Algorithmica (2011) 60: 819–828
DOI 10.1007/s00453-009-9375-3

On Word Equations in One Variable

Robert Dąbrowski · Wojciech Plandowski

Received: 3 December 2008 / Accepted: 11 November 2009 / Published online: 2 December 2009

Abstract For a word equation E of length n in one variable x occurring $\#_x$ times in E a resolution algorithm of $O(n + \#_x \log n)$ time complexity is presented here. This is the best result known and for the equations that feature $\#_x < \frac{n}{\log n}$ it yields time complexity of $O(n)$ which is optimal. Additionally it is proven here that the set of solutions of any one-variable word equation is either of the form F or of the form $F \cup (uv)^+u$ where F is a set of $O(\log n)$ words and u, v are some words such that uv is a primitive word.

Keywords Word equation · Equation in free semigroup · Algorithm analysis and design · Computational complexity

1 Introduction

In 1977 Makanin proved that solvability of word equations is decidable in general case. Makanin's algorithm [8] has remained one of the most famous and complicated algorithms in theoretical computer science. The algorithm takes as an input a word equation and decides whether the equation has a solution or not. It has been improved several times and its best version [5] works in EXPSPACE.

Recently new algorithms that decide solvability of word equations in general case have been found by Plandowski and Rytter [12, 13]. The first one works nondeterministically in polynomial time in respect to the length of the input equation and the logarithm of the length of its minimal solution. Since the currently best upper bound

R. Dąbrowski (✉) · W. Plandowski
Institute of Informatics, University of Warsaw, Banacha 2, 02-097 Warsaw, Poland
e-mail: r.dabrowski@mimuw.edu.pl

W. Plandowski
e-mail: wojtekpl@mimuw.edu.pl

for the length of the minimal solution is double exponential [11], with this bound the algorithm in [13] works in NEXPTIME. The algorithm in [12] works in PSPACE.

Obviously the algorithms solving the problem of satisfiability of word equations in general case cannot be called efficient. Actually we cannot even expect efficiency since the problem is known to be NP-hard [1, 7].

However, if we concentrate on some selected classes of word equations then there do exist polynomial time algorithms that either decide solvability (decide if a given word equation has any solutions) or actually solve word equations (find a polynomial description of all solutions of a given word equation); the latter is clearly a more general problem.

In case of two-variable word equations there exist two polynomial time algorithms [2, 6] that decide solvability. The best one [6] works in time $O(n^6)$, where n is the length of the input equation. Although the algorithm is of polynomial time complexity, it cannot be called efficient. In case of one-variable word equations there exists an efficient algorithm [10] which works in time $O(n \log n)$, where n is the length of the input equation. There exists also an efficient $O(n \log^2 n)$ time algorithm for a certain class of special equations with two variables [9].

In this paper we deepen the analysis of Goralcik et al. [10] and present an algorithm that works in time $O(n + \#_x \log n)$ where n is the length of the input equation and $\#_x$ is the number of occurrences of the variable x in the input equation. Our algorithm consists of two phases: in the first phase it selects $O(\log n)$ candidates for a solution, in the second phase it verifies in time $O(\#_x)$ if a candidate is a solution. The second phase assumes that the alphabet Σ is of constant size (or that it is a set of numbers $\{1, \ldots, |\Sigma|\}$).

Additionally we prove that the set of solutions of any one-variable word equation is either of the form F or of the form $F \cup (uv)^+u$ where F is a set of $O(\log n)$ words and u, v are some words such that word uv is primitive.

2 Preliminaries

A *word equation* over an alphabet Σ in one variable $x \notin \Sigma$ is a pair $E = (L, R)$ of words $L, R \in (\Sigma \cup \{x\})^\star$ and is usually written as $E : L = R$. A solution of E is a homomorphism $\varphi : (\Sigma \cup \{x\})^\star \to \Sigma^\star$ leaving the letters of Σ invariant (i.e. $\varphi(a) = a$ for all $a \in \Sigma$) and such that $\varphi(L) = \varphi(R)$. Each solution is uniquely identified by a mapping of x into $\Sigma^\star$ for which the homomorphism is the canonical extension. By $Sol(E)$ we denote the *solution set* of the equation E. The *length* of the equation E, denoted by $|E|$, is the number $|LR|$.

The *prefix*, *proper prefix*, *suffix* and *proper suffix* relations on words are denoted respectively by $\sqsubseteq$, $\sqsubset$, $\sqsupseteq$ and $\sqsupset$. For a word B, B^ω denotes an infinite word which is an infinite repetition of the word B.

We start by recalling basic facts. Words $A, B \in \Sigma^\star$ are *conjugate* iff there exist two words $u, v \in \Sigma^\star$ such that $A = uv$ and $B = vu$. A non-empty word $w \in \Sigma^\star$ is *primitive* iff it has exactly $|w|$ distinct conjugations or, equivalently, if it is not a power of a different word. For arbitrary words A and w we define $A/w = \max\{k : w^k \sqsubseteq A\}$. Every non-empty word $A \in \Sigma^\star$ is a power of a unique primitive word $w \in \Sigma^\star$ called its *primitive root*, that is $A = w^{A/w}$.

Proposition 1 (See [3]) *For an arbitrary word A let $u, v, w \sqsubseteq A$ be distinct primitive words such that $u^2 \sqsubset v^2 \sqsubset w^2 \sqsubseteq A$. Then $|u| + |v| \leq |w|$.*

By $Rpp(A)$ we denote the sequence of all *repetitive primitive prefixes* of A, that is all primitive $w \sqsubseteq A$ such that $w^2 \sqsubseteq A$. As an immediate consequence of Proposition 1 we have the following proposition.

Proposition 2 (See [3]) *Let $Rpp(A) = \{w_1, w_2, \ldots, w_k\}$ and $w_1 \sqsubset w_2 \sqsubset \cdots \sqsubset w_k$. Then $|w_{i+2}| \geq 2|w_i|$. Moreover $k \leq 2 \log |A|$ and $\sum_{i=1}^{k} |w_i| \leq 2|A|$.*

For given u, v we define

$$(uv)^\star u = \{(uv)^k u : k \geq 0\},$$
$$(uv)^+ u = \{(uv)^k u : k \geq 1\}.$$

A word A is called *periodic* with a *period* uv iff $A \in (uv)^+ u$.

Proposition 3 (Periodicity lemma) *Let integers p, q be some periods of a given word w. If $p + q \leq |w| + 1$ then $\gcd(p, q)$ is also a period of w.*

In a few places we shall refer to the algorithm in [10] that consists of two phases. In the first one it finds $O(\log n)$ pairs of words (u_i, v_i) such that $Sol(E) \subseteq \bigcup_i (u_i v_i)^\star u_i$. In the second phase it finds $Sol(E) \cap (u_i v_i)^\star u_i$ for all i. The latter phase contains a procedure finding (for given u, v) the set $Sol(E) \cap (uv)^\star u$ in time $O(n)$, which we will utilize by the following proposition.

Proposition 4 (See [10]) *Given two words u, v of length $O(n)$ and an equation E of length n it is possible to find the set $(uv)^+ u \cap Sol(E)$ in time $O(n)$. The problem of finding all $k \geq 1$ such that $(uv)^k u \in Sol(E)$ reduces to solving a system of linear diophantine equations in k.*

A word equation with one variable in generic form can be denoted as

$$E : A_0 x A_1 x \cdots\ x A_s = B_0 x B_1 x \cdots x B_t$$

where $A_i, B_i \in \Sigma^\star$. If the equation E has a solution then (by canceling the same leftmost and rightmost symbols from the left-hand side and right-hand side of E respectively) we can reduce the equation to the form

$$E : A_0 x A_1 x \cdots x A_s = x B_1 x \cdots x B_t$$

where A_0 is a nonempty word and either A_s is a nonempty word and B_t is the empty word or vice versa. This form of equation implies that any solution x of E is a prefix of A_0^ω.

If $s \neq t$, then by analyzing lengths of both sides of the equation we can reduce the number of possible solutions of E to at most one. By Proposition 4 the verification whether this word is a solution or not can be done in time $O(|E|)$. If $s = t = 1$, then

E is of the form $A_0x = xB_1$ which is well known in combinatorics of words. This equation has a solution only if there exist two words p, q such that pq is a primitive word and $A_0 = (pq)^i$ and $B_1 = (qp)^i$ for some i (which can be decided in time $O(|A_0| + |B_1|)$ using standard text algorithms), and then the set of all solutions is $(pq)^\star p$. In our further analysis we assume $s = t \geq 2$. Let $\#_x$ denote the number of occurrences of the variable x in E, that is $\#_x = s + t = 2s$.

3 Solution Candidates

Lemma 1 *Let E be an equation and let u, v be two words such that uv is primitive. Then*

$$Sol(E) \cap (uv)^+u = \begin{cases} \emptyset & & (0) \\ \{(uv)^k u\} & \text{for certain } k \geq 1 & (1) \\ (uv)^+u & & (\infty) \end{cases}$$

Proof By Proposition 4 the problem of computing all integers $k \geq 1$ such that $(uv)^k u \in Sol(E)$ can be reduced to solving a system of diophantine equations in k. Since the system may only have: (0) no solutions at all; (1) exactly one solution for certain $k \geq 1$; and (∞) infinitely many solutions for all $\{k : k \geq 1\}$, this completes the proof. □

To simplify the notation, for given $u, v \in \Sigma^*$ and $k \geq 0$ we will call $(uv)^k u$ a *finite solution* and $(uv)^+u$ an *infinite solution*. We show first that for a given equation there is at most one infinite solution and it can be found in linear time.

Theorem 1 *For a given equation E there exists at most one infinite solution and in time $O(|E|)$ it is possible to find the solution or decide it does not exist.*

Proof The proof falls naturally into two parts.

(1) Assume $|A_0| \leq |B_1|$. Denote by B' the prefix of B_1 of length $|B'| = |A_0|$. Fix any $X \in Sol(E)$. Then A_0X is the prefix of the left-hand side of the equation and XB' is the prefix of the right-hand of the equation. By the definition of B' the lengths of A_0X and XB' are the same, therefore $A_0X = XB'$ and consequently A_0 and B' are conjugate. Therefore the primitive roots of A_0 and B' are also conjugate and thus respectively equal to uv and vu for some $u, v \in \Sigma^\star$. Therefore $Sol(E) \subseteq (uv)^\star u$ and by Proposition 4 we can find $Sol(E)$ in time $O(|E|)$. Since $Sol(E) \subseteq (uv)^\star u$ it follows from Lemma 1 that there can be at most one infinite solution namely $(uv)^+u$.

(2) Assume $|A_0| > |B_1|$. Denote by A' the prefix of A_0 of length $|A'| = |A_0| - |B_1|$. Fix any $X \in Sol(E)$ of length$|X| \geq |A'|$. Then A' being a prefix of A_0 is also a prefix of X. Now A_0X is a prefix of the left-hand side of the equation and XB_1X is a prefix of the right-hand side of the equation. Moreover, since A' is a prefix of X, the word XB_1A' is a prefix of the right-hand side of the equation. The length of XB_1A' is $|A_0X|$. Hence $A_0X = XB_1A'$ and consequently A_0 and B_1A' are conjugate. Therefore $X \in (uv)^*u$ for some $u, v \in \Sigma^*$, uv being the primitive root

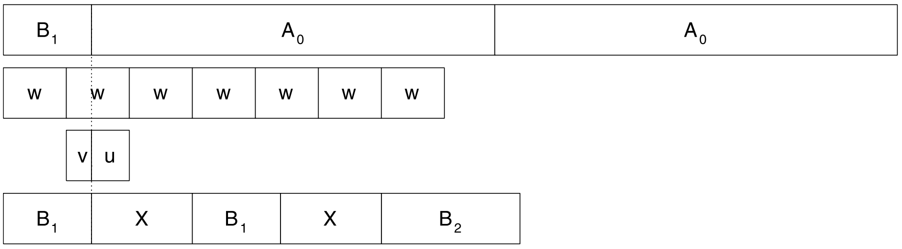

Fig. 1 The pair (u, v) is the generator for w

of A_0. Analogically to the previous case we conclude that there is at most one infinite solution and that it can be found in time $O(|E|)$. □

We follow to show that for the set of finite solutions it is possible to find in linear time its logarithmically bounded superset.

Theorem 2 *For a given equation E let S be the set of finite solutions of E. It is possible to find in time $O(|E|)$ a set $\widehat{S}$ of finite solution candidates of size $|\widehat{S}| = O(\log |E|)$ and such that $S \subseteq \widehat{S}$.*

Proof An immediate conclusion from the proof of Theorem 1 is that for the equations that feature $|A_0| \leq |B_1|$ all the solutions can be detected in time $O(|E|)$, and for the equations that feature $|A_0| > |B_1|$ the solutions not shorter than $|A_0| - |B_1|$ can be detected in time $O(|E|)$. In both cases the respective solution sets contribute to the magnitude of $\widehat{S}$ in only a constant factor. Thus without loss of generality it suffices to focus on the equations that feature $|A_0| > |B_1|$ and search for solutions candidates shorter than $|A_0| - |B_1|$.

Let $|A_0| > |B_1|$. Fix $X \in S$ of length $|X| < |A_0| - |B_1|$. Then $XB_1X \sqsubset A_0X \sqsubset A_0A_0$ and consequently $(B_1X)^2$ is a proper prefix of $B_1A_0A_0$. Let w denote the primitive root of B_1X. Therefore $w^2 \sqsubset B_1A_0A_0$ and our aim is to search for finite solution candidates with respect to the elements of $Rpp(B_1A_0A_0)$. Then $B_1X = w^k$ for some $k \geq 1$. Hence there is a unique pair of words (u, v) such that word u is non-empty and there are numbers i, j such that $w = vu$ and $B_1 = (vu)^i v$ and $X = (uv)^j u$, see Fig. 1. We say that (u, v) is the unique *generator* for w. We put all $u = (uv)^0 u$ into $\widehat{S}$. We prove next that for each $w \in Rpp(B_1A_0A_0)$ we shall put into $\widehat{S}$ at most one word of the form $(uv)^j u$ where (u, v) is the generator for w and $j > 0$ in total time $O(|E|)$.

We start by showing that the set $Rpp(B_1A_0A_0)$ can be computed in time $O(|E|)$. Following Knuth, Morris and Pratt [4] let

$$P[j] = \max\{0 \leq k < j : B_1A_0A_0[1 \ldots k] \sqsupset B_1A_0A_0[1 \ldots j]\}.$$

The *failure table* P can be calculated [4] in time $O(|B_1A_0A_0|)$. The property of P that $j - P[j]$ is the length of the shortest period of $B_1A_0A_0[1 \ldots j]$ allows us to calculate $Rpp(B_1A_0A_0)$ in total time $O(|B_1A_0A_0|)$. More precisely

$$w \in Rpp(B_1A_0A_0) \quad \Longleftrightarrow \quad 2|w| - P[2|w|] = |w|.$$

Fix $w \in Rpp(|B_1A_0A_0|)$ and let (u, v) be the generator for w. By Lemma 1 there is at most one $j > 0$ such that $X = (uv)^j u$ is a solution of E. We show how to eliminate all but one $j > 0$ so that $X = (uv)^j u$ remains the respective solution candidate for E. For this purpose we analyze the sequence of consecutive occurrences of w in both sides of the equation.

First we calculate the number of consecutive occurrences of w in $B_1A_0A_0$. Since $ww \sqsubset B_1A_0A_0$ it follows from the periodicity lemma and primitivity of w that there is at most one w for which the sequence of its consecutive occurrences does not end inside $B_1A_0A_0$ and such w can be found in time $O(|B_1A_0A_0|)$. To find its respective solution candidate we run in time $O(|E|)$ the algorithm described in [10]. Therefore without loss of generality we may assume the sequence ends inside $B_1A_0A_0$. We can use now the failure table P to find for each w the maximum number of its consecutive occurrences in $B_1A_0A_0$ in total time $O(|B_1A_0A_0|)$. Consider a sequence of w starting in the right-hand side of equation E. All such w that their sequence ends inside B_1 can be discarded, since they could never be primitive roots of B_1X. Thus let (u, v) be the unique generator for $w = vu$. Since B_1 must equal $(vu)^i v$ for some $i \geq 0$ we use the failure table for B_1 to determine for all w in total time $O(|B_1|)$ which of the cases takes place.

Case $i \geq 2$. It follows from the periodicity lemma and primitivity of w that there is at most one such w. To find a candidate corresponding to w we run the algorithm described in [10].

Case $i = 1$. There are at most two such w. Indeed, for the set $Rpp(B_1A_0A_0) = \{w_1, \dots, w_k\}$ of repetitive primitive prefixes we have $2|w_i| \leq |w_{i+2}|$. Since $|w| \leq |B_1| < |ww|$ at most two consecutive words w_i, w_{i+1} meet the conditions. Therefore for a constant number of words w we run the algorithm described in [10].

Case $i = 0$. Then $B_1 = v$ and let t be the first index such that $B_t \neq B_1$. If $B_t = (vu)^j v$, $j \geq 1$ then again for a constant number of words w we run the algorithm described in [10]. Otherwise $B_t \neq (vu)^j v$ for any $j \geq 0$ and for each w we find the place its sequence ends inside B_tuv (note that B_tuv is a prefix of B_tX) in total time $O(|B_t|)$ using a failure table for B_tA_0. Suppose that $s = B_tuv/w$ and $r = B_1A_0A_0/w$. Then $(B_1X)^t B_tX/w = B_1A_0A_0/w$ so $jt + t + s = r$ has at most one solution with respect to j. If such a solution exists then we put $X = (uv)^j u$ into $\widehat{S}$. $\square$

As an immediate consequence of Theorem 1 and Theorem 2 we have

Theorem 3 *Let E be a one-variable word equation. Then $Sol(E) = F$ or $Sol(E) = F \cup (uv)^+ u$ for some $u, v \in \Sigma^*$ and a set F of words in Σ^* where $|F| = O(\log |E|)$.*

4 Verification of Solution Candidates

Throughout this section we assume that the alphabet Σ is of constant size or $\Sigma = \{1, \dots, |\Sigma|\}$.

Lemma 2 *Given a finite set Π of words over an alphabet Σ and of total length $n = \sum_{u \in \Pi} |u|$, after an $O(n)$-time preprocessing it is possible to answer in time $O(1)$ if for given a, b being some prefixes of words in Π it is true that $a \sqsubseteq b$.*

Proof The preprocessing phase goes as follows. First we build a *trie* tree to represent all words in set Π. Additionally we keep a link from every letter in every word in Π to its respective node in the tree. Then we traverse the trie tree from the root in prefix order and assign the nodes consecutive numbers. Then we traverse the trie tree in suffix order and assign every node the range of numbers in the subtree rooted at the node, including the node's own number. Thus every node is additionally assigned its subtree's minimum number and maximum number. This can be done in time $O(n)$. We follow now the links from the last letters of a and b to check if the respective nodes are in a child-parent relation in trie tree, that is if one of the nodes is within the range of the other node. This can be done in time $O(1)$, which completes the proof. □

Let Π be a set of words and $\mathcal{T}$ be the trie tree for the set Π. Recall that for a given vertex v in $\mathcal{T}$ its *label* is the word $\text{Label}(v)$ consisting of letters on consecutive edges of the path from the *root* of $\mathcal{T}$ to v. By definition (of a trie tree) the label for a vertex v in $\mathcal{T}$ is a prefix of some word in Π. Lets enrich the trie tree by maintaining an array Vertex of links from prefixes of words in Π to their respective *vertices* of $\mathcal{T}$, so that for any a being a prefix of a word in Π we have $\text{Label}(\text{Vertex}(a)) = a$.

Recall that for a given word u a *prefix table* is the array $\text{Pref}[1\dots|u|]$ such that $\text{Pref}[j]$ is the length of the longest word starting at j in u which is a prefix of u. More precisely $\text{Pref}[j] = \max\{0 < k \le |u| - j + 1 : u[j \dots j + k - 1] \sqsubseteq u\} \cup \{0\}$.

We generalize now the notion of a prefix table to a set of words Π.

Definition 1 For a given set Π of words a *prefix table* is defined as

$$\text{Pref}[u, i] = v$$

where $u \in \Pi$, $1 \le i \le |u|$ and v is a vertex in the trie tree for Π such that $\text{Label}(v)$ is the longest prefix of a word in Π which occurs at position i in u.

Lemma 3 *Given a finite set* Π *of words over an alphabet* Σ *and of total length* $n = \sum_{u\in\Pi} |u|$ *it is possible to construct the prefix table for* Π *in time* $O(n)$.

Proof Let $\mathcal{A}$ be the Aho-Corasick automaton for Π, that is a deterministic finite automaton accepting the set of all words containing a word from Π as a suffix. Recall [4] that the Aho-Corasick automaton is in fact a trie tree for the set of words Π with additional edges $s \to t$ labeled by symbols $b \in \Sigma$ such that $\text{Label}(t)b$ is the longest proper suffix of $\text{Label}(s)$ which is a prefix of a word in Π. If the automaton goes through such an edge we say it *backtracks*. Let $\mathcal{T}$ denote the trie tree being the skeleton of $\mathcal{A}$. Recall [4] that construction of $\mathcal{A}$ takes time proportional to $n|\Sigma|$ and since Σ is of constant size then it is possible to construct $\mathcal{A}$ in time $O(n)$.

Fix a word $u \in \Pi$. We fill the table Pref for u iteratively. Clearly $\text{Pref}[u, 1] = \text{Vertex}(u)$. We run $\mathcal{A}$ on u starting from position $i = 2$ in u. If $\mathcal{A}$ goes down $\mathcal{T}$ then we do nothing. Assume that $\mathcal{A}$ reached symbol at position j in u going down $\mathcal{T}$ and in the next step (reaching symbol at position $j + 1$ in u) backtracked in $\mathcal{T}$ from state v to state w, see Fig. 2. Hence by definition of $\mathcal{A}$ (as recalled above) we have $\text{Pref}[u, i] = v$. Before we resume the iteration from state w and position l in

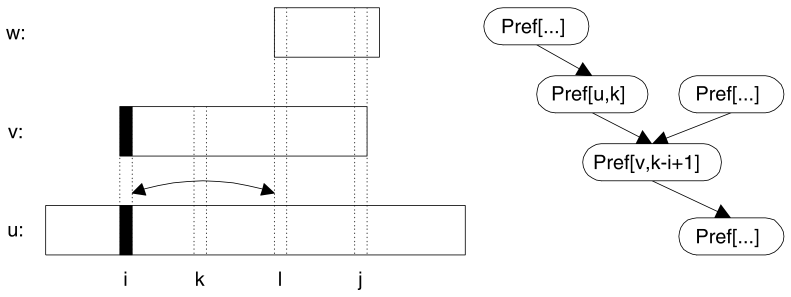

Fig. 2 *Left*: The Aho-Corasick algorithm backtracked at position $j+1$ from v to w, corresponding labels depicted. *Right*: The dependency graph

u we have to fill the missing values for $\mathrm{Pref}[u, i+1 \dots l-1]$ where $l = (j+1) - |\mathrm{Label}(w)| + 1$.

Fix $k \in (i, l)$. Let r be the length of the longest prefix of a word in Π which starts at position k in u, that is

$$r = \min\{|\mathrm{Label}(\mathrm{Pref}[\mathrm{Word}(v), k-i+1])|, j-k+1\}$$

where $\mathrm{Word}(v)$ denotes any word in Π such that $\mathrm{Label}(v)$ is its prefix (i.e. it is the first word that caused vertex v to be inserted into $\mathcal{T}$ while building $\mathcal{A}$). Then

$$\mathrm{Pref}[u, k] = \mathrm{Vertex}(\mathrm{Label}(\mathrm{Pref}[\mathrm{Word}(v), k-i+1])[1 \dots r]). \tag{1}$$

Since naive recurrent computation of missing values of the array Pref could yield multiple calculation of certain prefix table values, we resolve the problem dynamically. We say that $\mathrm{Pref}[u, k]$ *depends* on $\mathrm{Pref}[\mathrm{Word}(v), k-i+1]$ and build a *dependency* graph, see Fig. 2. The dependency graph is in fact a forest of trees since the outdegree of each node in the graph is at most one and there are no cycles in the graph. Indeed, every edge leads to a vertex representing a value of Pref with a smaller label. Using the dependency graph we compute the values in the trees in a root-to-leaves order using (1). It is possible since every tree must be rooted at a node that has already been calculated. Therefore the total computation takes time $O(n)$. □

Observe that having computed the prefix table for Π it is possible to calculate in constant time $\mathrm{Pref}[u, i]$ for any u being a prefix of a word in Π and $1 \le i \le |u|$. More precisely if u is a prefix of a word $w \in \Pi$ then $\mathrm{Pref}[u, i] = \mathrm{Vertex}(\mathrm{Label}(\mathrm{Pref}[w, i])[1 \dots r])$ where $r = \min\{|\mathrm{Label}(\mathrm{Pref}[w, i])|, |u| - i + 1\}$.

Theorem 4 *Given an equation $E : L = R$ over an alphabet Σ and in one variable $x \notin \Sigma$, after an $O(|E|)$-time preprocessing it is possible to answer in time $O(\#_x)$ if a given candidate $\widehat{X} \sqsubseteq A_0$ is a solution of E.*

Proof Let $\Pi = \{A_0, \dots, A_s, B_1, \dots, B_s\}$, that is Π be the set of equation *constants*. First we follow the proof of Lemma 2 and build a trie tree to represent the words in Π. Next we build the prefix table for Π. Since $\widehat{X} \sqsubset A_0$, $\widehat{X}$ is represented both in the trie

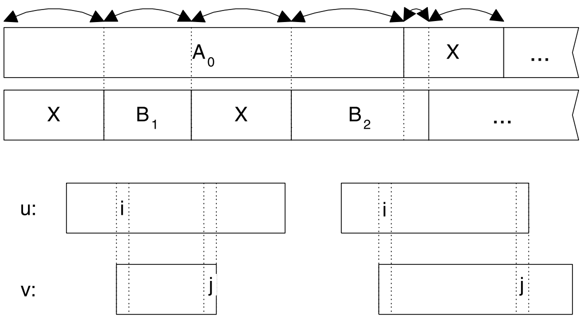

Fig. 3 *Top*: Slices correspond to arches. *Bottom*: Two possible situations where a slice occurs between positions i in u and j in v

tree and the prefix table. It follows from Lemmas 2 and 3 that the preprocessing can be done in time $O(|E|)$. Let the homomorphism φ be the canonical extension to domain $\Sigma \cup \{x\}$ of a mapping $x \to \widehat{X}$. We slice both $\varphi(L)$ and $\varphi(R)$ at the positions where any of the words $\Pi \cup \{\widehat{X}\}$ begin/end, see Fig. 3. Thus we get $\Theta(\#_x)$ pairs of slices. We shall compare the respective slices iteratively from left to right. In any case if we get a mismatch, we end with a "not a solution" answer. It suffices to prove a single comparison takes $O(1)$ steps. It is easy to note that the equation constants the slices origin from can be in at most two distinct configurations, see Fig. 3. Let u be the word that does not start in a given slice or any of the words otherwise. Let v be the other word. We use the prefix table for Π and find $\text{Pref}[u,i] = w$. It follows from Lemma 3 that it can be done in time $O(1)$. Now we check if $v[1\dots j] \sqsubseteq \text{Label}(w)[1\dots k]$. It follows from Lemma 2 that it can be done in time $O(1)$. If the comparison results true, then we cancel the slice and iterate the algorithm. Since every iteration takes time $O(1)$ and the number of iterations equals $O(\#_x)$, verification time for one solution candidate is $O(\#_x)$. This completes the proof. □

As an immediate consequence of Theorems 1, 2 and 4 we obtain

Theorem 5 *Let Σ be of constant size or $\Sigma = \{1, \dots, |\Sigma|\}$. There is an algorithm finding all solutions of any word equation E over Σ with one variable $x \notin \Sigma$ in time*

$$O(|E| + \#_x \log |E|).$$

References

1. Angluin, D.: Finding pattern common to a set of string. In: Proceedings of Symposium on the Theory of Computing, STOC'79, pp. 130–141. ACM, New York (1979)
2. Charatonik, W., Pacholski, L.: Word equations in two variables. In: Proceedings of the International Workshop on Word Equations and Related Topics, IWWERT'91. Lecture Notes in Computer Science, vol. 677, pp. 43–57. Springer, Berlin (1991)
3. Crochemore, M., Rytter, W.: Periodic prefixes in texts. In: Capocelli, R., De Santis, A., Vaccaro, U. (eds) Sequences II, Positano, 1991, pp. 153–165. Springer, New York (1993)
4. Crochemore, M., Rytter, W.: Text Algorithms. Oxford University Press, London (1994)
5. Gutierrez, C.: Satisfiability of word equations with constants is in exponential space. In: Proceedings of the Annual Symposium on Foundations of Computer Science, FOCS'98, pp. 112–119. IEEE Comput. Soc. Press, Los Alamitos (1998)

6. Ilie, L., Plandowski, W.: Two-variable word equations. Theor. Inform. Appl. **34**, 467–501 (2000)
7. Kościelski, A., Pacholski, L.: Complexity of Makanin's algorithm. J. Assoc. Comput. Math. **43**(4), 670–684 (1996)
8. Makanin, G.S.: The problem of solvability of equations in a free semigroup. Mat. Sb. **103**(2), 147–236 (1977) (in Russian). English translation in: Math. USSR Sb. **32**, 129–198 (1977)
9. Neraud, J.: Equations in words: an algorithmic contribution. Bull. Belg. Math. Soc. **1**, 253–283 (1994)
10. Eyono Obono, S., Goralcik, P., Maksimenko, M.N.: Efficient solving of the word equations in one variable. In: Mathematical Foundations of Computer Science 1994, 19th International Symposium, MFCS'94. Lecture Notes in Computer Science, vol. 841, pp. 336–341. Springer, Berlin (1994)
11. Plandowski, W.: Satisfiability of word equations is in NEXPTIME. In: Proceedings of the Symposium on the Theory of Computing, STOC'99, pp. 721–725. ACM, New York (1999)
12. Plandowski, W.: Satisfiability of word equations is in PSPACE. In: Proceedings of the Annual Symposium on Foundations of Computer Science, FOCS'99, pp. 495–500. IEEE Comput. Soc., Los Alamitos (1999)
13. Plandowski, W., Rytter, W.: Application of Lempel-Ziv encodings to the solution of word equations. In: Proceedings of the International Colloquium on Automata, Languages and Programming, ICALP'98. Lecture Notes in Computer Science, vol. 1443, pp. 731–742. Springer, Berlin (1998)