leq n$). Let $m$ be the number states of $A$ and $M = m!$.

**Definition 8** The natural number $t$ and $t'$ are called $A$-equivalent (we write $t \equiv_A t'$) if the following two conditions hold:

1. $t \equiv t' \mod M$,
2. $t > m$ if and only if $t' > m$.

**Lemma 7** ([44]). Let $u \equiv u_1 w^{t_1} u_2 w^{t_2} \ldots u_k w^{t_k} u_{k+1}$ and $u' \equiv u_1 w'^{t_1} u_2 w'^{t_2} \ldots u_k w'^{t_k} u_{k+1}$ ($k \geq 1$) be two words over the alphabet $\Sigma$ and $\Gamma$ is any $A$-constraint. If $t_i \equiv_A t'_i$ ($1 \leq i \leq k$), then $v \in L(A, \Gamma)$ if and only if $v' \in L(A, \Gamma)$.

**C.1 Proof of Proposition 2**

**Proof** An assignment for a variable of a solution obtained from labels including any cycle in the resulting tree is a word $v \equiv u_1 w^t u_2 w^t \ldots u_k w^t u_{k+1}$ where $t_i > m$ and $t_i \mod M = 0$. Following Lemma 7 a word $v$ obtained from labels $\sigma$ including cycles in the resulting tree is of a solution if and only if the word $v$ obtained from $\sigma \setminus \sigma_c$ (where $\sigma_c$ are labels obtained from cycles) is of a solution. □

**D** $EDT0L$ Languages

**D.1 L Systems and Finite-Index $EDT0L$ Language**

$EDT0L$ language is a subclass of indexed languages (denoted as $IND$) in the sense of Aho [9]. In [9], Aho shows that the class of indexed languages includes all context-free languages and some context-sensitive languages, but yet is a proper subset of the class of context-sensitive languages. Fig. 8 shows the containment relationships among classes of indexed languages where $L(CS)$ denotes the class of all context-sensitive languages, and $L(RE)$ the class of all regular languages. In the name $EDT0L$, each capital letter has a standard meaning in connection with $L$ systems [41]. Thus, $L$ refers to parallel rewriting. The character $0$ means that information between individual letters is zero-sided, and $D$ (deterministic) that, for each configuration (a letter in a context), there is only one rule. The letter $T$ (tables) means that the rules are divided into subsets. In each derivation step, rules from the same subset have to be used; and $D$ in this context means that each subset is deterministic. Finally, $E$ (extended) means $L$ is intersected with $\Sigma^*$. 

26
D.2 Proof of Corollary 4.1

Proof  Given a trimmed reduction tree $T_n$, function $\text{extract\_edt1}$ constructs for it a $\text{ET0L}$ grammar $G = \langle V, \Sigma, P, S \rangle$ as follows. $\Sigma$ is the alphabet. $S$ is a fresh variable which does not appear in the tree. $V$ is the union of the set of all variables appearing in the tree and the set $\{S\}$. For each path $(v_r, v_l_i)$ in the trimmed $T_n$ where $v_r$ is the root and $v_l_i$ is either a satisfiable leaf node or a bud of a cycle, we create a new table $P_i$ as:

$$P_i = \{S \rightarrow s_1\} \cup \bigcup \{X \rightarrow s \mid (E, [s/X], E') \text{ in } (v_r, v_l_i)\}$$

Assume that we create $m$ such tables: $P_1, \ldots, P_m$. Then, $P = \{P_1, \ldots, P_m\}$.

Moreover, as each table $P_i$ for all $i \in \{1, \ldots, m\}$ corresponds to a path of the tree, the rule in $P_i$ is deterministic. Hence $G$ is a $\text{EDT0L}$ system.

Finally, let $k$ be the maximum of the lengths of all nodes in the trimmed $T_n$. Then, for every node $e_i(N_i)$ in the trimmed $T_n$, $N_i \leq k$. As so, the number of variables appearing in every node in the tree is less than or equal to $k$. Hence the language generated by $G$ is finite index. \hfill $\Box$

E Proof of Proposition 3

Lemma 8. Let $e$ be a quadratic word equation and $e' = \text{match}(e)$. Then, $e'$ is also a quadratic word equation.

Lemma 9. Let $e(N)$ be a quadratic word equation and $L = \text{reduce}(e)$. Then, for each $(e'(N'), \sigma) \in L$, the following two properties are true:

- $P1$. $e'$ is unsatisfiable or a quadratic equation.
- $P2$. $N' \leq N$.  

27
Lemma 10. Let $e(N)$ be a quadratic word equation. And $\omega$-$SAT$ generates for it a reduction tree $T_n$ in finite time. For every node $e'(N')$ in $T_n$, $e'$ is unsatisfiable or a quadratic word equation and $N' \leq N$.

E.1 Proof of Lemma 8

Proof Based on the definition of $match$, we consider two following cases.

\begin{itemize}
  \item $e \equiv untr_1 = untr_2$ (where $u$ is a string variable or a letter), then $e' \equiv tr_1 = tr_2$. As $e$ is a quadratic word equation, every variable in $FV(tr_1) \cup FV(tr_2)$ occurs at most twice in $tr_1 = tr_2$. Hence, $e'$ is a quadratic word equation.
  \item M2) $e \equiv u_1tr_1 = u_2tr_2$ (where $u_1$ and $u_2$ are string variables or letters), then $e' \equiv e$. Trivially.
\end{itemize}

\hfill $\square$

E.2 Proof of Lemma 9

Proof We consider two following cases based on the definition of procedure $complete$.

\begin{enumerate}
  \item $e(N) \equiv x_1tr_1 = c_2tr_2$, then $L_i = complete(e) \equiv \{(e_1, (N_1), \sigma_1); (e_2, (N_2), \sigma_2)\}$ where $e_1 \equiv (x_1tr_1 = c_2tr_2)\rho_1, \rho_1 = x/x_1, e_2 \equiv (x_1tr_1 = c_2tr_2)\rho_2 (x_i' \text{ is a fresh variable})$ and $\rho_2 = c_2x_i'/x_1$. Let $e_1(N_1') \equiv match(e_1)$.
    \begin{itemize}
      \item P1. As every variable in $FV(e) \setminus \{x_1\}$ occurs at most twice, $e_1$ is a quadratic word equation. And following Lemma 8, $e_1$ is also a quadratic word equation.
      \item P2. As in $e_1$, $x$ is substituted by $c$, $N_1 \leq N - 1$. And following Lemma 4, $N_1' \leq N - 1$. In consequence, $N_1' \leq N - 1$.
    \end{itemize}

  \item $e_2 \equiv (x_1tr_1 = c_2tr_2)[c_2x_i'/x_1].$ Let $e_2(N_2') \equiv match(e_2)$.
    \begin{itemize}
      \item P1. As $e$ is a quadratic equation, $x_1$ as well as every variable in $FV(tr_1 = tr_2) \setminus \{x_1\}$ occurs at most twice in $e$. Hence, $x_i'$ as well as $FV(tr_1 = tr_2) \setminus \{x_1\}$ occurs at most twice in $e_1$. In consequence, $e_2$ is a quadratic equation. And following Lemma 8, $e_2$ is also a quadratic word equation.
      \item P2. As $e$ is a quadratic equation, $x_1$ occurs at most twice in $e$. Hence, $N_2 \leq N + 2$. Then, $e_2(N_2') \equiv match(c_2x_i' = c_2tr_2)[c_2x_i'/x_1] \equiv match(x_i' = tr_2)[c_2x_i'/x_1]$. Thus, $N_2' \leq N_2 - 2$. In consequence, $N_2' \leq N$.
    \end{itemize}
\end{enumerate}

2. $e \equiv x_1tr_1 = x_2tr_2$, then $L_i = complete(e) \equiv \{(e_1, \sigma_1); (e_2, \sigma_2); (e_3, \sigma_3); (e_4, \sigma_4)\}$. The proof is similar to the case above.

\hfill $\square$
E.3 Proof of Proposition 3

**Proof** We show the following property, called PathLength property: Let $e(N)$ be a quadratic word equation, then the length of every path in a reduction tree whose root is $e(N)$ is $O(N^2(N!))$.

We prove the PathLength property by structural induction on $N$.

**Base Case**: $N=1$. Trivially.

**Induction Case**: Assume that PathLength property holds for all $N \leq k$. We now prove that PathLength property holds for $N = k + 1$. Consider a path with the sequence of nodes: $e(k + 1), e_1(N_1), \ldots, e_i(N_i)$ where $e_i(N_i)$ is a leaf node. According to Lemma 10, we consider two following subcases.

1. All these nodes have the same length with $e: k + 1 = N_1 = \ldots = N_M$. There are $O((k+1)!)$ possibilities to arrange a sequence of $N$ symbols of the respective either string variables or characters. And there is $O(k)$ possibilities to put symbol $=$ into one sequence of $k + 1$ symbols above. Thus, we have $O(k(k + 1)!)$ possibilities. In other words, the length of this path is $O((k+1)(k+1)!))$. As $(k+1)^2 = k^2 + 2k + 1$ and $k + 1 < k^2 + 2k + 1$ for all $k \geq 1$, PathLength property holds.

2. There exists a node $e_M(N_M) (1 \leq M < l)$ such that (i) all nodes $e_j(N_j)$ where $j \in \{1, \ldots, M\}$ have the same length with $e: k + 1 = N_1 = \ldots = N_M$ and (ii) node $e_{M+1}(N_{M+1})$ has the smaller length than $e$. By induction, the length of the path $(e, e_i)$ is $O(k^2(k)!))$. Similar to the previous sub-case, the length of the path $(e, e_M)$ is $O(k(k + 1)!)$. Hence, the length of the path $(e, e_i)$ is: $O(k^2(k)!)) + O(k(k + 1)!) = O((k^2/(k + 1) + k(k + 1)!))$ As $(k + 1)^2 = k^2 + 2k + 1$ and $k^2/(k + 1) + k < k^2 + 2k + 1$ for all $k \geq 1$, PathLength property holds.

□

F Correctness of $\text{ST}^{\text{dec}}_{\text{flat}}$ Decidable Fragment

F.1 Straight-Line Formula

**Lemma 11.** Let $e(N)$ be an acyclic word equation. Then, $\omega$-$\text{SAT}$ takes $e(N)$ as input and runs in $O(N)$ time to construct a cyclic reduction tree $T_n$. Furthermore, $T_n$ does not contain any cycle.

**Proof** Let $e(N)$ be a quadratic word equation and $L=\text{complete}(e)$. It is easy to show that for all $e'(N') \in L$, $N' \leq N-1$. Let $e(N)$ be a quadratic word equation and $L=\text{reduce}(e)$. Then, together with Lemma 4 we have: for all $e'(N') \in L$, $N' \leq N-1$.

As each step, $\omega$-$\text{SAT}$ reduces the length of word equation by at least one, the length of each path in the reduction tree is at most $O(N)$.

Furthermore, as the lengths of children are always less than their parents, function $\text{link\_back}$ never successfully links a child back to a interior node. Thus, the tree has no cycle.

□
F.2 Base Case of Proposition

**Lemma 12.** Suppose that $\mathcal{E}$ is a solvable regular word equation. Then, $\omega$-$\text{SAT}$ takes $\mathcal{E}(N)$ as input, and produces a cyclic reduction tree $\mathcal{T}_n$ in a finite time. Furthermore, for any cycle $C(\mathcal{E}_c \to \mathcal{E}_b, \sigma_{\text{cyc}})$ of $\mathcal{T}_n$, both three following properties hold:

- The labels along the path $(\mathcal{E}_c, \mathcal{E}_b)$ (assume that this path has $k$ edges) is of the form: 
  $[c_1 X_1/X], [c_2 X_2/X_1], ..., [c_k X_k/X_{k-1}]$ where $X, X_i$ ($i \in \{1, ..., k+1\}$) are string variables and $c_i$ ($i \in \{1, ..., k+1\}$) is a letter.

- The substitution $\sigma_{\text{cyc}} = [X/X_{k-1}]$.

- The length of each path in the tree is $O(N)$.

**Proof** We prove this lemma by structural induction on $n$, the number of duals in the input. We note that in our following proof, $s_1$, $s_2$, $s_3$ and $s_4$ are string terms and $X$, $Y$ are string variables.

**Case n=0.** The truth of this case is shown by Lemma 11.

**Case n=1.** Wlog, assume that $\mathcal{E} \equiv Xs_1 = s_2Xs_3$ and $X$ does not occur in $s_1$, $s_2$, $s_3$. We consider two following cases.

1. $s_2 \equiv as_1^2$ where $a$ is a letter. Then, $\mathcal{E} \equiv Xs_1 = as_1^2Xs_3$ where all variables in $s_1$, $s_1^2$ and $s_3$ occur at most once. Following the definition of function $\text{complete}$, $\mathcal{E}$ has two children $e_1$ and $e_2$ as follows.
   a. $e_1 \equiv \mathcal{E}[\epsilon/X]$. As all variables in $e_1$ occur at most one, there is no cycle in the subtree with the root is $e_1$ (Lemma 11).
   b. $e_2 \equiv \mathcal{E}[aX_1/X] \equiv X_1s_1 = s_2^2aX_1s_3$.

2. $s_2 \equiv Ys_1^2$ where $Y$ is a string variable. $\mathcal{E} \equiv Xs_1 = Ys_2^2Xs_3$ where $Y$ and all variables in $s_1$, $s_1^2$ and $s_3$ occur at most once. Following the definition of function $\text{complete}$, $\mathcal{E}$ has four children $e_1$, $e_2$, $e_3$ and $e_4$ as follows.
   a. $e_1 \equiv \mathcal{E}[\epsilon/X]$. Similar to Case 1a).
   b. $e_2 \equiv \mathcal{E}[YX_1/X] \equiv X_1s_1 = s_1^2YX_1s_3$.
   c. $e_3 \equiv \mathcal{E}[\epsilon/Y]$. By induction.
   d. $e_4 \equiv \mathcal{E}[YX_1/Y] \equiv s_1 = Ys_1^2Xs_3$. Now, all variables in $e_4$ occurs at most one. Similar to Case 1a).

If Case 1b) or Case 2b) are kept applying, after $k = |s_2|$ times the node generated is $e_k \equiv X_{k+1}s_1 = s_2X_{k+1}s_3$. Then, function $\text{link}_{\text{back}}$ links $e_k$ back to $\mathcal{E}$ to form a cyclic proof. It is easy to check that the Lemma holds for this scenario.

**Case n=2.** $\mathcal{E} \equiv Xs_1Y = Ys_2X$ where $X$ and $Y$ do not occur in $s_1$, $s_2$ and each variable in $s_1$, $s_2$ occurs at most once. We consider following two cases. Following the definition of function $\text{complete}$, $\mathcal{E}$ has four children $e_1$, $e_2$, $e_3$ and $e_4$ as follows.

1. $e_1 \equiv \mathcal{E}[\epsilon/X]$. By Case n=1.
2. $e_2 \equiv \mathcal{E}[YX_1/X] \equiv X_1s_1Y = s_2YX_1$. Consider two following subcases.
   a. $s_2 \equiv as_1^2$ and $e_2 \equiv X_1s_1Y = as_1^2YX_1$ where $a$ is a letter. Then, following the definition of function $\text{complete}$, $e_2$ has two children:
      i. $e_{21} \equiv (X_1s_1Y = as_2^2YX_1)[\epsilon/X_1] \equiv s_1Y = as_1^2Y$. Case n=1.
      ii. $e_{22} \equiv (X_1s_1Y = as_2^2YX_1)[aX_2/X_1] \equiv X_2s_1Y = s_1^2Y\epsilon X_2$.
   b. $s_2 \equiv Zs_1^2$ and $e_2 \equiv X_1s_1Y = Zs_1^2YX_1$ where $Z$ is a string variable. Then, following the definition of function $\text{complete}$, $e_2$ has four children:
i. \( e_{21} \equiv (X_1s_1Y = Zs_2^1YX_1)[c/X_1] \equiv s_1Y = Zs_2^1Y. \) Case \( n=1. \)

ii. \( e_{22} \equiv (X_1s_1Y = Zs_2^1YX_1)[ZX_2/X_1] \equiv X_2s_1Y = s_1^2YZX_2. \)

iii. \( e_{23} \equiv (X_1s_1Y = Zs_2^1YX_1)[c/Z] \equiv X_1s_1Y = s_1^2YX_1. \)

iv. \( e_{24} \equiv (X_1s_1Y = Zs_2^1YX_1)[X_1Z_1/Z] \equiv s_1Y = Zs_2^1YX_1. \) Case \( n=1. \)

3. \( e_3 \equiv e[c/Y]. \) By Case \( n=1. \)

4. \( e_4 \equiv e[XY_1/Y] \equiv s_1XY_1 = Y_1s_2X. \) Similar to Case 2.

If Case 2.b.ii) or Case 2.b.iv) are kept applying, after \( k = |s_2| \) times the node generated is \( e_k \equiv X_{k+1}s_1Y = Ys_2X_{k+1}. \) Then, function \( \text{link}_{\text{back}} \) links \( e_k \) back to \( e \) to form a cyclic proof. It is easy to check that the Lemma holds for this scenario. It is similar to Case 4. We remark that Case 2 and Case 4 are never applied in an interleaving sequence.

\[\square\]

F.3 Proof of Proposition 5

**Proof** Wlog, we assume that \( E \) is a phased-regular word equation. We prove this Theorem by structural induction on \( n \) where the Lemma 12 is used for the base case.

\[\square\]