Word Equations as Abstract Domain for String Manipulating Programs

Antonina Nepeivoda
Program Systems Institute of RAS
Russia
a_nevod@mail.ru

***Abstract*—The paper presents a conceptual approach to abstract interpretation of string-manipulating programs, based on the existential theory of strings.**

We propose the word equation language as a base for lattices forming abstract domains of the string data. We construct a quantifier-free layer of the lattices, capturing the uniqueness properties of join and meet operations. The resulting finite-height lattice WL_0 utilizes useful properties of primitive roots of words and can be used as a base for future developments of word-equation-based abstract domains.

We describe a tokenization procedure as a monotone lattice mapping, in order to enhance expressiveness of word equation language by means of string morphisms and special cases of other finite-state-machine transformations.

***Index Terms*—program analysis, abstract interpretation, word equations, lattice mappings**

I. Introduction

The problem of static analysis of string manipulating programs, especially in dynamically typed languages, is known to be hard. For instance, even the theory with the replace-all and concatenation functions is undecidable [1], as well as the theory with the concatenation and letter counting operations [2].

Moreover, most of linear orders on the set of strings depend on the alphabet numeration. This fact makes construction of partition of the set of strings to polyhedra or intervals non-trivial and problem-specific ones.

In order to make the problem tractable, appropriate over-approximations and restrictions are used in static analysis [3]–[5]. In abstract interpretation, if a decidable set of predicates is taken as an abstract domain, the main two problems arise:

- how to over-approximate the wide variety of string operations in the string domain by the operations in the abstract domain;
- how to over-approximate the infinite chains of the predicates in the abstract domain by finite chains, in order to make the static analysis terminating.

The more precise are mappings into the abstract domain, the longer chains can occur; on the other hand, too small lattice height guarantees very fast convergence of the analysis, but may have drastically low preciseness of the analysis, compared with the methods admitting finite chains of non-uniformly bounded length.

The research was partially supported by Huawei Technologies Co. Ltd., and by Russian Academy of Sciences, research project №122012700089-0.

Thus, the main two approaches to construct the abstract domains exist. The first one considers some decidable fragment of string theory, and defines the appropriate generalizations (widening operations [6]) in order to collapse the infinite chains [6]–[9]. This approach is language-independent, allowing high flexibility of the tracked program properties, by varying the widening operation. The second one takes concrete practical properties of interest as the abstract domain, and solves concrete verification tasks in terms of the chosen programming language [10], [11]. The lattices used in this analysis are of small fixed height making the analysis fast.

For example, in ECMAScript language [12], the set of numbers is defined over a wider alphabet than $\{[0-9], ., -\}$. The constants `infinity` and `NaN` are also considered as numerical data. Thus, if the property "can represent numerical data" is tracked, then the approach making use of a string theory is forced to make the widening operator more precise, risking to make the whole analysis potentially slower.

In order to combine these two approaches, one can use an abstract domain in a decidable string theory, together with taking a quotient [13] wrt a partition of the string data set, taking into account language-specific properties of its elements. The partition can be defined as a preprocessing tokenization procedure, thus changing the underlying alphabet in the same string theory. Hence, the string manipulating operations are to be interpreted both by tokenization algorithm and the computations over the abstract domain. Under certain conditions, the tokenized strings and predicates on the tokens can happen to be fixed points of the lattice on the input string data, thus forming a proper finite sub-lattice [14]. Thus, a decidable string theory may be chosen in such a way that the tokenization procedure becomes in some sense "orthogonal" to predicates of the theory. It is known that the existential theory of words (the theory of word equations) is decidable [15], and the set of word equation languages neither contains nor is contained in the set of regular languages [16]. The set of word equation languages is not closed under morphic images and inverse morphic images [16]. Hence, the tokenization is able to significantly improve expressiveness of the given theory.

In order to address the widening problem, we advocate to use a natural string property known as the primitive root factorization, which is expressible in the word equation language. A root of a word ω is a ξ s.t. $\xi^n = \omega$. A word ξ is primitive iff $\forall \tau, n\big(\xi = \tau^n \Rightarrow n = 1\big)$. Word equations may encapsulate a

https://doi.org/10.34727/2024/isbn.978-3-85448-065-5_14

wide variety of properties including some of statements about primitive roots. E.g. the equation $XY = YX$ represents the predicate "if the strings X and Y are non-empty, then they have the same primitive root".

Let us show an example of how the notion of the primitive root helps to solve the widening problem. The set of the regular expressions, which is known to be closed under both intersection and union, forms a distributive lattice [6], [8]. However, the regular expressions admit infinite ascending chains, e.g. $\mathcal{L}(a) \subseteq \mathcal{L}(a|a^2) \subseteq \mathcal{L}(a|a^2|a^3) \subseteq \dots$, where $\mathcal{L}(r)$ denotes the language recognised by expression r. The most obvious widening is to define the widened value as the Kleene iteration a^*. Under this definition, the values a^* and $(a^2)^*$ are still distinct, and the latter implies the former. Thus, we can define an infinite descending chain of the predicates, i.e. a^*, $(a^2)^*$, …, $(a^{2^n})^*$, …, which violates the lattice finiteness condition. Now let us define the widened value Iter_a as a predicate satisfied by all words X satisfying the word equation $aX = Xa$. The definition for Iter_a using the word equations makes it possible to collapse the predicates to a single layer of the lattice. Since $\forall \tau, n(n > 0 \Rightarrow \tau a^n = a^n \tau \Leftrightarrow \tau a = a\tau)$, any predicate Iter_{a^n} is equivalent to the predicate Iter_a, thus the chains collapse to single elements.

The contributions of the paper are as follows.

First, we suggest a word equation language as a base for a lattice forming an abstract domain of string data. We construct a first (quantifier-free) layer of the lattice, capturing the uniqueness properties of joins and meets (Section III). The resulting finite-height lattice $\mathbf{WL_0}$ utilizes useful properties of primitive roots and can be used as a base for future developments of word-equation-based abstract domains.

Second, we suggest a tokenization procedure as a monotone lattice mapping, in order to enhance expressiveness of the word equation language by means of string morphisms and special cases of other finite-state-machine transformations, based on inverse mappings of the string morphisms (Section V). Moreover, given a string morphism onto the string set of the initial abstract domain, the set of its fixpoints forms a complete sublattice of the lattice $\mathbf{WL_0}$, and no additional construction is required to track additional program properties captured by the morphism.

The paper is organized as follows. In Section II, the main notions of lattice theory and word equation theory are given. In Section III, the experimental lattice based on the word equations is presented, and Section IV presents abstract domain semantics of the standard string operations used in ECMAScript programs. Section V considers the tokenization transformations, Section VI discusses related works, and Section VII concludes the paper.

II. Preliminaries

Small Greek letters (maybe with indices) stand for finite constant words (strings); domains and sets are denoted with Greek capitals. Small Latin letters a, b, c, d are considered to be elements of Σ. Capital Latin letters X, Y, Z stand for elements of the variable alphabet. The notation τ^n stands for n-concatenation of τ with itself, i.e. $\underbrace{\tau\tau\dots\tau}_{n}$. The empty word is denoted by ε. Given a word τ, $|\tau|$ stands for its length.

A word τ is said to be *primitive* (denoted with $\mathrm{prm}(\tau)$), if $\forall \xi, n\big(\tau = \xi^n \Rightarrow n = 1\big)$. Thus ε is not a primitive word, since $\forall n(\varepsilon^n = \varepsilon)$. Every non-empty word τ has a unique primitive root ξ, i.e. $\forall n, m, \xi, \xi'\big(\xi^n = \tau = (\xi')^m \;\&\; |\tau| > 0 \;\&\; \mathrm{prm}(\xi) \;\&\; \mathrm{prm}(\xi') \Rightarrow \xi = \xi'\big)$. We denote the primitive root of τ with $\rho(\tau)$.

Definition II.1. *Given a letter alphabet Σ and a variable alphabet Ξ,* a word equation *is an equation $\mathcal{U} = \mathcal{V}$, where $\mathcal{U}, \mathcal{V} \in (\Sigma \cup \Xi)^*$.*

A solution *to an equation $\mathcal{U} = \mathcal{V}$ is a morphism σ which is identity on Σ and maps elements of Ξ into Σ^*, s.t. $\sigma(\mathcal{U}) = \sigma(\mathcal{V})$ [15], [16].*

We also call the set of possible tuples of variable images determined by solutions of $\mathcal{U} = \mathcal{V}$ the solution set of $\mathcal{U} = \mathcal{V}$. Given a variable set $\mathcal{Q}$, the solution set of $\mathcal{U} = \mathcal{V}$ wrt $\mathcal{Q}$ is the projection of the solution set of $\mathcal{U} = \mathcal{V}$ on the coordinates corresponding to the elements of $\mathcal{Q}$.

The following examples are classical [17], [18].

Example II.1. *Given an equation $aX = Xa$, where $X \in \Xi$ and $a \in \Sigma$, its solution set is $\{a^n \mid n \in \mathbb{N}\}$.*

Given an equation $ZX = XY$, where $X, Y, Z \in \Xi$, its solution set for (X, Y, Z) is $\{((\xi_1\xi_2)^n\xi_1, \xi_2\xi_1, \xi_1\xi_2) \mid n \in \mathbb{N} \;\&\; \xi_1, \xi_2 \in \Sigma^*\}$. The solution set of $ZX = XY$ wrt the variable X is $\{(\xi_1\xi_2)^n\xi_1 \mid \xi_1, \xi_2 \in \Sigma^*\}$. It implies that, given an equation $\tau_1 X = X\tau_2$, its solution set is non-empty iff $\exists \eta_1, \eta_2\big(\tau_1 = \eta_1\eta_2 \;\&\; \tau_2 = \eta_2\eta_1\big)$.

Given an equation $\xi_1\xi_2 X = X\xi_2\xi_1$, where $\xi_1, \xi_2 \in \Sigma^*$, $\mathrm{prm}(\xi_1\xi_2)$ and $|\xi_2| > 0$, and $X \in \Xi$, its solution set is $\{(\xi_1\xi_2)^n\xi_1 \mid n \in \mathbb{N}\}$. If $|\xi_1| = 0$, then the equation is reduced to $\xi_2 X = X\xi_2$, and its solution set is $\{\xi_2^n \mid n \in \mathbb{N}\}$.

Let us denote the predicate "τ satisfies the equation $\xi_1\xi_2\tau = \tau\xi_2\xi_1$" with $\mathrm{Cnj}_{\xi_1,\xi_2}(\tau)$. We assume that the representation of $\mathrm{Cnj}_{\xi_1,\xi_2}$ is reduced by default to the shortest possible value of $\xi_1\xi_2$, i.e. the word $\xi_1\xi_2$ is primitive in $\mathrm{Cnj}_{\xi_1,\xi_2}$. Moreover, since $\mathrm{Cnj}_{\tau,\varepsilon} = \mathrm{Cnj}_{\varepsilon,\tau}$, we always choose $\mathrm{Cnj}_{\varepsilon,\tau}$ as a default.

We recall the following classical Fine–Wilf theorem [18].

Theorem II.1. *Let $\xi_1, \xi_2 \in \Sigma^+$. Suppose ξ_1^m and ξ_2^n, for some $m, n \in \mathbb{N}$, have a common prefix of length $|\xi_1| + |\xi_2| - gcd(|\xi_1|, |\xi_2|)$. Then there exists $\tau \in \Sigma^*$ of length $gcd(|\xi_1|, |\xi_2|)$ such that $\tau = \rho(\xi_1) = \rho(\xi_2)$, i.e. τ is the primitive root both of ξ_1 and ξ_2.*

Definition II.2. *A triple $\langle \mathcal{L}, \vee, \wedge \rangle$, where $\mathcal{L}$ is a set, $\vee$ and $\wedge$ are binary operations over $\mathcal{L}$ (also called join and meet respectively), is said to be a lattice if it satisfies the following axioms [19] for all $x, y, z \in \mathcal{L}$:*

- $\big(x \vee (x \wedge y) = x\big) \;\&\; \big(x \wedge (x \vee y) = x\big)$;
- $\big(x \vee y = y \vee x\big) \;\&\; \big(x \wedge y = y \wedge x\big)$;
- $\big(x \vee (y \vee z) = (x \vee y) \vee z\big) \;\&\; \big(x \wedge (y \wedge z) = (x \wedge y) \wedge z\big)$.

An order induced on a lattice E with the lattice operations is defined as follows: $x \leq y \equiv (x \vee y = y)$.

Given lattices E, F, a mapping $\phi : E \to F$ is said to be consistent with the order (*isotonic*) iff $\forall x, y\big(x \leq y \Rightarrow \phi(x) \leq \phi(y)\big)$ [14]. A mapping ϕ is said to be *a lattice morphism* iff it respects both joins and meets [14].

The following lemma demonstrates a useful property of the equations $\xi_1\xi_2X = X\xi_2\xi_1$ (assuming by definition that $\mathrm{prm}(\xi_1\xi_2)$). Henceforth we call such equations elementary.

Lemma II.1. *If the words ξ_1, ξ_2, ξ_3, ξ_4 satisfy $\mathrm{prm}(\xi_1\xi_2)$ and $\mathrm{prm}(\xi_3\xi_4)$, and $\xi_1 \neq \xi_3$ or $\xi_2 \neq \xi_4$, then there exists at most one word $\tau \in \Sigma^*$ satisfying both $\mathrm{Cnj}_{\xi_1,\xi_2}(\tau)$ and $\mathrm{Cnj}_{\xi_3,\xi_4}(\tau)$.*

Proof. Let $\tau = (\xi_1\xi_2)^n\xi_1 = (\xi_3\xi_4)^m\xi_3$. Without loss of generality, we assume that $|\xi_1\xi_2| \geq |\xi_3\xi_4|$; the opposite case is symmetric.

If $n > 1$ (and hence $m > 1$), then the word $\tau\xi_2 = (\xi_1\xi_2)^{n+1}$ and the word $\tau\xi_4 = (\xi_3\xi_4)^{m+1}$ share a common prefix of the length $|\tau|$, which is at least $|\xi_1|+|\xi_2|+|\xi_3|+|\xi_4|$. Hence, by the Fine–Wilf theorem [20], $\xi_1\xi_2$ and $\xi_3\xi_4$ share a common primitive root, i.e. are equal, because they are primitive. Hence, $n = m$ and $\xi_1 = \xi_3$, which contradicts the choice of ξ_i.

Thus, if there are such distinct $\tau_0, \tau_1 \in \Sigma^*$, both belonging to the solution sets of $\xi_1\xi_2X = X\xi_2\xi_1$ and $\xi_3\xi_4X = X\xi_4\xi_3$, then $\tau_i = (\xi_1\xi_2)^i\xi_1$. I.e. $\exists k_1 \geq 0, k_2 > 0$ s.t. $\tau_0 = \xi_1 = (\xi_3\xi_4)^{k_1}\xi_3$, $\tau_1 = \xi_1\xi_2\xi_1 = (\xi_3\xi_4)^{k_1+k_2}\xi_3 = (\xi_3\xi_4)^{k_2}(\xi_3\xi_4)^{k_1}\xi_3 = (\xi_3\xi_4)^{k_2}\xi_1$. That implies $\xi_1\xi_2 = (\xi_3\xi_4)^{k_2}$, hence, $k_2 = 1$ and $\xi_1 = \xi_3$, since $\xi_1\xi_2$ is primitive, which contradicts the choice of ξ_i.

□

The proof above immediately implies the following Corollary. If Lemma's II.1 premise is true, then the only one of the following three cases can hold.

- No word satisfies the predicate $\mathrm{Cnj}_{\xi_1,\xi_2}$ & $\mathrm{Cnj}_{\xi_3,\xi_4}$.
- $\exists k\big(\xi_1 = (\xi_3\xi_4)^k\xi_3\big)$.
- $\exists k\big(\xi_1\xi_2\xi_1 = (\xi_3\xi_4)^k\xi_3\big)$.

We denote the word satisfying both $\mathrm{Cnj}_{\xi_1,\xi_2}$ and $\mathrm{Cnj}_{\xi_3,\xi_4}$ by $\mathrm{conjr}(\xi_1, \xi_2, \xi_3, \xi_4)$. Lemma II.1 shows that the predicates $\mathrm{Cnj}_{\xi_1,\xi_2}$ and $\mathrm{Cnj}_{\xi_3,\xi_4}$ are "orthogonal" wrt the sets of words satisfying them. For example, if $\xi_2 \neq \xi_4$ then $\mathrm{conjr}(\varepsilon, \xi_2, \varepsilon, \xi_4) = \varepsilon$.

III. The Lattice Construction

Let us introduce a relation $\propto$ between elements of the concrete string domain $\mathcal{S}^{\#}$ and an abstract domain Δ. A word τ satisfies a predicate P, where $\tau \in \mathcal{S}^{\#}$ and $P \in \Delta$, iff $\tau \propto P$.

The antimonotonous Galois connection defined by the relation $\propto$ determines the abstraction and concretisation operations wrt the abstract domain Δ.

As usual, the values $\top$ and $\bot$ represent the greatest and the least element of the lattice. The first level higher than $\bot$ (i.e. layer 1) captures the trivial word equations $X = \xi$, denoted by Eq_ξ, which is standard for the string abstract domains [9], [10]. As for the next lattice levels (layers whose numbers start with 2), we require them to satisfy the following properties.

- For any element P of a layer higher than the layer 1 (i.e., the layer of the trivial word equations), there is an infinite string set for which P holds (expressiveness).
- Given any two distinct elements P_1 and P_2 of the layer N, there is at most one predicate P of the layer $N+1$ s.t. $P_1 \Rightarrow P$ and $P_2 \Rightarrow P$ hold (unique join).
- Given any two distinct predicates P_1 and P_2 of the layer $N+1$, there is at most one predicate P of the layer N s.t. $P \Rightarrow P_1$ and $P \Rightarrow P_2$ hold (unique meet).

The first property guarantees that all the elements of the layer are expressible enough; the second property is required to define unique join elements, and the third is used to define unique meet elements. Obviously, the top element $\top$ satisfies all the three conditions.

A natural question arises: how can one introduce a partial word equation order that is able to distinguish the equations belonging to various levels? Given equations $\mathcal{U}_1 = \mathcal{V}_1$ and $\mathcal{U}_2 = \mathcal{V}_2$ in alphabet Σ s.t. $|\Sigma| \geq 1$, an equation whose solution set wrt the variables occurring in $\mathcal{U}_1$, $\mathcal{U}_2$, $\mathcal{V}_1$, $\mathcal{V}_2$ is a union of the solution sets of the given equations can be constructed with introducing 2 additional (fresh) variables (see [16], in the earlier work [21] a construction with 4 additional variables is given). On the other hand, an equation with the solution set representing the intersection of the two solution sets above can be constructed without any additional variable, provided that $|\Sigma| > 1$. Hence, the number of distinct variables in a given equation can be treated as a measure for its "generality", provided that the solution set of the equation is considered wrt a single variable X. With respect to this measure, the simplest equations depend only on X itself, i.e. are of the form $P : \xi_1X\xi_2X\ldots\xi_nX = X\xi'_1X...X\xi'_m$, where $|\xi_1| > 0$. If $m \neq n$, then P has finitely many solutions; thus, the expressiveness requirement is satisfied[1] only if $m = n$. The lemma below shows that any equation P with infinitely many solutions is equivalent to an equation of the form $\mathrm{Cnj}_{\xi_1,\xi_2}$.

Lemma III.1. *The set of predicates of the form $\mathrm{Cnj}_{\xi_1,\xi_2}$, where the word $\xi_1\xi_2$ is primitive, satisfies all the three conditions given above. Any other quantifier-free predicate satisfying the conditions is equivalent to a predicate $\mathrm{Cnj}_{\xi_1,\xi_2}$ in the given set.*

Proof. Given an equation P of the form $\xi_1X\xi_2X\ldots\xi_nX = X\xi'_1X...X\xi'_n$, let us assume that there exists its solution ω s.t. $|\omega| \geq |\xi_1|$. Then $\omega = \xi_1\omega'$, and ω' is a solution of the equation which arises from P in virtue of the substitution $X \mapsto \xi_1X'$. I.e., removing the ξ_1-prefixes on both sides of $P[X \mapsto \xi_1X']$, we obtain the equation $P' : \xi_1X'\xi_2\xi_1X'\ldots\xi_n\xi_1X' = X'\xi'_1\xi_1X'\ldots X'\xi'_n$ that is to be satisfied by ω'. As well as the initial equation P, equation P' has prefixes ξ_1X' and X' in its left- and right-hand sides, hence, the reasoning above can be repeated until $|\omega'| < |\xi_1|$.

[1] As shown in the paper [22], such equations have either at most 3 solutions or infinitely many solutions.

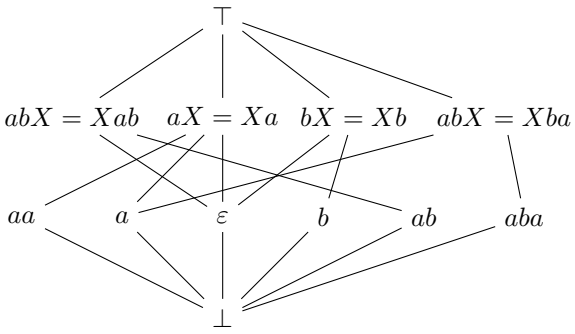


Fig. 1: Lattice built over constants ε, a, a^2, ab, aba. The values Eq_ξ are represented as ξ; the values $\mathrm{Cnj}_{\xi_1,\xi_2}$ are represented as the equations $\xi_1\xi_2X = X\xi_2\xi_1$.

Therefore, any solution to the equation P, where $|\xi_1| > 0$, is of the form $(\xi_{1,p}\xi_{1,s})^k\xi_{1,p}$, where $\xi_{1,p}\xi_{1,s} = \xi_1$.

Let us take such a number k_0 that $\max(\max_{1\le i\le n}|\xi_i|, \max_{1\le i\le n}|\xi_i'|)\cdot n < |\xi_1|\cdot k_0$, and separate the solution set of P into the following two sets.

- The words of the length less than $|\xi_1|\cdot k_0$.
- All the other words from the solution set of P. These words start with the prefix $\xi_1^{k_0}$; they can be seen as the solutions to equation $\sigma(P)$, where $\sigma : X \mapsto \xi_1^{k_0}X$.

Due to the choice of k_0, the equation $\xi_1X\xi_2\xi_1^{k_0}X\ldots\xi_n\xi_1^{k_0}X = X\xi_1'\xi_1^{k_0}X\ldots\xi_1^{k_0}X\xi_n'$ resulting from the mapping $X \mapsto \xi_1^{k_0}X$ can be split into n equations of the form $\tau_{i,1}X = X\tau_{i,2}$ (possibly, after reducing common prefixes and suffixes of the equation parts). Some of these equations are equivalent (if for some i, j and $k \in \{1, 2\}$ the primitive roots of $\tau_{i,k}$ and $\tau_{j,k}$ coincide), so we take only the subset of non-equivalent equations.

If this subset is a singleton, then the resulting equation is equal to the first equation $\xi_1X = X\tau$, where $|\tau| = |\xi_1|$, and τ may be either a prefix of ξ_1' (if $|\xi_1| < |\xi_1'|$) or of the form $\xi_1'\tau'$. In both cases, the solution set of this equation also includes any solution to P of the length less than $|\xi_1|\cdot k_0$.

If the set of the non-equivalent equations is not a singleton, then by Lemma II.1 the equation P has finitely many solutions and does not satisfy the expressiveness condition. Lemma II.1 guarantees that the unique meet condition holds.

Let us show that the unique join condition also holds. Given two distinct $\tau_1, \tau_2 \in \Sigma^*$ satisfying some elementary equation $\xi_1\xi_2X = X\xi_2\xi_1$ with ξ_1 and ξ_2 unknown, let $|\tau_1| > |\tau_2|$. Then $\exists\tau_3(\tau_3 \ne \varepsilon \;\&\; \tau_1 = \tau_3\tau_2)$, and the primitive root of τ_3 is equal to $\xi_1\xi_2$, while the suffix of τ_2 after the maximal prefix of the form $\rho(\tau_3)^k$ coincides with ξ_1. Hence, the values ξ_1 and ξ_2 in equation $\xi_1\xi_2X = X\xi_2\xi_1$ are determined by any two distinct words τ_1 and τ_2 satisfying this equation. □

Lemma III.1 determines elements of the third level of lattice $\mathbf{WL_0}$, namely the set of predicates $\mathrm{Cnj}_{\xi_1,\xi_2}$ defining infinite solution sets of one-variable equations. A simple word-equation-based lattice can consist of the three given layers, and the top layer above them. Other possible extensions of the lattice are discussed in Section VII.

Based on the reasoning above, now we formally introduce the lattice elements and operations. The abstract domain Δ of the simplest lattice $\mathbf{WL_0}$ proposed in this paper consists of the following elements. As usual, we always assume that given a predicate $\mathrm{Cnj}_{\xi_1,\xi_2}$, the word $\xi_1\xi_2$ is primitive.

- Predicates Eq_ξ. $\mathrm{Eq}_\xi(\tau)$ iff $\tau = \xi$.
- Predicates "conjugates $\xi_1\xi_2$ and $\xi_2\xi_1$", denoted with $\mathrm{Cnj}_{\xi_1,\xi_2}$, where $|\xi_1\xi_2| > 0$. $\mathrm{Cnj}_{\xi_1,\xi_2}(\tau)$ iff $\xi_1\xi_2\tau = \tau\xi_2\xi_1$.
- The top element $\top$ representing all possible strings, and the bottom element $\bot$.

A simple example of such a lattice constructed over constants ε, a, a^2, ab, aba is presented in Fig. 1.

A. Operations of Lattice $\mathbf{WL_0}$

Let us define the join operation over the given domain. The right-hand sides of the definitions below are ordered to be applied from top to bottom.

- $\mathrm{Cnj}_{\tau_1,\tau_2} \vee \mathrm{Cnj}_{\xi_1,\xi_2} = \begin{cases}\mathrm{Cnj}_{\tau_1,\tau_2}, \text{ if } \forall i(\xi_i = \tau_i);\\ \top, \text{ otherwise;}\end{cases}$
- $\mathrm{Cnj}_{\tau_1,\tau_2} \vee \mathrm{Eq}_\xi = \begin{cases}\mathrm{Cnj}_{\tau_1,\tau_2}, \text{ if } \xi\tau_2\tau_1 = \tau_1\tau_2\xi;\\ \top, \text{ otherwise;}\end{cases}$
- $\mathrm{Eq}_\xi \vee \mathrm{Cnj}_{\tau_1,\tau_2} = \mathrm{Cnj}_{\tau_1,\tau_2} \vee \mathrm{Eq}_\xi$;
- $\mathrm{Eq}_{\tau_1} \vee \mathrm{Eq}_{\tau_2} = \begin{cases}\mathrm{Eq}_{\tau_1}, \text{ if } \tau_1 = \tau_2;\\ \mathrm{Cnj}_{\xi_1,\xi_2}, \text{ if } \exists\xi_1,\xi_2,k_1,k_2(k_1,k_2 \in \mathbb{N}\\ \;\&\; \tau_1 = (\xi_1\xi_2)^{k_1}\xi_1 \;\&\; \tau_2 = (\xi_1\xi_2)^{k_2}\xi_1);\\ \top, \text{ otherwise.}\end{cases}$.

The case returning $\mathrm{Cnj}_{\xi_1,\xi_2}$ as a value of $\mathrm{Eq}_{\tau_1} \vee \mathrm{Eq}_{\tau_2}$ reproduces the construction given in the proof of Lemma III.1, when the unique join property is checked.

E.g., $\mathrm{Eq}_\varepsilon \vee \mathrm{Eq}_a = \mathrm{Cnj}_{\varepsilon,a}$, as well as $\mathrm{Eq}_\varepsilon \vee \mathrm{Eq}_{aa} = \mathrm{Cnj}_{\varepsilon,a}$. $\mathrm{Eq}_{aba} \vee \mathrm{Cnj}_{a,b} = \mathrm{Cnj}_{a,b}$, since $(aba)ba = ab(aba)$.

The commutativity axiom for the join operation holds by definition.

If some of elements x, y, z of $\mathbf{WL_0}$ are equal, or any two of them are distinct equations, then the associativity $x \vee (y \vee z) = (x \vee y) \vee z$ also holds by definition. Let us consider the subtle case of the associativity: $x = \mathrm{Eq}_{\tau_1}$, $y = \mathrm{Eq}_{\tau_2}$, $z = \mathrm{Eq}_{\tau_3}$, $x \vee y = \mathrm{Cnj}_{\xi_1,\xi_2}$, $y \vee z = \mathrm{Cnj}_{\xi_3,\xi_4}$, $\xi_1 \ne \xi_3$ or

$\xi_2 \neq \xi_4$. Then by Lemma II.1 τ_2 is the only word satisfying the predicates $\mathrm{Cnj}_{\xi_1,\xi_2}$ and $\mathrm{Cnj}_{\xi_3,\xi_4}$ (i.e. $y \implies \mathrm{Cnj}_{\xi_1,\xi_2}$ & $y \implies \mathrm{Cnj}_{\xi_3,\xi_4}$), thus, $x \vee \mathrm{Cnj}_{\xi_3,\xi_4} = \top$ and $\mathrm{Cnj}_{\xi_1,\xi_2} \vee z = \top$ hold.

Now we define the meet operation.

- $\mathrm{Cnj}_{\tau_1,\tau_2} \wedge \mathrm{Cnj}_{\xi_1,\xi_2} = \begin{cases} \mathrm{Cnj}_{\tau_1,\tau_2}, \text{ if } \forall i(\xi_i = \tau_i); \\ \mathrm{Eq}_{\mathrm{conjr}(\tau_1,\tau_2,\xi_1,\xi_2)}, \\ \quad \text{if } \mathrm{conjr}(\tau_1,\tau_2,\xi_1,\xi_2) \text{ exists}; \\ \bot, \text{ otherwise.} \end{cases}$
- $\mathrm{Cnj}_{\tau_1,\tau_2} \wedge \mathrm{Eq}_\xi = \begin{cases} \mathrm{Eq}_\xi, \text{ if } \tau_1\tau_2\xi = \xi\tau_2\tau_1; \\ \bot, \text{ otherwise;} \end{cases}$
- $\mathrm{Eq}_\tau \wedge \mathrm{Cnj}_{\xi_1,\xi_2} = \mathrm{Cnj}_{\xi_1,\xi_2} \wedge \mathrm{Eq}_\tau$;
- $\mathrm{Eq}_{\tau_1} \wedge \mathrm{Eq}_{\tau_2} = \begin{cases} \mathrm{Eq}_{\tau_1}, \text{ if } \tau_1 = \tau_2; \\ \bot, \text{ otherwise.} \end{cases}$

There in the first case we refer to the property of elementary equations guaranteed by Lemma II.1. E.g., $\mathrm{Cnj}_{a,b} \wedge \mathrm{Cnj}_{\varepsilon,a} = \mathrm{Eq}_a$, since a satisfies both equations $abX = Xba$ and $aX = Xa$, hence, $a = \mathrm{conjr}(a, b, \varepsilon, a)$ (see Fig. 1).

By a similar reasoning, the $\wedge$ operation is associative.

Now we consider the last lattice condition to be checked.

- Since $\forall x, y\big(x \wedge y \Rightarrow x\big)$, and $\forall q\big((q \Rightarrow x) \Rightarrow (x \vee q = x)\big)$, the law $x \vee (x \wedge y) = x$ also holds.
- $x \wedge (x \vee y)$ is x iff $x \vee y \Rightarrow x$. The condition $x \vee y \Rightarrow x$ is guaranteed by the construction of the operations.

Hence the lattice definition is consistent. This lattice is not distributive. E.g. $\mathrm{Cnj}_{\varepsilon,a} \wedge (\mathrm{Cnj}_{\varepsilon,b} \vee \mathrm{Cnj}_{\varepsilon,c}) = \mathrm{Cnj}_{\varepsilon,a}$, but $(\mathrm{Cnj}_{\varepsilon,a} \wedge \mathrm{Cnj}_{\varepsilon,b}) \vee (\mathrm{Cnj}_{\varepsilon,a} \wedge \mathrm{Cnj}_{\varepsilon,c}) = \mathrm{Eq}_\varepsilon$.

IV. Operations on Lattice Elements

A. A Model Program

In order to demonstrate the computations in the abstract domain given above, let us consider the following example, given in a pseudocode (Fig. 2).

The $x + y$ concatenates x and y; $x - y$ deletes a prefix y from x; $prefix(x, y)$ checks whether a string x is a prefix of a string y.

```
1   z = ξ
2   x,y = ε
3   while (cond1(i,j)) {        (depends only on i, j)
4     i = i + 1
5     x = x + z }
6   while (cond2(i,j)) {        (depends only on i, j)
7     j = j + 1
8     y = y + z }
9   while (true) {
10    if (prefix(x,y))
11      y = y − x
12    elif (prefix(y,x))
13      x = x − y
14    else break }
```

Fig. 2: A fragment of a string-manipulating program incorrectly checking that a quotient of strings x and y is ε.

The program lines 9–14 aim at computing a "quotient" of the two strings, i.e. the word witnessing that the strings have different primitive roots. For example, if $x = abba$, $y = abbaab$, then the loop 9–14 breaks at the state $x = ba$, $y = ab$ after the two iterations. If the roots coincide, then the loop 9–14 is assumed to return ε, however if x is assigned to ε, the loop does not terminate. The reason of the non-termination is that ε is a prefix of any string, hence $\tau - \varepsilon = \tau$ for any $\tau \in \Sigma^*$. Moreover, the program given in Fig. 2 never terminates, because after executing lines 1–8 the values of x and y always have equal primitive roots.

Let us see how the corresponding operations are computed over the lattice $\mathbf{WL_0}$, and how the problem with the infinite loop can be revealed.

B. Computations in $\mathbf{WL_0}$

The following operations are chosen in order to demonstrate computations in $\mathbf{WL_0}$. The operations are analogous to operations included in standard string operating libraries, e.g. for ECMAScript [12]. Such a library includes at least concatenation operation, denoted with $x + y$; string replacement and truncation operations. In JavaScript, there exist the function replacing the first occurrence of a given string ξ in a string τ, and the function replacing all occurrences of ξ in τ. We denote the operation replacing the first occurrence of z_1 in y with z_2 with $replace(y, z_1, z_2)$. The string truncation usually depends on a given input — start and end positions of a substring that is to be deleted or extracted as an infix. We consider the following instance of the truncation: the string minus operation of the form $x - y$, where the prefix y is deleted from x.

We consider the versions of the operations with the numerical parameters unknown to the interpreter; if these parameters are known, the more precise over-approximations can be constructed. We assume that the right-hand sides of the interpretation rules in the interpretations given below are ordered from top to bottom to be applied. Some of the interpretations are straightforward; we comment only on the non-obvious ones. The order $\leq$ is induced by the join operation (see Section II). As usual, $\xi_1\xi_2$ in $\mathrm{Cnj}_{\xi_1,\xi_2}$ is assumed to be primitive.

Below the abstract version of the string concatenation is given.

$$x + y = \begin{cases} \mathrm{Eq}_{\xi_1\xi_2}, \text{ if } x = \mathrm{Eq}_{\xi_1}, y = \mathrm{Eq}_{\xi_2}; \\ \mathrm{Cnj}_{\xi_5,\xi_6}, \text{ if } x \leq \mathrm{Cnj}_{\xi_1,\xi_2} \;\&\; y \leq \mathrm{Cnj}_{\xi_3,\xi_4} \\ \qquad \text{and } \xi_2\xi_1 = \xi_3\xi_4 \text{ and } \xi_5\xi_6 = \xi_1\xi_2 \\ \qquad\quad \text{and } \exists n\big(\xi_1\xi_3 = (\xi_1\xi_2)^n\xi_5\big); \\ \top, \text{ otherwise.} \end{cases}$$

Given words τ_1 and τ_2 in the concrete string domain, if $\tau_1 + \tau_2 = (\xi_5\xi_6)^k\xi_5$, where k is large enough, then either $\tau_1 = (\xi_5\xi_6)^{k_1}\tau_5$ and $\tau_2 = \tau_6(\xi_5\xi_6)^{k_2}\xi_5$, where $\tau_5\tau_6 = \xi_5\xi_6$, or $\tau_1 = (\xi_5\xi_6)^k\tau_5$ and $\tau_2 = \tau_6$, where $\tau_5\tau_6 = \xi_5$. The case returning $\mathrm{Cnj}_{\xi_5,\xi_6}$ above includes both these instances. The parameter n above equals 0 if $|\xi_3| < |\xi_2|$, and equals 1 otherwise.

Below the abstract version of the string subtraction is given.

$$x-y=\begin{cases}\mathrm{Eq}_\tau,\ \text{if } x=\mathrm{Eq}_{\xi\tau}\ \&\ y=\mathrm{Eq}_\xi;\\ \text{error, if } x=\mathrm{Eq}_\tau\ \&\ y=\mathrm{Eq}_\xi\ \&\ \forall\tau'(\tau\neq\xi\tau');\\ \text{error, if } x=\mathrm{Cnj}_{\tau_1,\tau_2}\ \&\ y=\mathrm{Eq}_\xi\\ \qquad\&\ \neg(y\leq\mathrm{Cnj}_{\xi_1,\xi_2}),\ \text{where } \tau_1\tau_2=\xi_1\xi_2;\\ x,\ \text{if } y=\mathrm{Cnj}_{\varepsilon,\xi}\ \text{and}\\ \quad\text{either}\quad x=\mathrm{Eq}_\tau\quad\text{and}\quad\forall\tau'\big(\tau\neq\xi\tau'\big),\\ \quad\text{or } x=\mathrm{Cnj}_{\tau_1,\tau_2}\ \text{and}\ \forall\tau',k\big((\tau_1\tau_2)^k\tau_1\neq\xi\tau'\big);\\ \top,\ \text{if } y=\mathrm{Cnj}_{\xi_1,\xi_2}\ \text{and } x-y\ \text{can satisfy at least}\\ \quad\text{two different predicates of the form}\,\mathrm{Cnj}_{\omega_1,\omega_2};\\ \mathrm{Cnj}_{\tau_{1,2},\tau_2\tau_{1,1}},\ \text{if } x\leq\mathrm{Cnj}_{\tau_1,\tau_2}\ \text{and } y\leq\mathrm{Cnj}_{\xi_1,\xi_2}\\ \qquad\text{and } \exists k,k',\tau_{1,1},\tau_{1,2}\\ \quad\big((\tau_1\tau_2)^k\tau_{1,1}=(\xi_1\xi_2)^{k'}\xi_1\ \&\ \tau_1=\tau_{1,1}\tau_{1,2}\big);\\ \mathrm{Cnj}_{\tau_{2,2}\tau_1,\tau_{2,1}},\ \text{if } x\leq\mathrm{Cnj}_{\tau_1,\tau_2}\ \text{and } y\leq\mathrm{Cnj}_{\xi_1,\xi_2}\\ \qquad\text{and } \exists k,k',\tau_{2,1},\tau_{2,2}\\ \quad\big((\tau_1\tau_2)^k\tau_{1}\tau_{2,1}=(\xi_1\xi_2)^{k'}\xi_1\ \&\ \tau_2=\tau_{2,1}\tau_{2,2}\big);\\ \top,\ \text{otherwise.}\end{cases}$$

The x case above corresponds to the case when the prefix subtraction from x should succeed on the only possible concrete string value satisfying the predicate y. The intermediate case returning $\top$ corresponds to the case when a predicate of the form $\mathrm{Cnj}_{\tau_1,\tau_2}$ capturing concrete values of $x-y$ is undetermined, given arbitrary values satisfying the predicates – abstract values x and y. The remaining cases (besides the trivial one) consider the computations when such a predicate is unique. The detailed comments on the case with the undetermined value of $x-y$ and the cases returning conjugation predicates are given in Appendix (see Subsect. VII).

Now we consider the abstract version of the string replacement operation.

$$replace(y,z_1,z_2)=\begin{cases}\text{error, if } z_1=\mathrm{Eq}_\varepsilon;\\ \mathrm{Eq}_{replace(\tau,\xi_1,\xi_2)},\\ \ \text{if } y=\mathrm{Eq}_\tau, z_1=\mathrm{Eq}_{\xi_1}, z_2=\mathrm{Eq}_{\xi_2};\\ \mathrm{Cnj}_{\tau_1,\tau_2},\ \text{if } y\leq\mathrm{Cnj}_{\tau_1,\tau_2}\\ \quad\text{and } z_1,z_2\leq\mathrm{Cnj}_{\varepsilon,\xi_1\xi_2}\ \text{s.t. } \xi_2\xi_1=\tau_1\tau_2;\\ \mathrm{Cnj}_{\tau_1,\tau_2},\ \text{if } y=\mathrm{Cnj}_{\tau_1,\tau_2}\\ \quad\text{and } z_1\leq\mathrm{Cnj}_{\xi_1,\xi_2}\ \text{and}\ \forall n,k,\tau_3,\tau_4\\ \qquad\Big(\big(\xi_1\neq\varepsilon\Rightarrow(\tau_1\tau_2)^n\neq\tau_3\xi_1\tau_4\big)\\ \&\ \big(k>0\Rightarrow(\tau_1\tau_2)^n\neq\tau_3(\xi_1\xi_2)^k\xi_1\tau_4\big)\Big);\\ \top,\ \text{otherwise.}\end{cases}$$

There the satisfiability of y to the predicate $\mathrm{Cnj}_{\tau_1,\tau_2}$ is preserved in the following two cases. First, the result of the replacement satisfies $\mathrm{Cnj}_{\tau_1,\tau_2}$ if a power of a primitive word $\xi_1\xi_2$ conjugating with $\tau_1\tau_2$ (i.e. s.t. $\xi_2\xi_1=\tau_1\tau_2$) is replaced with $(\xi_1\xi_2)^k$. Second, the value of y is unchanged if no occurrence of a string satisfying z_1 can appear in a string satisfying the predicate given by y.

C. Predicates

The predicates defined on the string domain, under certain conditions, may be equivalent to the predicates defined on some other domain, e.g. integers. For example, if $\exists z\big(y=xz\big)$, then we may deduce that $|x|\leq|y|$. If additionally x and y are known to belong to the language a^*, then the predicate $|x|\leq|y|$ becomes equivalent to $\exists z\big(y=xz\big)$. Thus, if the latter is replaced by the former, sometimes a dead code can be eliminated by a simple static analyser. On the other hand, if $\exists z\big(y=xz\big)$ and y is known to be ε, then the predicate $\exists z\big(y=xz\big)$ can be replaced by the equivalent condition $\big(y=\varepsilon\big)\ \&\ \big(x=\varepsilon\big)$, which can also simplify tracking some of unreachable computation branches.

In fact, the predicate processing searches for invariants of the conditionals or loops, that can be derived from the values of the variables involved in the predicates over the abstract domain. This technique is close to one used in the paper [23], in order to prune unreachable computation branches in string manipulating programs.

The following simple interpretation of the predicate $prefix(x,y)\ \Leftrightarrow\ \exists z(y=xz)$ helps a static analysis tool to detect the non-terminating loop shown in the program in Fig. 2. We denote a value of the concretisation function on the abstract value x with $a(x)$. There the line $prefix(x,y)=f\big(a(x),a(y)\big)$ is interpreted as "if intersection of the x-concretisation set and $Pref(y)$ is non-empty, where $Pref(y)$ is the set of prefixes of all elements of the y-concretisation set, then the predicate $prefix(x,y)$ can be replaced with $f\big(a(x),a(y)\big)$". The capitalized OR notation stands for the logical operation in the target program language.

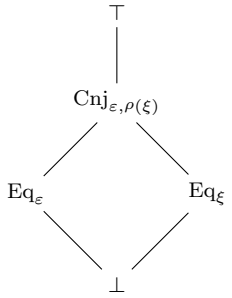
$$prefix(x,y)=\begin{cases}\big(a(x)=\xi_1\ \text{OR}\ \ldots\ \text{OR}\ a(x)=(\xi_1\xi_2)^n\xi_1\big),\\ \qquad\text{if } x=\mathrm{Cnj}_{\xi_1,\xi_2}\\ \quad\text{and}\ \exists\xi_3,n\big(n\in\mathbb{N}\ \&\ y=\mathrm{Eq}_{(\xi_1\xi_2)^n\xi_1\xi_3}\big);\\ |a(x)|\leq|a(y)|,\ \text{if } x=\mathrm{Cnj}_{\xi_1,\xi_2}\\ \qquad\text{and } y=\mathrm{Cnj}_{\xi_3,\xi_4}\ \text{and}\ \xi_1\xi_2=\xi_3\xi_4;\\ a(x)=\varepsilon,\ \text{if } y=\mathrm{Eq}_\varepsilon;\\ true,\ \text{if}\ \exists\xi\big(y=\mathrm{Eq}_{\xi_1\xi}\ \&\ x=\mathrm{Eq}_{\xi_1}\big);\\ false,\ \text{if } x=\mathrm{Eq}_{\xi_1}, y=\mathrm{Eq}_{\xi_2},\ \text{otherwise;}\\ \big(a(x)=\xi_1\ \text{OR}\ \ldots\ \text{OR}\ a(x)=(\xi_1\xi_2)^n\xi_1\big),\\ \qquad\text{if } x=\mathrm{Cnj}_{\xi_1,\xi_2}, y=\mathrm{Cnj}_{\xi_3,\xi_4},\exists\tau,m,n\\ \quad\big(m,n\in\mathbb{N}\ \&\ (\xi_3\xi_4)^m\xi_3=(\xi_1\xi_2)^n\xi_1\tau\big);\\ prefix\big(a(x),a(y)\big),\ \text{otherwise.}\end{cases}$$

The 1-st and the 6-th cases of the definition above contain a disjunction of n possible equalities for $a(x)$, which can be derived from the corresponding abstract values of x and y. The value of n is also determined by these abstract values. In the 6-th case n is bounded because $\xi_1\xi_2\neq\xi_3\xi_4$ holds, since the case $\xi_1\xi_2=\xi_3\xi_4$ is completely handled by the previous cases. In the 1-st case n is trivially bounded. Hence, the n-disjunction can be constructed without a loop.

A trace of the abstract interpretation using the interpretations given above is presented in Fig. 3. The notation $x\mapsto w$ states that the abstract value of x is w; $x\mapsto^*w$ states that the abstract value of x converges to w. In lines 5, 8, 11, 13, the fixed points of the computations are constructed. The join of Eq_ε and Eq_ξ, which is a value both of x and y, is $\mathrm{Cnj}_{\varepsilon,\rho(\xi)}$, and then $\mathrm{Cnj}_{\varepsilon,\rho(\xi)}$ is concatenated with Eq_ξ using the 2-nd rule for concatenation (Subsect. IV-B). The results of string subtraction stabilize in the same way. When the abstract interpretation converges, the predicates can be replaced in the concrete domain. After the replacement, a simple static

1	`z = ξ`		$z \mapsto \mathrm{Eq}_{\xi}$
2	`x,y = ε`		$x \mapsto \mathrm{Eq}_{\varepsilon}$, $y \mapsto \mathrm{Eq}_{\varepsilon}$
3	**while** (*cond*$_1$ `(i, j)`) {		(*cond*$_1$ `(i, j)` is outside the string domain)
4	`i = i + 1`		(ignored)
5	`x = x + z` }	$\mathrm{Eq}_{\varepsilon} \vee \mathrm{Eq}_{\xi} = \mathrm{Cnj}_{\varepsilon,\rho(\xi)}$; $\mathrm{Cnj}_{\varepsilon,\rho(\xi)} \vee \mathrm{Cnj}_{\varepsilon,\rho(\xi)} = \mathrm{Cnj}_{\varepsilon,\rho(\xi)}$, $x \mapsto^{*} \mathrm{Cnj}_{\varepsilon,\rho(\xi)}$	
6	**while** (*cond*$_2$ `(i, j)`) {		(*cond*$_2$ `(i, j)` is outside the string domain)
7	`j = j + 1`		(ignored)
8	`y = y + z` }	$\mathrm{Eq}_{\varepsilon} \vee \mathrm{Eq}_{\xi} = \mathrm{Cnj}_{\varepsilon,\rho(\xi)}$; $\mathrm{Cnj}_{\varepsilon,\rho(\xi)} \vee \mathrm{Cnj}_{\varepsilon,\rho(\xi)} = \mathrm{Cnj}_{\varepsilon,\rho(\xi)}$, $y \mapsto^{*} \mathrm{Cnj}_{\varepsilon,\rho(\xi)}$	
9	**while** (**true**) {		
10	**if** (*prefix* `(x,y)`)		(equivalent to $\lvert x\rvert \le \lvert y\rvert$ after the interpretation)
11	`y = y − x`		$\mathrm{Cnj}_{\varepsilon,\rho(\xi)} \vee \mathrm{Cnj}_{\varepsilon,\rho(\xi)} = \mathrm{Cnj}_{\varepsilon,\rho(\xi)}$; $y \mapsto^{*} \mathrm{Cnj}_{\varepsilon,\rho(\xi)}$
12	**elif** (*prefix* `(y,x)`)		(equivalent to $\lvert y\rvert \le \lvert x\rvert$ after the interpretation)
13	`x = x − y`		$\mathrm{Cnj}_{\varepsilon,\rho(\xi)} \vee \mathrm{Cnj}_{\varepsilon,\rho(\xi)} = \mathrm{Cnj}_{\varepsilon,\rho(\xi)}$; $x \mapsto^{*} \mathrm{Cnj}_{\varepsilon,\rho(\xi)}$
14	**else break** }		(unreachable after the interpretation)

(a) Tracking the abstract values of the program variables.



(b) The lattice on the abstract values used in the interpretation. Eq_{ξ} corresponds to the equation $X = \xi$; $\mathrm{Cnj}_{\xi_1,\xi_2}$ corresponds to the equation $\xi_1\xi_2 X = X\xi_2\xi_1$.

Fig. 3: Static analysis of the program that incorrectly checks that a quotient of strings x and y is ε.

analysis tool can determine that the line 14 is unreachable because the disjunction $|x| \le |y|$ OR $|y| \le |x|$ always holds, and the loop given in the lines 9–14 never terminates.

V. Tokenization

In order to construct a sound mapping from the string set into a set of token sequences, in general we have to describe $\mathbf{WL_0}$-induced extensions of any finite state machine function. We postpone this problem to a future work, and now suggest a simple subclass of the finite-state-machine functions whose extensions are monotone lattice mappings.

Definition V.1. *Given alphabets Σ and Σ', let h be a string morphism being defined by the mapping $h' : \Sigma \to \Sigma'^{*}$. We use the same name h for the following extension of h over the lattice elements.*

- $h\big(\mathrm{Eq}_{\xi}\big) = \mathrm{Eq}_{h(\xi)}$
- $h\big(\mathrm{Cnj}_{\tau_1,\tau_2}\big) = \mathrm{Cnj}_{\rho\left(h(\tau_1)\right),\,\rho\left(h(\tau_1\tau_2)-\rho(h(\tau_1))\right)}$

We recall the following classical lemma [20], which ensures that h is monotonic wrt the lattice order.

Lemma V.1. *If σ is a solution to equation $\mathcal{U} = \mathcal{V}$, and h is a morphism, then $h \circ \sigma$ is a solution to $h\big(\mathcal{U}\big) = h\big(\mathcal{V}\big)$.*

Given any values $x, y \in \mathbf{WL_0}$ and a string morphism h, we can now show that $(h(x) \vee h(y)) \le h(x \vee y)$. Moreover, in case $x \vee y \ne \top$, $h(x \vee y) = h(x) \vee h(y)$, due to Lemma V.1.

Let $\preceq$ be a linear order on Σ, and $\preceq_*$ be the length-lexicographical[2] order on Σ^* induced by $\preceq$. Given a string morphism $h : \Sigma \to \Sigma'^{*}$ s.t. $\forall a \in \Sigma\big(h(a) \ne \varepsilon\big)$, we define its minimal inverse mapping $h^{-1}_{\min} : \Sigma'^{*} \to \Sigma^*$ as follows.

$$h^{-1}_{\min}(\xi) = \tau \text{ s.t. } h(\tau) = \xi \;\&\; \forall \tau'\big(h(\tau') = \xi \Rightarrow \tau \preceq_* \tau'\big)$$

In general, the inverse image $h^{-1}_{\min}$ does not respect the lattice order. For example, given distinct $a \succ b \succ c$, if $h(c) = aba$, $h(a) = a$, $h(b) = b$, then $h^{-1}_{\min}(abab) = cb$, and $h^{-1}_{\min}\big(\mathrm{Eq}_{ababa}\big) \vee h^{-1}_{\min}\big(\mathrm{Eq}_{ab}\big) = \top$, while $h^{-1}_{\min}\big(\mathrm{Eq}_{ababa} \vee \mathrm{Eq}_{ab}\big) = h^{-1}_{\min}\big(\mathrm{Cnj}_{\varepsilon,ab}\big)$.

In order to address the monotonicity issue, we choose a special subset of string morphisms whose inverse mappings extensions can be used as lattice morphisms. After the paper [16], we say that an infix ξ of a word $a_1 \dots a_n$ contains a border between subwords $a_1 \dots a_k$ and $a_{k+1} \dots a_n$, if the word ξ includes an infix $a_{k-j_1} \dots a_{k+j_2}$, where $j_1 \ge 0$ and $j_2 > 0$. Given any predicate $\mathrm{Cnj}_{\xi_1,\xi_2}$ in lattice $\mathcal{L}$, the inverse mappings we consider preserve the borders between ξ_1 and ξ_2, as well as between ξ_2 and ξ_1.

Formally, given a lattice $\mathcal{L}$, the inverse mapping of a morphism h is border-preserving wrt $\mathcal{L}$, if for any lattice element of the form $\mathrm{Cnj}_{a_1\dots a_k, a_{k+1}\dots a_n}$ and for any $b \in \Sigma'$, the morphism $h(b)$ is equal neither to $\xi_2(a_1 \dots a_n)^m\xi_1$ nor to ξ_3, for any $m \in \mathbb{N}$, $\xi_1, \xi_2, \xi_3 \in \Sigma^*$ satisfying the following conditions:

- ξ_1 is a prefix of $a_1 \dots a_n$; ξ_2 is a suffix of $a_1 \dots a_n$; $|\xi_1\xi_2| > 0$, $|\xi_i| < n$, and either $(|\xi_1| > 0)$ & $(|\xi_2| > 0)$, or $m > 0$,
- ξ_3 is $a_{k-j_1} \dots a_{k+j_2}$, where $k > j_1 \ge 0$ and $n - k \ge j_2 > 0$, and $|\xi_3| < n$.

Hence, there are the two possibilities to violate the border-preserving condition: $h(b)$ equals to an infix of $a_1 \dots a_n$ containing the border between $a_1 \dots a_k$ and $a_{k+1} \dots a_n$, or to a subword of $(a_1 \dots a_n)^m$ containing at least one border between the occurrences of $a_1 \dots a_n$. The border-preserving condition does not depend on the order induced on Σ in the definition of $h^{-1}_{\min}$, because the condition holds for images of all elements of Σ'.

Example V.1. *Given a lattice including the element $\mathrm{Cnj}_{\varepsilon,ab}$ and $c \ne a$, $c \ne b$, the inverse of any morphism h s.t. $\exists c \in \Sigma'\big(h(c) = aba\big)$ is not border-preserving: $h(c)$ includes the border between two occurrences of ab.*

[2] If $|\omega_1| < |\omega_2|$, then $\omega_1 \preceq_* \omega_2$; if $|\omega_1| = |\omega_2|$, then the order $\preceq$ is used lexicographically.

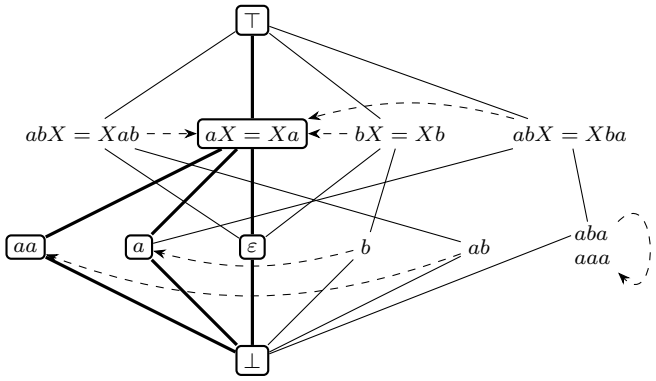


Fig. 4: Isotonic lattice mapping using the string morphism $h_{\Sigma\to a}$ of the lattice given in Fig. 1. The sublattice of the fixed points of $h_{\Sigma\to a}$ is given in framed nodes and thick edges. Dashed arcs point to the morphic images of the elements.

Given a lattice including the element $\mathrm{Cnj}_{aba,ba}$*, neither of the inverses of morphisms* h_1*,* h_2 *s.t.* $\exists c_1, c_2 \in \Sigma'(h_1(c_1) = bab \;\&\; h_2(c_2) = aa)$ *is border-preserving. The* bab *value of* $h_1(c_1)$ *includes the border between* aba *and* ba*, given* $\xi_3 = bab$*,* $k = 3$*,* $j_1 = 1$*,* $j_2 = 1$*; the* aa *value of* $h_2(c_2)$ *contains the border between* ba *and* aba*.*

Lemma V.2. *Given lattice* $\mathbf{WL_0}$ *and a string morphism* h *satisfying the conditions above, for any order induced on* Σ*,* $h^{-1}_{\min}$ *is a lattice morphism.*

Proof. Given $x, y \in \mathcal{L}$, let us show that $h^{-1}_{\min}(x \vee y)$ and $h^{-1}_{\min}(x) \vee h^{-1}_{\min}(y)$ are equal.

If x and y are both of the form $\mathrm{Cnj}_{\tau_i,\tau_j}$, then their join is non-trivial iff $x = y$; and the equality is trivially preserved.

Given $x = \mathrm{Eq}_{\xi_1}$ and $y = \mathrm{Eq}_{\xi_2}$, if they are equal or their join is $\top$, the morphism property holds trivially. Let us consider the case when $x \vee y$ is $\mathrm{Cnj}_{\tau_1,\tau_2}$, then for any $b \in \Sigma', \xi'$ s.t. $h(b) = \xi'$ and $\xi_i = \xi_{i,p}\xi'\xi_{i,s} = (\tau_1\tau_2)^{k_i}\tau_1$, the occurrence of ξ' can appear strictly inside the subwords τ_1 and τ_2 of ξ_1 and ξ_2. Thus $h^{-1}_{\min}\big(\mathrm{Eq}_{\xi_1} \vee \mathrm{Eq}_{\xi_2}\big) = \mathrm{Cnj}_{h^{-1}_{\min}(\tau_1),h^{-1}_{\min}(\tau_2)} = h^{-1}_{\min}\big(\mathrm{Eq}_{\xi_1}\big) \vee h^{-1}_{\min}\big(\mathrm{Eq}_{\xi_2}\big)$.

Given $x = \mathrm{Eq}_{\xi}$ and $y = \mathrm{Cnj}_{\tau_1,\tau_2}$, let us compute $h^{-1}_{\min}\big(x \vee y\big)$ when the join is non-trivial. In this case $\xi = (\tau_1\tau_2)^k\tau_1$, and again all the subwords ξ' s.t. $\exists b \in \Sigma'(h(b) = \xi')$ can occur only inside τ_1 or τ_2, thus the resulting join is $\mathrm{Cnj}_{h^{-1}_{\min}(\tau_1),h^{-1}_{\min}(\tau_2)}$.

The meet case is symmetric to the join case. Note that both by the definition of $h^{-1}_{\min}$ and choice of h, $h^{-1}_{\min}(\varepsilon) = \varepsilon$. □

Therefore, compositions of the border-preserving mappings with the string morphisms result in monotonic lattice mappings. Hence, the image lattices can be analysed with the same algorithms as the lattice described in Section III.

Moreover, if a string morphism maps elements of Σ into elements of Σ^* (i.e. acts in the same alphabet), then by the Knaster–Tarski theorem [14] the set of its fixpoints forms a complete sublattice of the lattice $\mathbf{WL_0}$, and no additional construction is required to track the properties captured by the morphism. Such a sublattice is shown in Fig. 4 in framed nodes and thick edges.

Tracing both the properties given in the lattice $\mathbf{WL_0}$ and in its sublattices wrt the string morphisms can be useful, for example, in the following analyses.

First, we can obtain a simple length analysis of strings, tracking constant values of string lengths. Indeed, the morphism $h_{\Sigma\to a}$ defined as $\forall c \in \Sigma\big(h_{\Sigma\to a}(c) = a\big)$ maps any given string to the unary Peano number representing its length.

Second, we can obtain a symbol occurrence analysis. Occurrences of forbidden symbols (i.e. all symbols from a set $\Sigma' \subset \Sigma$) can be traced in the sublattice produced by the morphism $h_{\Sigma\setminus\Sigma'\to\varepsilon}$ defined as follows: $\forall c \in \Sigma'\big(h_{\Sigma\setminus\Sigma'\to\varepsilon}(c) = a\big) \;\&\; \forall c \in \Sigma \setminus \Sigma'\big(h_{\Sigma\setminus\Sigma'\to\varepsilon}(c) = \varepsilon\big)$.

Third, simple string classification wrt the letter sets that are contained in the strings may be done. Namely, given $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \ldots \Sigma_k$, where $\forall i, j(i \neq j \Rightarrow \Sigma_i \cap \Sigma_j = \varnothing)$, the morphism $h_{\Sigma_1,\ldots,\Sigma_k}$ mapping any symbol from Σ_i to a single letter a_i produces a sublattice capturing abstract properties "to contain only the letters belonging to the set Σ_i".

VI. Related Works and Discussion

Practical string analysis tools [4], [8], [10], [23] tend to apply combinations of string domains. A natural abstraction of string sets may be expressed in terms of the regular language $\mathbf{RL}$ [6], [7], [9]. The $\mathbf{RL}$-based abstract domains are easily defined given the union and intersection operations, however, the set of all regular languages cannot be directly used as an abstract domain. The main two problems to be solved when using the $\mathbf{RL}$-based domains are listed below.

First, the lattice based on $\mathbf{RL}$ abstract domain admits infinite ascending chains. Thus, the straightforward usage of the regular expressions as elements of the lattice results in non-termination of the abstract interpretation.

Second, when the first problem is addressed via widening, infinite descending chains can still remain. The termination issue of the abstract interpretation relies only on the upper semi-lattice completeness. But the existence of such descending chains may indicate that the convergence speed is not uniformly bounded wrt the program to be analysed. For example, if the two strings start with the same common prefix

ξ, and ξ is long enough, then their widening to $\xi.*$, where $.*$ defines an arbitrary string, (as defined in the paper [6]) may result in $|\xi|$ iterations computing the upper bound, if the loop dropping the first letter of a word is analysed. The widening defined in the book [7] shares this feature as well.

The paper [9] discussed the following four abstract string domains often used in practical string analysis.

The first is an abstract domain with values Eq_ξ. This domain is included in $\mathbf{WL_0}$ presented in this paper.

The second is an abstract domain with values tracking the string lengths. Its simplest version can be modelled in a sublattice of $\mathbf{WL_0}$ by means of the morphism $h_{\Sigma\to a}$. Versions of the string length domain involving more complex length properties are independent from $\mathbf{WL_0}$ and can be used in the direct product with $\mathbf{WL_0}$ in order to improve preciseness of the analysis [24].

The third is an abstract domain with values representing predicates "string X contains a letter a". If the known set of the values $\Sigma' \subset \Sigma$ is tracked, then this domain is embedded in $\mathbf{WL_0}$ as a set of sublattices, by means of providing the set of string morphisms $h_{\Sigma\setminus\{c\}\to\varepsilon}$ mapping a chosen $c \in \Sigma'$ into itself, and all the other letters from Σ to ε.

The fourth is an abstract domain with values representing prefix predicates "string X starts with ξ", and the corresponding domain of suffix predicates. This domain contains infinite descending chains, since if $\xi_1\xi_2$ is a prefix of X, then ξ_1 (including $\xi_1 = \varepsilon$) is also a prefix of X.

The authors of the paper [10] use an abstract string domain separating unknown strings into numeric, non-numeric and special strings reflecting key words of JS syntax. Although the string domain is hard-coded, the JSAI tool makes use of configurable sensitivity in the trace analysis, thus allowing a user to redefine the tracked breakpoints. Thus, the idea presented in this paper to make the string analysis configurable by constructing the tokenizer mapping can be considered as an attempt to make the string-specified domain more configurable.

The works combining expressiveness of the word equation languages and regular languages emerged in program verification for finding loop invariants and pruning unreachable computation branches, see e.g. the paper [23]. There the straight-line fragment of the word equation language is considered, i.e. the variables in an equation cannot occur more than once. Nevertheless, such a fragment can still express some of language properties that are non-expressible by means of the multi-track finite automata [7].

The fresh work [25] reasons on so-called chain-free word equations, in which the variable dependences are bounded. The decision procedures for the existential theory of the chain-free equations together with regular constraints are given.

A. Complexity of operations in $\mathbf{WL_0}$

Several operations presented in this paper depend on finding either a primitive root of a string or its maximal suffix and prefix being a power of the same primitive word, given concatenation, string subtraction, and replacement operators. This task can be efficiently solved, e.g. by means of suffix arrays and LSP arrays, hence, the resulting complexity of the operations over abstract values x and y can be estimated as $O\big((|x| + |y|)\log(|x| + |y|)\big)$, where $|\,\mathrm{Eq}_\omega\,| = |\omega|$, and $|\,\mathrm{Cnj}_{\xi_1,\xi_2}\,| = |\xi_1| + |\xi_2|$.

VII. Conclusion and Future Work

In this paper, we have presented a first attempt to use the word equations as a basic language for constructing a string abstract domain. We have introduced the first, quantifier-free, layer of the resulting lattice $\mathbf{WL_0}$, together with the interpretation of the usual string processing operations in the domain based on the lattice $\mathbf{WL_0}$.

Extending the lattice with existential predicates (i.e. equations involving at least two variables) is an interesting and non-trivial task. A simplest choice of the existential two-variable predicates is to take one-variable patterns, i.e. equations of the form $X = \xi_1 Y \xi_2 \dots Y \xi_{n+1}$. However, if we consider arbitrary one-variable patterns, the domain will include unbounded descending chains, e.g. $X = a_1 \dots a_n Y, \dots, X = a_1 Y$, which can make abstract interpretation too slow. Moreover, some predicates of this form can violate the upper semilattice condition. E.g. given an abstract value of the form $\mathrm{Cnj}_{\xi_1,\xi_2}$ if the predicate $\exists Y(X = YY)$ is also introduced as an abstract value, then $\mathrm{Eq}_{aa} \vee \mathrm{Eq}_\varepsilon$ becomes undefined, because we cannot choose the least element from $\mathrm{Cnj}_{\varepsilon,a}$ and $\exists Y(X = YY)$ unless we introduce additional lattice layers beyond the layer consisting of elementary equations.

Nevertheless, the patterns are the interesting and expressible language to be considered as a closest development of $\mathbf{WL_0}$.

Definitely there are other possibilities of the lattice enhancing, e.g. with the balanced two-variable equations. An equation is called balanced if multisets of the terms in its left- and right-sides coincide. For example, the predicates of the form $\exists Y(X\omega_1\omega_3\omega_2 Y = Y\omega_2\omega_3\omega_1 X)$, as well as the patterns, are basic in languages of two-variable equations, as shown in the paper [26]. I.e. an infinite language of any two-variable equation wrt variable X either contains a pattern or words satisfying the predicate $\exists Y(X\omega_1\omega_3\omega_2 Y = Y\omega_2\omega_3\omega_1 X)$, maybe intersected with a language of $\mathrm{Cnj}_{\xi_1,\xi_2}$. This approach has several benefits. First, the balanced two-variable equations as the abstract values can capture non-trivial properties of one-variable solution set projections, e.g. the X-solution set of the equation $XaYYb = YaXbY$ is $\big\{(b^n a)^m b^n \mid m, n \in \mathbb{N}\big\}$, describing a non-regular property of the X value. Second, the two-variable equations are able to express relations between concrete values over that the given variables can range. E.g., the solution set of the equation $XaY = YaX$ is $\big\{((\omega a)^*\omega, (\omega a)^*\omega) \mid \omega \in \Sigma^*\big\}$, where ω is a word parameter. While both X- and Y-projections of the solution set are trivial, the whole set indicates that X and Y values consist of repetitions of the same substring, separated with the letter a. In the case a solution-set description includes word parameters a static analysis may track not only known but also unknown repeated substrings in the data to be analysed.

If we do not restrict the equations with k variables, then construction of joins and meets becomes even harder, since, e.g. the pattern language inclusion is undecidable [27].

ACKNOWLEDGEMENTS

The author of the paper thanks Andrei Nemytykh for the support and many fruitful discussions on the draft of this paper, Denis Fokin for posing the problem of investigating string domain lattices; and anonymous referees for their valuable comments on the paper.

The research has been partially supported by Huawei Technologies Co. Ltd., and by Russian Academy of Sciences, research project №122012700089-0.

REFERENCES

[1] J. D. Day, V. Ganesh, N. Grewal, and F. Manea, "Formal languages via theories over strings," *CoRR*, vol. abs/2205.00475, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2205.00475

[2] V. G. Durnev, "Undecidability of a simple fragment of a positive theory with a single constant for a free semigroup of rank 2, (in Russian)," *Matem. Zametki*, vol. 67, pp. 191–200, 2000. [Online]. Available: https://doi.org/10.4213/mzm827

[3] N. Bjørner, N. Tillmann, and A. Voronkov, "Path feasibility analysis for string-manipulating programs," in *Tools and Algorithms for the Construction and Analysis of Systems*, S. Kowalewski and A. Philippou, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 307–321. [Online]. Available: https://doi.org/10.1007/978-3-642-00768-2_27

[4] A. Reynolds, A. Nötzli, C. W. Barrett, and C. Tinelli, "Reductions for strings and regular expressions revisited," in *2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24, 2020*. IEEE, 2020, pp. 225–235. [Online]. Available: https://doi.org/10.34727/2020/isbn.978-3-85448-042-6_30

[5] B. Eriksson, A. Stjerna, R. D. Masellis, P. Rümmer, and A. Sabelfeld, "Black Ostrich: Web application scanning with string solvers," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, W. Meng, C. D. Jensen, C. Cremers, and E. Kirda, Eds. ACM, 2023, pp. 549–563. [Online]. Available: https://doi.org/10.1145/3576915.3616582

[6] T. Choi, O. Lee, H. Kim, and K. Doh, "A practical string analyzer by the widening approach," in *Programming Languages and Systems, 4th Asian Symposium, APLAS 2006, Sydney, Australia, November 8-10, 2006, Proceedings*, ser. Lecture Notes in Computer Science, N. Kobayashi, Ed., vol. 4279. Springer, 2006, pp. 374–388. [Online]. Available: https://doi.org/10.1007/11924661_23

[7] T. Bultan, F. Yu, M. Alkhalaf, and A. Aydin, *String Analysis for Software Verification and Security*. Springer, 2017. [Online]. Available: https://doi.org/10.1007/978-3-319-68670-7

[8] B. Loring, D. Mitchell, and J. Kinder, "Sound regular expression semantics for dynamic symbolic execution of JavaScript," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019*, K. S. McKinley and K. Fisher, Eds. ACM, 2019, pp. 425–438. [Online]. Available: https://doi.org/10.1145/3314221.3314645

[9] V. Arceri, M. Olliaro, A. Cortesi, and I. Mastroeni, "Completeness of abstract domains for string analysis of JavaScript programs," in *Theoretical Aspects of Computing - ICTAC 2019 - 16th International Colloquium, Hammamet, Tunisia, October 31 - November 4, 2019, Proceedings*, ser. Lecture Notes in Computer Science, R. M. Hierons and M. Mosbah, Eds., vol. 11884. Springer, 2019, pp. 255–272. [Online]. Available: https://doi.org/10.1007/978-3-030-32505-3_15

[10] V. Kashyap, K. Dewey, E. A. Kuefner, J. Wagner, K. Gibbons, J. Sarracino, B. Wiedermann, and B. Hardekopf, "JSAI: a static analysis platform for JavaScript," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014*, S. Cheung, A. Orso, and M. D. Storey, Eds. ACM, 2014, pp. 121–132. [Online]. Available: https://doi.org/10.1145/2635868.2635904

[11] F. Logozzo and H. Venter, "RATA: rapid atomic type analysis by abstract interpretation - application to JavaScript optimization," in *Proceedings of the 19th Joint European Conference on Theory and Practice of Software, International Conference on Compiler Construction*, ser. CC'10/ETAPS'10. Berlin, Heidelberg: Springer-Verlag, 2010, p. 66–83. [Online]. Available: https://doi.org/10.1007/978-3-642-11970-5_5

[12] ECMA. (2023) ECMAScript specification. [Online]. Available: https://ecma-international.org/publications-and-standards/standards/ecma-262/

[13] G. Birkhoff and C. Bartee, *Modern Applied Algebra*, 1970.

[14] A. Tarski, "A lattice-theoretical fixpoint theorem and its applications," *Pacific J. Math.*, vol. 5, no. 2, pp. 285–309, 1955. [Online]. Available: https://doi.org/10.2140/pjm.1955.5.285

[15] G. S. Makanin, "The problem of solvability of equations in a free semigroup," *Mat. Sb. (N.S.)*, vol. 103(145), pp. 147–236, 1977. [Online]. Available: https://doi.org/10.1070/SM1977v032n02ABEH002376

[16] J. Karhumäki, F. Mignosi, and W. Plandowski, "The expressibility of languages and relations by word equations," *J. ACM*, vol. 47, no. 3, pp. 483–505, May 2000. [Online]. Available: http://doi.acm.org/10.1145/337244.337255

[17] J. I. Hmelevskij, "Equations in a free semigroup, (in Russian)," *Trudy Mat. Inst. Steklov*, vol. 107, p. 286, 1971. [Online]. Available: https://www.mathnet.ru/rus/tm2975

[18] C. Choffrut and J. Karhumäki, "Combinatorics of words," in *Handbook of Formal Languages, Volume 1: Word, Language, Grammar*, G. Rozenberg and A. Salomaa, Eds. Springer, 1997, pp. 329–438. [Online]. Available: https://doi.org/10.1007/978-3-642-59136-5_6

[19] P. Cousot and R. Cousot, "Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977*, R. M. Graham, M. A. Harrison, and R. Sethi, Eds. ACM, 1977, pp. 238–252. [Online]. Available: https://doi.org/10.1145/512950.512973

[20] A. Saarela, "Systems of word equations, polynomials and linear algebra: A new approach," *Eur. J. Comb.*, vol. 47, pp. 1–14, 2015. [Online]. Available: https://doi.org/10.1016/j.ejc.2015.01.005

[21] N. K. Kosovskij, "Some properties of solutions to equations in a free semigroup (in Russian)," *Zap. Nauch. Sem. LOMI*, vol. 32, pp. 21–28, 1972. [Online]. Available: https://www.mathnet.ru/rus/znsl2560

[22] D. Nowotka and A. Saarela, "An optimal bound on the solution sets of one-variable word equations and its consequences," *SIAM J. Comput.*, vol. 51, no. 1, pp. 1–18, 2022. [Online]. Available: https://doi.org/10.1137/20m1310448

[23] T. Chen, A. Flores-Lamas, M. Hague, Z. Han, D. Hu, S. Kan, A. W. Lin, P. Rümmer, and Z. Wu, "Solving string constraints with regex-dependent functions through transducers with priorities and variables," *Proc. ACM Program. Lang.*, vol. 6, no. POPL, pp. 1–31, 2022. [Online]. Available: https://doi.org/10.1145/3498707

[24] T. Chen, M. Hague, J. He, D. Hu, A. W. Lin, P. Rümmer, and Z. Wu, "A decision procedure for path feasibility of string manipulating programs with integer data type," in *Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings*, ser. Lecture Notes in Computer Science, D. V. Hung and O. Sokolsky, Eds., vol. 12302. Springer, 2020, pp. 325–342. [Online]. Available: https://doi.org/10.1007/978-3-030-59152-6_18

[25] D. Chocholatý, T. Fiedor, V. Havlena, L. Holík, M. Hruška, O. Lengál, and J. Síč, "Mata: A fast and simple finite automata library," in *Tools and Algorithms for the Construction and Analysis of Systems*, B. Finkbeiner and L. Kovács, Eds. Cham: Springer Nature Switzerland, 2024, pp. 130–151. [Online]. Available: https://doi.org/10.1007/978-3-031-57249-4_7

[26] L. Ilie and W. Plandowski, "Two-variable word equations," *RAIRO Theor. Informatics Appl.*, vol. 34, no. 6, pp. 467–501, 2000. [Online]. Available: https://doi.org/10.1051/ita:2000126

[27] T. Jiang, A. Salomaa, K. Salomaa, and S. Yu, "Inclusion is undecidable for pattern languages," in *Automata, Languages and Programming*, A. Lingas, R. Karlsson, and S. Carlsson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 301–312. [Online]. Available: https://dl.acm.org/doi/10.5555/646247.684876

APPENDIX

Computation of $x - y$

A word ξ is said to be a fractional power of τ, if $\exists\omega_1,\omega_2,k\big(k \in \mathbb{N} \;\&\; \tau = \omega_1\omega_2 \;\&\; \xi = \tau^k\omega_1\big)$. The fractional power of τ in ξ is computed as $k + \frac{|\omega_1|}{|\tau|}$ [18]. E.g. $abaabaa = (aba)^{2\frac{1}{3}}$. Hence, words satisfying any predicate $\mathrm{Cnj}_{\omega_1,\omega_2}$ are fractional powers of $\omega_1\omega_2$ with the non-integer fractional part consisting of ω_1 (the fractional part is non-empty if $|\omega_1| \neq 0$). We recall that the operation $a(x)$ takes a concrete value of a predicate x.

Below the definition of the prefix subtraction operation in $\mathbf{WL_0}$ is repeated.

$$x - y = \begin{cases} \mathrm{Eq}_\tau, \text{ if } x = \mathrm{Eq}_{\xi\tau} \;\&\; y = \mathrm{Eq}_\xi; \\ \text{error, if } x = \mathrm{Eq}_\tau \;\&\; y = \mathrm{Eq}_\xi \;\&\; \forall\tau'(\tau \neq \xi\tau'); \\ \text{error, if } x = \mathrm{Cnj}_{\tau_1,\tau_2} \;\&\; y = \mathrm{Eq}_\xi \\ \qquad\qquad \&\; \neg(y \leq \mathrm{Cnj}_{\xi_1,\xi_2}), \text{ where } \tau_1\tau_2 = \xi_1\xi_2; \\ x, \text{ if } y = \mathrm{Cnj}_{\varepsilon,\xi} \text{ and} \\ \quad \text{either} \quad x = \mathrm{Eq}_\tau \quad \text{and} \quad \forall\tau'\big(\tau \neq \xi\tau'\big), \\ \quad \text{or } x = \mathrm{Cnj}_{\tau_1,\tau_2} \text{ and } \forall\tau', k\big((\tau_1\tau_2)^k\tau_1 \neq \xi\tau'\big); \\ \top, \text{ if } y = \mathrm{Cnj}_{\xi_1,\xi_2} \text{ and } x - y \text{ can satisfy at least} \\ \quad \text{two different predicates of the form } \mathrm{Cnj}_{\omega_1,\omega_2}; \\ \mathrm{Cnj}_{\tau_{1,2},\tau_2\tau_{1,1}}, \text{ if } x \leq \mathrm{Cnj}_{\tau_1,\tau_2} \text{ and } y \leq \mathrm{Cnj}_{\xi_1,\xi_2} \\ \qquad\qquad \text{and } \exists k,k',\tau_{1,1},\tau_{1,2} \\ \qquad \big((\tau_1\tau_2)^k\tau_{1,1} = (\xi_1\xi_2)^{k'}\xi_1 \;\&\; \tau_1 = \tau_{1,1}\tau_{1,2}\big); \\ \mathrm{Cnj}_{\tau_{2,2}\tau_1,\tau_{2,1}}, \text{ if } x \leq \mathrm{Cnj}_{\tau_1,\tau_2} \text{ and } y \leq \mathrm{Cnj}_{\xi_1,\xi_2} \\ \qquad\qquad \text{and } \exists k,k',\tau_{2,1},\tau_{2,2} \\ \quad \big((\tau_1\tau_2)^k\tau_1\tau_{2,1} = (\xi_1\xi_2)^{k'}\xi_1 \;\&\; \tau_2 = \tau_{2,1}\tau_{2,2}\big); \\ \top, \text{ otherwise.} \end{cases}$$

The first three cases of the definition are self-explanatory; now we consider the case returning x. Given $y = \mathrm{Cnj}_{\varepsilon,\xi}$, if any concrete value of $a(x)$ does not start with ξ, the only value of $a(y)$ that can be subtracted from $a(x)$ is ε, i.e. ξ^0.

In the following cases, we treat the predicate $x = \mathrm{Eq}_\tau$ uniformly with $\mathrm{Cnj}_{\tau_1,\tau_2}$, making use of the fact that any τ can be represented as $(\tau_1\tau_2)^m\tau_1$, where $\tau_1 = \varepsilon$, $\tau_2 = \rho(\tau)$.

Let $x = \mathrm{Cnj}_{\tau_1,\tau_2}$ or $x = \mathrm{Eq}_{(\tau_1\tau_2)^m\tau_1}$, $y = \mathrm{Cnj}_{\xi_1,\xi_2}$, and let $a(x)$ start with $a(y)$. Then $\exists k_1\big(a(x) = (\tau_1\tau_2)^{k_1}\tau_1\big)$; $\exists k_2\big(a(y) = (\xi_1\xi_2)^{k_2}\xi_1\big)$; $\exists\tau'\big(a(x) = a(y)\tau'\big)$.

If both ξ_1 and $\xi_1\xi_2\xi_1$ are fractional powers of $\tau_1\tau_2$ and the fractional parts of $\tau_1\tau_2$ in ξ_1 and $\xi_1\xi_2\xi_1$ do not coincide, then the abstract value of $a(x) - a(y)$ cannot be determined, and the value $\top$ is returned by the computation of $x - y$ (shown in the fifth case of the definition).

In the remaining cases below the fifth (the one returning $\top$), we assume that either ξ_1 and $\xi_1\xi_2\xi_1$ are powers of $\tau_1\tau_2$ and the fractional parts of $\tau_1\tau_2$ in ξ_1 and $\xi_1\xi_2\xi_1$ coincide, or that ξ_1 is a power of $\tau_1\tau_2$, while $\xi_1\xi_2\xi_1$ is not.

If the string $a(y)$ ends inside the string τ_1, then $\tau_1 = \tau_{1,1}\tau_{1,2}$, and $\exists k_3\big(\tau' = (\tau_{1,2}\tau_2\tau_{1,1})^{k_3}\tau_{1,2}\big)$. Hence, τ' satisfies the predicate $\mathrm{Cnj}_{\tau_{1,2},\tau_2\tau_{1,1}}$ (the sixth case of the definition).

If the string $a(y)$ ends inside the string τ_2, then $\tau_2 = \tau_{2,1}\tau_{2,2}$, and $\exists k_3\big(\tau' = (\tau_{2,2}\tau_1\tau_{2,1})^{k_3}\tau_{2,2}\tau_1\big)$. Hence, τ' satisfies the predicate $\mathrm{Cnj}_{\tau_{2,2}\tau_1,\tau_{2,1}}$ (the seventh case of the definition).

Note that the order of the right-hand sides of the interpretation rule for $x - y$ guarantees that if $a(y)$ satisfies the predicate $\mathrm{Cnj}_{\tau_1,\tau_2}$ then the resulting abstract value is $\mathrm{Cnj}_{\varepsilon,\tau_2\tau_1}$, but not $\mathrm{Cnj}_{\tau_2\tau_1,\varepsilon}$. These two predicates are equivalent, but only the former is consistent with the definition of the abstract values in $\mathbf{WL_0}$.