Fundamenta Informaticae 126 (2013) 1–34
IOS Press

On language equations with one-sided concatenation

Franz Baader[*]

Alexander Okhotin[†]

Abstract. Language equations are equations where both the constants occurring in the equations and the solutions are formal languages. They have first been introduced in formal language theory, but are now also considered in other areas of computer science. In the present paper, we restrict the attention to language equations with one-sided concatenation, but in contrast to previous work on these equations, we allow not just union but all Boolean operations to be used when formulating them. In addition, we are not just interested in deciding solvability of such equations, but also in deciding other properties of the set of solutions, like its cardinality (finite, infinite, uncountable) and whether it contains least/greatest solutions. We show that all these decision problems are ExpTime-complete.

1. Introduction

Equations with formal languages as constant parameters and unknowns have been studied since the 1960s, when two basic concepts of the theory of computation, finite automata and context-free grammars, were respectively represented as systems of equations with union and one-sided concatenation [8] and with union and unrestricted concatenation [14]. This topic was further studied in the monographs on algebraic automata theory by Salomaa [34] and Conway [12]. There has been a renewed interest in the topic over the last two decades, with the state-of-the-art as of 2007 presented in a survey by Kunc [19], and with more research on various aspects of language equations appearing in the last few years [13, 16, 17, 20, 22, 30].

As an example, consider the equation $X = AX \cup B$, where A, B are fixed formal languages. It is well-known, that this equation has A^*B as a solution. If the empty word does not belong to A, then this is the only solution. Otherwise, A^*B is the least solution (w.r.t. inclusion), and all solutions are of the

[*]Institute for Theoretical Computer Science, Technical University of Dresden, Dresden D–01062, Germany. E-mail: `baader@tcs.inf.tu-dresden.de`

[†]Department of Mathematics and Statistics, University of Turku, Turku FI–20014, Finland. Supported by the Academy of Finland under grants 134860 and 257857. E-mail: `alexander.okhotin@utu.fi`

form C^*B for $C \supseteq A$. Depending on A and the available alphabet, the equation may thus have finitely many, countably infinitely many, or even uncountably many solutions. The above equation is an equation with one-sided concatenation since concatenation occurs only on one side of the variable. In contrast, the equation $X = aXb \cup XX \cup \varepsilon$ is not one-sided.[1] Its least solution is the Dyck language of balanced parentheses generated by the context-free grammar $S \to aSb \mid SS \mid \varepsilon$, whereas its greatest solution is $\{a, b\}^*$.

Both examples are *resolved* equations in the sense that their left-hand sides consist of a single variable. If only monotonic operations (in the examples: union and concatenation) are used, then such resolved systems of equations $X_i = \varphi_i(X_1, \ldots, X_n)$ with $i = 1, \ldots, n$ always have a least and a greatest solution due to the Tarski–Knaster fixpoint theorem [37]. Once the resolved form of equations is no longer required or non-monotonic operations (like complementation) are used, a given language equation need no longer have solutions, and thus the problem of deciding solvability of such an equation becomes non-trivial. The same is true for other decision problems, like asking for the existence of a least/greatest solution or determining the cardinality of the set of solutions. In some cases, the basic properties of such equations can be effectively tested: such are, for instance, equations of the form $\varphi(X_1, \ldots, X_n) = const$, studied by Bala [7], where the solution existence problem is EXPSPACE-hard, while the special case $XY = const$ was proved to be PSPACE-complete by Martens et al. [22]. Another example is given by resolved systems of equations with concatenation and complementation, investigated by Okhotin and Yakimova [31], which have NP-complete solvability testing.

In the case of language equations of the general form $\varphi(X_1, \ldots, X_n) = \phi(X_1, \ldots, X_n)$ with the operations of union and unrestricted concatenation, the solvability problem becomes undecidable since the equivalence problem for context-free grammars can easily be encoded [32]. A systematic study of the hardness of decision problems for such language equations was carried out by Okhotin [26, 27, 28, 29], who also characterized recursive and recursively enumerable sets by solutions of these equations. Recently these computational completeness results were extended to language equations over a one-letter alphabet by Jeż and Okhotin [16].

A surprising proof of the computational universality of very simple language equations of the form $LX = XL$, where $L \subseteq \{a, b\}^*$ is a finite constant language, was given by Kunc [18]. Later, Jeż and Okhotin [17] and Lehtinen and Okhotin [20] demonstrated that already systems of two equations $\{XXK = XXL,\ XM = N\}$, with regular constants $K, L, M, N \subseteq a^*$, possess a full range of undecidable problems, and can represent an encoding of any recursive (r.e., co-r.e.) set in their unique (least, greatest) solutions. Though language equations with concatenation as the only operation are syntactically close to word equations [21], like the equation $aX = Xa$, there is no strong relationship between the two types of equations since the unknowns stand for different mathematical objects: a single word in the case of word equations versus a set of words in the case of language equations.

Language equations with one-sided concatenation usually do not have undecidable decision problems. In fact, many properties of the solution sets of such equations, such as existence and uniqueness of their solutions, can be expressed in Rabin's monadic second-order logic on infinite trees [33]. This implies the decidability of these problems, but only yields a non-elementary complexity upper-bound [36]. Language equations with one-sided concatenation can also be regarded as a particular case of equations on sets of terms, known as *set constraints*, which received significant attention [1, 10, 11, 15] since they can, e.g., be used in program analysis. In fact, language equations with one-sided concatenation corre-

[1] As usual, we omit set parentheses for singleton languages.

spond to monadic set constraints, where all function symbols are unary. Thus, decidability results for set constraints also yield decidability results for the corresponding language equations. However, since set constraints are in general more complex than monadic set constraints, this does not necessarily yield optimal complexity bounds.

Language equations with one-sided concatenation *and union* have been studied in the context of unification problems in description logics: Baader and Narendran [3] show that the existence of a finite solution (i.e., a solution where all unknowns are replaced by finite languages) is an ExpTime-complete problem; Baader and Küsters [2] show the same for the existence of an arbitrary (possibly infinite) solution. In the latter work, it is also shown that a solvable equation always has a greatest solution, and that this solution is regular (i.e., consists of regular languages).

The present paper extends the aforementioned results in two directions. On the one hand, we consider language equations with one-sided concatenation and *all Boolean operations*, and on the other hand we consider *additional decision problems*, like determining the existence of least/greatest solutions and the cardinality of the solution set. All these problems turn out to be ExpTime-complete for language equations with one-sided concatenation and any set of available Boolean operations between $\{\cup\}$ and $\{\cup, \cap, \neg\}$.

After a preliminary section in which we give the relevant definitions, we first concentrate in Section 3 on showing the ExpTime upper-bounds for the mentioned decision problems in the case of the most general type of one-sided equations where all Boolean operations are available. This is done by translating language equations into a special kind of looping tree automata, showing a 1–1-relationship between the solutions of the equation and the runs of the corresponding automaton, and then characterizing the relevant properties of solution sets by decidable properties of the automaton. Thus, we have a uniform approach for solving all decision problems by one automaton construction. The decision procedures for the respective problems only differ in what property of the constructed automaton must be decided. Furthermore, this construction implies an easy proof of the regularity of unique, least and greatest solutions of such language equations, as well as an effective construction of finite automata for these solutions.

In Section 4, we then show the ExpTime lower-bounds for the mentioned decision problems in the case of one-sided language equations with union: the reduction is from the intersection emptiness problem for deterministic looping tree automata, whose ExpTime-completeness easily follows from the ExpTime-completeness of the same problem for deterministic top-down tree automata on finite trees [35, 2]. Again, the hardness proofs are uniform: one reduction shows hardness of all decision problems under consideration.

2. Preliminaries

In this section, we first introduce the language equations investigated in this paper, and show that they can be transformed into a simpler normal form. Then, we introduce some notions regarding automata working on infinite trees, which will be important for showing both the upper and the lower complexity bounds.

2.1. Language equations with one-sided concatenation

For a fixed finite alphabet Σ, we consider systems of equations of the following general form:

$$\begin{cases} \psi_1(X_1,\ldots,X_n) & = & \xi_1(X_1,\ldots,X_n), \\ & \vdots & \\ \psi_m(X_1,\ldots,X_n) & = & \xi_m(X_1,\ldots,X_n), \end{cases} \tag{1}$$

where the form of the expressions ψ_i and ξ_i is defined inductively:

- every variable X_i is an expression;
- every regular language $L \subseteq \Sigma^*$ is an expression;
- a concatenation $L\varphi$ of a regular constant language $L \subseteq \Sigma^*$ with an expression φ is an expression;
- if φ, φ' are expressions, then so are $(\varphi \cup \varphi')$, $(\varphi \cap \varphi')$ and $(\sim\varphi)$.

We assume that the regular languages in expressions are given by non-deterministic finite automata. An effective description of a system (1) would contain transition tables and accepting states of these automata, and thus the number of their states and transitions adds to the size of the description.

If the expressions in such a system contain neither intersection nor complement, then we call it a system of language equations with one-sided concatenation and *union*.

The above definition allows concatenation of a constant regular language on the left with an arbitrary expression on the right. One could symmetrically restrict the concatenation to be only of the form φL, where φ is any expression and L is a constant , and the resulting equations will operate isomorphically, with all languages replaced by their mirror images. Thus, there is no need to distinguish between equations with left concatenation and equations with right concatenation: they are handled by the same methods and algorithms, and shall be referred as *equations with one-sided concatenation*, without concentrating on the direction of concatenation.

A *solution* of a general system (1) is a vector of languages $(L_1,\ldots,L_n)$ such that a substitution of L_j for X_j for all j turns each instantiated equation into an equality. Solutions can be compared w.r.t. inclusion of their components: we define $(L_1,\ldots,L_n) \preceq (L'_1,\ldots,L'_n)$ if and only if $L_i \subseteq L'_i$ holds for $i = 1,\ldots,n$. In addition to the problem of deciding whether a system has a solution or not, we consider additional decision problems that look more closely at properties of the set of solutions: its cardinality (is there a unique solution, are there finitely or infinitely many solutions, are there countably or uncountably many solutions) and whether it contains least/greatest elements w.r.t. $\preceq$.

In order to design algorithms for solving these decision problems, it is more convenient to consider language equations in the following normal form: a single equation

$$\varphi(Z_1,\ldots,Z_k) = \varnothing, \tag{2}$$

in the unknowns $Z_1,\ldots,Z_k$, where the constant regular languages occurring in φ are singleton languages $\{\varepsilon\}$ and $\{a\}$ for $a \in \Sigma$, which we simply write as ε and a.

The next lemma implies that w.r.t. all decision problems concerned with the cardinality of the set of solutions (including the existence of a solution), the restriction to equations of form (2) is without loss of generality.

Lemma 1. For every system (1) in the unknowns $X_1, \dots, X_n$ we can construct in polynomial time an equation (2) in the unknowns $X_1, \dots, X_n, Y_1, \dots, Y_\ell$ for some $\ell \geqslant 0$ such that the set of solutions of (2) is

$$\big\{\big(L_1, \dots, L_n, \eta_1(L_1, \dots, L_n), \dots, \eta_\ell(L_1, \dots, L_n)\big) \bigm| (L_1, \dots, L_n) \text{ solves (1)}\big\}$$

for some functions $\eta_1, \dots \eta_\ell : (2^{\Sigma^*})^n \to 2^{\Sigma^*}$.

Proof:
[Proof sketch:] Regular languages in (1) can be expressed by employing resolved equations for additional variables $Y_1, \dots, Y_\ell$. For example, the expression $(\sim X)a^*b$ can be replaced by Y_2 if we add the resolved equations $Y_2 = Y_1 b$ and $Y_1 = Y_1 a \cup \sim X$. Since resolved equations of this form have a unique solution, any value for X yields unique values for Y_1, Y_2. The total size of equations added is proportional to the number of transitions in an NFA, and hence the growth is linear.

Every equation $\psi_i = \xi_i$ has the same solutions as $(\psi_i \cap \sim\xi_i) \cup (\xi \cap \sim\psi_i) = \varnothing$, and the system $\varphi_1 = \varnothing, \varphi_2 = \varnothing$ has the same solutions as $\varphi_1 \cup \varphi_2 = \varnothing$. □

Regarding the existence of least/greatest solutions, we must be more careful. For example, when representing $(\sim X)a^*b$ by Y_2 and the equations $Y_2 = Y_1 b, Y_1 = Y_1 a \cup \sim X$, a larger value for X yields smaller values for Y_1, Y_2. Thus, even if the original system has a least/greatest solution, the new one need not have one. The solution to this problem will be that when defining the relation $\preceq$ on solutions, we do not necessarily compare solutions w.r.t. all components, but only w.r.t. the components corresponding to a set of *focus variables*. In this case, the constructed system (2) with unknowns $X_1, \dots, X_n, Y_1, \dots, Y_\ell$ has a least/greatest solution w.r.t. the focus variables $X_1, \dots, X_n$ if and only if the original system (1) has a least/greatest solution. Note that $\preceq$ is then no longer a partial order, but only a preorder. Accordingly, a system of language equations may have multiple least (greatest) solutions with respect to $\preceq$, yet all of them have to coincide on the focus variables.

2.2. Automata on infinite trees

Given a ranked alphabet Γ, where every symbol has a nonzero rank, infinite trees over Γ are defined in the usual way, that is, every node in the tree is labeled with an element $f \in \Gamma$ and has as many successor nodes as is the rank of f. A *looping tree automaton*[2] $\mathcal{A} = (Q, \Gamma, Q_0, \Delta)$ consists of a finite set of states Q, a ranked alphabet Γ, a set of initial states $Q_0 \subseteq Q$, and a transition function $\Delta : Q \times \Gamma \to 2^{Q^*}$ that maps each pair (q, f) to a subset of Q^k, where k is the rank of f. This automaton is *deterministic* if $|Q_0| = 1$ and $|\Delta(q, f)| \leqslant 1$ for all pairs (q, f). A *run* r of $\mathcal{A}$ on a tree t labels the nodes of t with elements of Q, such that the root is labeled with $q_0 \in Q_0$, and the labels respect the transition function, that is, if a node v has label $t(v)$ in t and label $r(v)$ in r, then the tuple $(q_1, \dots, q_k)$ labeling the successors of v in r must belong to $\Delta(q, t(v))$. The tree t is *accepted* by $\mathcal{A}$ if there is a run of $\mathcal{A}$ on t. The *language accepted by* $\mathcal{A}$ is defined as

$$L(\mathcal{A}) := \{t \mid t \text{ is an infinite tree over } \Gamma \text{ that is accepted by } \mathcal{A}\}.$$

[2]The difference between looping tree automata and Büchi tree automata [38] is that there is no acceptance condition involving final states.

It is well-known that the emptiness problem for looping tree automata, that is, the question whether the accepted language is non-empty, is decidable in linear time [6]. However, the intersection emptiness problem, that is, given looping tree automata $\mathcal{A}_1, \dots, \mathcal{A}_k$, is $L(\mathcal{A}_1) \cap \dots \cap L(\mathcal{A}_k)$ empty or not, is EXPTIME-complete even for deterministic automata [35, 2]. This result will be used to show the complexity lower-bounds in Section 4.

When showing the complexity upper-bounds in Section 3, we actually employ a very restricted form of looping automata. First, we restrict the attention to a ranked alphabet Γ containing a single symbol γ of some fixed rank $k > 0$. Thus, there is only one infinite tree, and the labeling of its nodes by γ can be ignored. Given an arbitrary finite alphabet $\Sigma := \{a_1, \dots, a_k\}$ of cardinality k, every node in this tree can uniquely be represented by a word $w \in \Sigma^*$, where a_i corresponds to the ith successor. Second, we consider not arbitrary looping tree automata working on this tree, but tree automata induced by word automata. A non-deterministic finite automaton (NFA) $A = (Q, \Sigma, Q_0, \delta)$ without accepting states working on words over Σ induces a looping tree automaton $\mathcal{A} = (Q, \Gamma, Q_0, \Delta)$ working on the infinite tree over Γ as follows:

$$\Delta(q, \gamma) := \{(q_1, \dots, q_k) \mid q_i \in \delta(q, a_i) \text{ for } i = 1, \dots, k\}.$$

We call such an automaton a *looping tree automaton with independent transitions (ILTA)* since in every component the successor states can be chosen independently from what is chosen in the rest of the components. In the following, we do not distinguish between the NFA and the ILTA it represents. For example, we will talk about runs of the NFA, but mean the runs of the corresponding ILTA. The runs of the NFA $A = (Q, \Sigma, Q_0, \delta)$ can thus be represented as functions $r : \Sigma^* \to Q$ such that $r(\varepsilon) \in Q_0$ and $r(wa) \in \delta(r(w), a)$ for all $w \in \Sigma^*$ and $a \in \Sigma$. In addition, when defining an ILTA, we will usually introduce just the corresponding NFA, and call it ILTA. In the next section, we are not interested in the tree language accepted by an ILTA (which is either empty or a singleton set); instead, we are interested in the runs themselves.

Following the definition of looping tree automata, an ILTA is called *deterministic* if $|\delta(q, a)| \leqslant 1$ for all $q \in Q$ and $a \in \Sigma$, that is, if the underlying NFA is a partial DFA. Note that a deterministic ILTA has at most one run; furthermore, having a completely defined function δ is a sufficient condition of having exactly one run.

We call an NFA $A = (Q, \Sigma, Q_0, \delta)$ and the ILTA it represents *trim* if every state is reachable from an initial state, and $\delta(q, a) \neq \varnothing$ for all $q \in Q$ and $a \in \Sigma$. It is easy to see that every NFA can be transformed into an trim NFA that is *equivalent* in the sense of having the same runs. In such a trim NFA, every finite or infinite path can be completed to a run containing it. In addition, it has a run if and only if Q is non-empty.

Lemma 2. For every ILTA $A = (\Sigma, Q, Q_0, \delta)$ an equivalent trim ILTA $B = (\Sigma, Q', Q'_0, \delta')$ can be constructed in polynomial time.

Proof:
The construction proceeds in two steps. First, we construct the set

$$Q_{\text{defin}} := \{q \in Q \mid \forall w \in \Sigma^*.\ \delta(q, w) \neq \varnothing\}.$$

The complement of this set can be computed in polynomial time by the following iteration:

$$\begin{aligned} Q^{(0)} &:= \{q \in Q \mid \exists a \in \Sigma.\ \delta(q,a) = \varnothing\}, \\ Q^{(i+1)} &:= Q^{(i)} \cup \{q \in Q \mid \exists a \in \Sigma.\ \delta(q,a) \subseteq Q^{(i)}\}. \end{aligned}$$

Since Q is finite, there is an $n \leqslant |Q|$, such that $Q^{(n)} = Q^{(n+1)} = \bigcup_{i \geq 0} Q^{(i)}$, and it is easy to show that $Q_{\text{defin}} = Q \setminus Q^{(n)}$.

Let $A' := (\Sigma, Q_{\text{defin}}, Q_0 \cap Q_{\text{defin}}, \delta')$ be the ILTA obtained by restricting A to the set of states Q_{defin}, that is, $\delta'(q,a) := \delta(q,a) \cap Q_{\text{defin}}$ for all $q \in Q_{\text{defin}}, a \in \Sigma$. It is easy to show that A' satisfies the second condition in the definition of trim, that is, $\delta'(q,a) \neq \varnothing$ holds for all $q \in Q_{\text{defin}}, a \in \Sigma$. In fact, assume that $\delta'(q,a) = \varnothing$ for some $q \in Q_{\text{defin}}$. Then $\delta(q,a) \subseteq \overline{Q}_{\text{defin}} = Q^{(n)}$, which implies $q \in Q^{(n+1)} = Q^{(n)} = \overline{Q}_{\text{defin}}$, contradicting our assumption that $q \in Q_{\text{defin}}$. In addition, A' has the same set of runs as A, since it is easy to see that no state in $\overline{Q}_{\text{defin}}$ can occur in a run: if $q = r(u)$ for a run r of A, then $r(uw) \in \delta(q,w)$ for all words w, and thus $q \in Q_{\text{defin}}$.

Second, we construct the set

$$Q_{\text{reach}} := \{q \in Q_{\text{defin}} \mid \exists q_0 \in Q_0 \cap Q_{\text{defin}}. \exists w \in \Sigma^*.\ q \in \delta'(q_0, w)\}.$$

This set can obviously be computed by a simple polynomial-time search in the graph corresponding to the automaton A': test whether q is reachable from some initial state q_0. Now, define $B := (\Sigma, Q_{\text{reach}}, Q_0 \cap Q_{\text{reach}}, \delta'')$ where $\delta''(q,a) = \delta'(q,a) \cap Q_{\text{reach}}$ for all $q \in Q_{\text{reach}}, a \in \Sigma$.

It is easy to see that B is trim. In fact, by the definition of Q_{reach}, every state of B is reachable from some initial state. In addition, since $q \in Q_{\text{reach}}$ implies $q' \in Q_{\text{reach}}$ for all states $q' \in \delta'(q,a)$, the second condition in the definition of trim remains satisfied. Finally, B has the same set of runs as A' since any state in a run r of A' is reachable from the initial state $r(\varepsilon)$. □

3. The complexity upper-bounds

In this section we show that all the decision problems for language equations with one-sided concatenation introduced above can be solved within deterministic exponential time. To this purpose, we show how to translate a given language equation in normal form, $\varphi = \varnothing$, into an ILTA such that there is a one-to-one correspondence between the solutions of the equation and the runs of the corresponding ILTA. The states of this ILTA are sets of subexpressions of φ.

3.1. Translating language equations into ILTA

Let $\Sigma = \{a_1, \ldots, a_m\}$ be an alphabet and let $X_1, \ldots, X_n$ be a set of variables, and consider expressions of the following form:

- ε is an expression;
- every variable X_i is an expression;
- if ψ is an expression and $a \in \Sigma$, then ψa is an expresion;
- if ψ and ξ are expressions, then so are $(\psi \cup \xi)$, $(\psi \cap \xi)$ and $(\sim \psi)$.

An arbitrary expression with one-sided concatenation can be transformed to this form by Lemma 1.

Let $\varphi(X_1, \ldots, X_n)$ be an expression of this form. In the following, we assume that φ is fixed, and denote the set of its subexpressions by Φ. We assume that $\varepsilon, X_1, \ldots, X_n \in \Phi$ (otherwise, we simply add them). Let $\Phi_0 = \{\psi a \mid \psi a \in \Phi\} \cup \{\varepsilon\}$ denote the subset of subexpressions from Φ with concatenation as the top operation, augmented by the constant ε; let the set $\Phi_1 = \Phi_0 \cup \{X_1, \ldots, X_n\}$ include the variables as well. We define two elementary operations on subsets of Φ. The first of them, $select$, maps a set $q_0 \subseteq \Phi_0$ to a finite collection of subsets of Φ_1:

$$select(q_0) = \{q \subseteq \Phi_1 \mid q \setminus \{X_1, \ldots, X_n\} = q_0\}.$$

Note that $|select(q_0)| = 2^n$, and the elements of $select(q_0)$ correspond to different choices of a set of variables.

The other operation, $closure$, completes a subset $q \subseteq \Phi_1$ by computing all applicable Boolean operations over these subexpressions. In order to define the set $closure(q) \subseteq \Phi$, we specify for every expression $\xi \in \Phi$ whether $\xi \in closure(q)$ or not by induction on the structure of ξ:

Base case: For each $\xi \in \{\varepsilon, X_1, \ldots, X_n\}$, let $\xi \in closure(q)$ if and only if $\xi \in q$.

Induction step: Consider $\xi \in \Phi \setminus \{\varepsilon, X_1, \ldots, X_n\}$ and assume that the membership of all proper subexpressions of ξ in $closure(q)$ has already been defined. There are four cases depending on the top operation of ξ:

- If ξ is of the form ψc, then $\xi \in closure(q)$ if and only if $\xi \in q$.
- If $\xi = \psi \cup \eta$, then $\xi \in closure(q)$ if and only if $\{\psi, \eta\} \cap closure(q) \neq \varnothing$.
- If $\xi = \psi \cap \eta$, then $\xi \in closure(q)$ if and only if $\{\psi, \eta\} \subseteq closure(q)$.
- If $\xi = {\sim}\psi$, then $\xi \in closure(q)$ if and only if ψ is not in $closure(q)$.

This operator has the following effect: if we take the set of all subexpressions ξ in Φ_1 that *produce* the word w if applied to the vector L, i.e., that satisfy $w \in \xi(L)$, and apply the operator $closure$ to it, then we obtain the set of all subexpressions in Φ that produce w if applied to L. To be more precise:

Lemma 3. Let $L = (L_1, \ldots, L_n)$ be a vector of languages and $w \in \Sigma^*$. Then

$$closure(\{\xi \in \Phi_1 \mid w \in \xi(L)\}) = \{\xi \in \Phi \mid w \in \xi(L)\}.$$

Proof:
Let $q := \{\xi \in \Phi_1 \mid w \in \xi(L)\}$. We prove that any subexpression $\xi \in \Phi$ is in $closure(q)$ if and only if $w \in \xi(L)$, using induction on the structure of ξ.

Base case. If $\xi \in \Phi_1$, then, by the definition of closure, $\xi \in closure(q)$ if and only if $\xi \in q$. The latter, according to the definition of q, holds if and only if $w \in \xi(L)$.

Induction step. Let $\xi = \psi \cup \eta$. By the definition of closure, $\psi \cup \eta \in closure(q)$ if and only if $\psi \in closure(q)$ or $\eta \in closure(q)$. By the induction hypothesis, $\psi \in closure(q)$ if and only if $w \in \psi(L)$, and $\eta \in closure(q)$ if and only if $w \in \eta(L)$. Therefore, $\psi \cup \eta \in closure(q)$ if and only if $w \in \psi(L)$ or $w \in \eta(L)$, which is equivalent to $w \in \psi(L) \cup \eta(L) = (\psi \cup \eta)(L)$. The proof for intersection and complement is analogous. □

Definition 1. The *ILTA* $A = (\Sigma, Q, Q_0, \delta)$ *induced by the expression* φ is defined as

- $Q := 2^{\Phi}$,
- $Q_0 := \{closure(q) \mid q \in select(\{\varepsilon\})\}$, and
- $\delta(q, a) := \{closure(q') \mid q' \in select(\{\psi a \in \Phi \mid \psi \in q\})\}$.

Note that $|Q_0| = 2^n$ and $|\delta(q, a)| = 2^n$ for all $q \in Q$ and $a \in \Sigma$. Intuitively, the non-determinism is used to "guess" the values of the variables.

There exists a one-to-one correspondence between the runs of A and n-tuples of languages over Σ. First, we show how to associate a run with every vector of languages. The run $r_L : \Sigma^* \to Q$ corresponding to $L = (L_1, \dots, L_n)$ is defined inductively as:

$$r_L(\varepsilon) = closure(\{\varepsilon\} \cup \{X_i \mid \varepsilon \in L_i\}), \tag{3a}$$

$$r_L(wa) = closure(\{\psi a \in \Phi \mid \psi \in r_L(w)\} \cup \{X_i \mid wa \in L_i\}). \tag{3b}$$

It is easy to see that r_L is indeed a run of A.

Conversely, a given run $r : \Sigma^* \to Q$ induces the vector of languages $L^r := (L_1^r, \dots, L_n^r)$, where $L_i^r := \{w \mid X_i \in r(w)\}$.

Lemma 4. The mapping of runs to vectors of languages introduced above is a bijection, and the mapping of vectors of languages to runs is its inverse.

Proof:
First, we prove that going from a vector $L = (L_1, \dots, L_n)$ to the corresponding run, and then back to the corresponding vector is the identity, that is, yields L. Let $L^{r_L} = (L_1', \dots, L_n')$ be the vector of languages corresponding to r_L. Then we have

$$L_i' = \{w \mid X_i \in r_L(w)\} = \{w \mid X_i \in \{X_j \mid w \in L_j\}\} = L_i.$$

The first identity holds by the definition of r_L and the fact that $closure$ does not alter the membership of unknowns X_j. This proves that $L = L^{r_L}$. In particular, this implies that the mapping from runs to vectors is surjective. To complete the proof, it is enough to show that this mapping is also injective.

We show that different runs correspond to different vectors. If $r \neq r'$, this means that $r(w) \neq r'(w)$ for some $w \in \Sigma^*$. Let w be one of the shortest of such strings. Let L and L' be the vectors corresponding to r and r', respectively. If $w = \varepsilon$, then, by (3a), $\{X_i \mid \varepsilon \in L_i\} \neq \{X_i \mid \varepsilon \in L_i'\}$, and so there exists an index i such that $L_i \neq L_i'$. If $w = ua$ for some $u \in \Sigma^*$ and $a \in \Sigma$, then, by (3b), $\{\psi a \in \Phi \mid \psi \in r(u)\} \cup \{X_i \mid ua \in L_i\} \neq \{\psi a \in \Phi \mid \psi \in r'(u)\} \cup \{X_i \mid ua \in L_i'\}$. Since $r(u) = r'(u)$ (which holds because r and r' coincide on all strings shorter than $w = ua$), the first parts are equal, and therefore $\{X_i \mid ua \in L_i\} \neq \{X_i \mid ua \in L_i'\}$, which, as in the previous case, implies that there is an i such that $L_i \neq L_i'$. □

Next, we prove that, for each run r_L, the set of subexpressions in a state $r_L(w)$ (for each string $w \in \Sigma^*$) contains exactly those subexpressions that produce this string when replacing $X_1, \dots, X_n$ by $L_1, \dots, L_n$:

Lemma 5. Let $L = (L_1, \dots, L_n)$ be a vector of languages and r_L be the corresponding run. Then, for every $w \in \Sigma^*$ and $\xi \in \Phi$, we have $w \in \xi(L)$ if and only if $\xi \in r_L(w)$.

Proof:
Induction on the length of w.

Base case: $w = \varepsilon$. According to (3a), it has to be proved that

$$closure\big(\{\varepsilon\} \cup \{X_i \mid \varepsilon \in L_i\}\big) = \{\xi \in \Phi \mid \varepsilon \in \xi(L)\}. \tag{4}$$

It is easy to see that

$$\{\varepsilon\} \cup \{X_i \mid \varepsilon \in L_i\} = \{\xi \in \Phi_1 \mid \varepsilon \in \xi(L)\}. \tag{5}$$

Indeed, looking at the right-hand side of (5), $\varepsilon \in \varepsilon(L)$ by definition, clearly $\varepsilon \notin \psi c(L)$ for all ψ and c, and as for the variables X_i, their membership in both sides is defined identically. By Lemma 3, (5) implies (4).

Induction step: $w = ua$ *for* $a \in \Sigma$. According to (3b) we must prove

$$closure\big(\{\psi a \in \Phi \mid \psi \in r_L(u)\} \cup \{X_i \mid ua \in L_i\}\big) = \{\xi \in \Phi \mid ua \in \xi(L)\}. \tag{6}$$

To show this, it is sufficient to establish the correctness of the following statement:

$$\{\psi a \in \Phi \mid \psi \in r_L(u)\} \cup \{X_i \mid ua \in L_i\} = \{\xi \in \Phi_1 \mid ua \in \xi(L)\}. \tag{7}$$

Again, for the variables X_i, their membership in both sides is defined identically. Obviously, $ua \notin \varepsilon(L)$ and $ua \notin \psi c(L)$ for any ψ and $c \neq a$. The statement $ua \in \psi a(L)$ is equivalent to $u \in \psi(L)$, which, by the induction hypothesis, holds if and only if $\psi \in r_L(u)$. This shows (7), and thus (6) by Lemma 3. □

Since the vector $L = (L_1, \dots, L_n)$ is a solution of $\varphi(X_1, \dots, X_n) = \varnothing$ if and only if $w \notin \varphi(L)$ for all $w \in \Sigma^*$, this lemma implies the following characterization of the runs corresponding to solutions:

Proposition 1. The vector $L = (L_1, \dots, L_n)$ is a solution of the equation $\varphi(X_1, \dots, X_n) = \varnothing$ if and only if $\varphi \notin r_L(w)$ for every $w \in \Sigma^*$.

Consequently, if we remove from A all states containing φ, then we obtain an automaton whose runs are in a 1–1-correspondence with the solutions of $\varphi(X_1, \dots, X_n) = \varnothing$. In addition, we can make this automaton trim without losing any runs/solutions. Let us call the resulting ILTA A_φ. Obviously, the size of A_φ is exponential in the size of φ, and this automaton can be constructed in exponential time.

Proposition 2. For every language equation $\varphi(X_1, \dots, X_n) = \varnothing$ of the form (2) one can construct in exponential time a trim ILTA A_φ whose states are subsets of the set of strict subexpressions of φ such that the mapping $r \mapsto L^r = (L_1^r, \dots, L_n^r)$ defined as $L_i^r := \{w \mid X_i \in r(w)\}$ is a bijection between the runs of A_φ and the solutions of $\varphi(X_1, \dots, X_n) = \varnothing$.

Let us illustrate the construction of A_φ with a small example. Consider the following language equation over the alphabet $\Sigma = \{a\}$ and in the variables X, Y:

$$\sim\!(X \cup Ya) = \varnothing. \tag{8}$$

The set of subexpressions of $\varphi := {\sim}(X \cup Ya)$ is

$$\Phi = \{\varepsilon,\ Ya,\ X,\ Y,\ X \cup Ya,\ {\sim}(X \cup Ya)\},$$

and the subsets Φ_0 and Φ_1 are given by

$$\Phi_0 = \{\varepsilon,\ Ya\} \quad \text{and} \quad \Phi_1 = \{\varepsilon,\ Ya,\ X,\ Y\}.$$

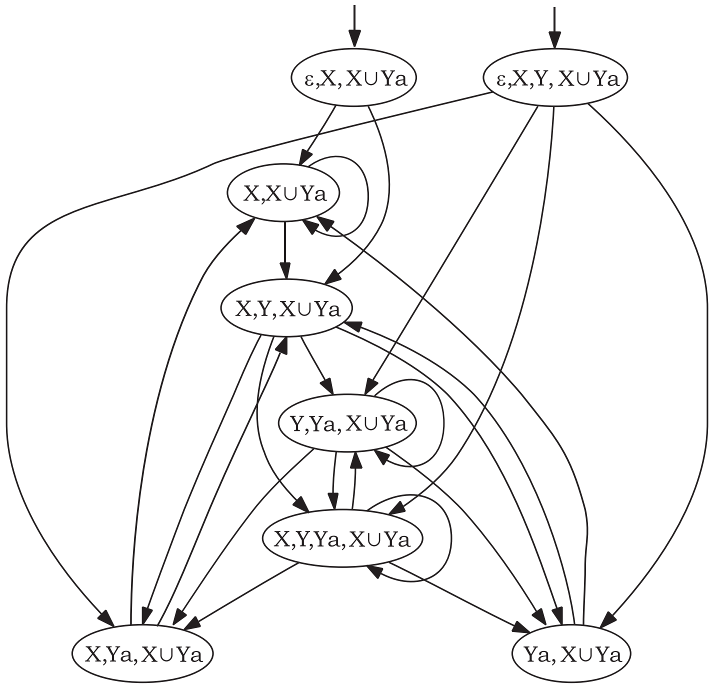

Figure 1. The trim ILTA for the equation (8), where all arcs are labeled by a.

Instead of first constructing the automaton A, then removing the states containing φ, and finally making the resulting automaton trim, we immediately construct an automaton consisting of those states not containing φ, and where every state is reachable from an initial state. First, consider the initial states of the original automaton A constructed from φ. The set $select(\{\varepsilon\})$ contains four elements: $\{\varepsilon\}$, $\{\varepsilon, X\}$, $\{\varepsilon, Y\}$, and $\{\varepsilon, X, Y\}$. After $closure$ is applied, the following initial states are obtained: $q_0 = \{\varepsilon, {\sim}(X \cup Ya)\}$, $q_0' = \{\varepsilon, X, X \cup Ya\}$, $q_0'' = \{\varepsilon, Y, {\sim}(X \cup Ya)\}$, and $q_0''' = \{\varepsilon, X, Y, X \cup Ya\}$. The states q_0 and q_0'' contain φ, and thus are not states of A_φ. Consequently, we begin our construction with the set of initial states $Q_0 := \{q_0', q_0'''\}$.

Consider the transitions from q_0''' by a. The only concatenation in $\{\psi a \in \Phi \mid \psi \in q_0'''\}$ is Ya, and then $closure(q)$ for all $q \in select(\{Ya\})$ yields the states $q_1 = \{Ya, X \cup Ya\}$, $q_1' = \{Ya, X, X \cup Ya\}$, $q_1'' = \{Ya, Y, X \cup Ya\}$, and $q_1''' = \{Ya, X, Y, X \cup Ya\}$. None of these states contains φ, and thus we define $\delta(q_0''', a) := \{q_1, q_1', q_1'', q_1'''\}$.

Next, consider the transitions from q_0' by a. There is no concatenation in the set $\{\psi a \in \Phi \mid \psi \in q_0'\}$, and thus we must construct the closures of the sets in $select(\varnothing)$, which yields the states $q_2 = \{{\sim}(X \cup$

$Ya)\}$, $q_2' = \{X, X \cup Ya\}$, $q_2'' = \{Y, \sim(X \cup Ya)\}$, and $q_2''' = \{X, Y, X \cup Ya\}$. If we remove the states containing φ, then we obtain $\delta(q_0', a) := \{q_2', q_2'''\}$.

If we continue this process until all states reachable from the initial states are constructed, then we obtain the ILTA shown in Figure 1. Since this automaton is already trim, it is the automaton A_φ for the equation (8).

3.2. Counting the number of solutions

As an immediate consequence of Proposition 2, solvability and unique solvability of a language equation can be characterized as follows:

Proposition 3. A language equation $\varphi = \varnothing$ with one-sided concatenation has

- at least one solution if and only if the corresponding ILTA A_φ is non-empty.
- exactly one solution if and only if the corresponding ILTA A_φ is non-empty and deterministic.

Before we can characterize the case of finitely many solutions, we must introduce some notation.

Definition 2. Let $A = (\Sigma, Q, Q_0, \delta)$ be an ILTA. A state $q \in Q$ is *cyclic* if $q \in \delta(q, w)$ for some $w \in \Sigma^+$, and it is *branching* if $|\delta(q, a)| > 1$ for some $a \in \Sigma$.

Paths in an ILTA are defined as usual, that is, a *finite path in A* is a finite sequence $q_1 a_1 q_2 a_2 \ldots a_{\ell-1} q_\ell \in Q(\Sigma Q)^*$ such that $q_{i+1} \in \delta(q_i, a_i)$ for all i ($1 \leqslant i < \ell$). If there is such a path, then q_ℓ is *reachable* from q_1. Similarly, an *infinite path in A* is an infinite sequence $q_1 a_1 q_2 a_2 \ldots \in Q(\Sigma Q)^\omega$ such that $q_{i+1} \in \delta(q_i, a_i)$ for all $i \geq 0$.

Lemma 6. A trim ILTA $A = (\Sigma, Q, Q_0, \delta)$ has finitely many runs if and only if no branching state is reachable from any cyclic state.

Proof:
If there are no paths from cyclic to branching states, then every infinite path in the ILTA can contain branching states only among the first $|Q|$ nodes, and after that the transitions become completely deterministic. Therefore, the first $|Q|$ levels of every run determine it completely, and thus the number of different runs is bounded by the number of different mappings from $\{w \in \Sigma^* \mid |w| \leqslant |Q|\}$ to Q, which is finite.

Suppose the condition does not hold, that is, there exists a cyclic state p, with $p \in \delta(p, u)$ for $u \in \Sigma^+$, and a branching state q, with $q', q'' \in \delta(q, a)$, $q \neq q'$, such that $q \in \delta(p, v)$ for some $v \in \Sigma^*$. Let $p\alpha p \in Q(\Sigma Q)^+$ be a path from p to p by u, and let $p\beta q \in Q(\Sigma Q)^*$ be a path from p to q by v. Without loss of generality we may assume that the path $p\alpha p$ contains at most one occurrence of q; it could be shortened otherwise. For the same reason, we can also assume that the path $p\beta q$ does not contain any internal occurrences of q. If $p\alpha p$ contains an occurrence of q, and the next symbol in the path is a, assume without loss of generality that the next state is q'.

Since A is assumed to be trim, there is a state $q_0 \in Q_0$ and a string $w \in \Sigma^*$, such that $p \in \delta(q_0, w)$. Let $q_0\gamma p$ be the corresponding path. Then, for every $\ell \geqslant 0$, there exists the following finite path in A:

$$q_0 \gamma p (\alpha p)^\ell \beta q a q'' \tag{9}$$

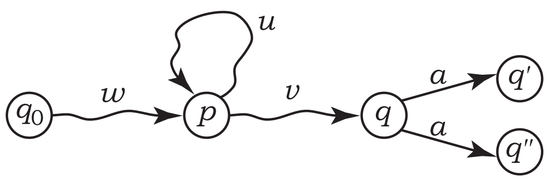

Figure 2. A path from a cyclic to a branching state inducing infinitely many runs.

Since A is trim, we can construct a run r_ℓ of A such that (i) r_L contains this path, and (ii) every transition from q by a except for the last one in this path goes to q'. Then the earliest occurrence of the transition from q by a to q'' in r_ℓ takes place at the end of the finite path (9), which makes the runs corresponding to different numbers ℓ_1, ℓ_2 pairwise distinct. Thus infinitely many different runs have been constructed. □

The condition in this lemma can obviously be tested in time polynomial in the size of the ILTA since it is basically a reachability problem. The conditions in the previous proposition can trivially be tested in time polynomial in the size of A_φ. Since the size of A_φ is exponential in the size of φ, we thus obtain the following complexity upper-bounds:

Theorem 1. The problems of testing whether a language equation with one-sided concatenation has a solution, a unique solution, or finitely many solutions are decidable in deterministic exponential time.

An EXPTIME decision procedure for the solvability problem was already sketched by Aiken et al. [1]. The other two results are new. Regarding the cardinality of the solution set, it remains to show how we can decide whether an equation has countably or uncountably many solutions. For this purpose, we adapt Niwiński's [24] condition for countability of the language accepted by a Rabin tree automaton to our situation of counting runs of ILTAs.[3] If A is an ILTA and q one of its states, then a *q-run* is defined like a run, with the only exception that instead of requiring that the root is labeled with an initial state we require that it is labeled with q. Two q-runs r_1, r_2 are called *essentially different*, if there are words v_1, v_2, w, such that

- $r_1(v_1) = q = r_2(v_2)$ and v_1, v_2 are not the empty word,
- $r_1(w) \neq r_2(w)$ and w has neither v_1 nor v_2 as prefix.

Proposition 4. (Niwiński)
A trim ILTA has uncountably many runs if and only if it has two essentially different q-runs, for some state q.

In contrast to the previous conditions, it is not immediately clear how this condition can be decided in time polynomial in the size of the ILTA. The proposed solution is to reduce this problem to the emptiness problem for Büchi tree automata, which is done in the following lemma.

Lemma 7. For a given ILTA A we can decide in polynomial time whether it has uncountably many runs or not.

[3]Actually, we never use that the automaton has independent transitions, and thus the results shown below also hold for arbitrary looping tree automata.

Proof:
Given two runs r_1, r_2, we denote by (r_1, r_2) the tree whose nodes $u \in \Sigma^*$ are labeled with $(r_1(u), r_2(u))$. For every state q of A we construct a Büchi automaton $\mathcal{B}_q$ that accepts exactly the trees (r_1, r_2) where r_1, r_2 are essentially different q-runs. We can then apply the emptiness test for Büchi automata to $\mathcal{B}_q$ for each states q to test whether there are essentially different q-runs of A. Recall that a Büchi tree automaton differs from a looping tree automaton in that it has a set of final states, and that a run of such an automaton is accepting if in every path at least one final state occurs infinitely often. Also recall that the emptiness test for Büchi tree-automata is polynomial in the size of the automaton [39].

The states of the Büchi automaton $\mathcal{B}_q$ are of the form (q_1, q_2, M) where q_1, q_2 are states of A and M is a subset of $\{\textit{qfirst?}, \textit{qsecond?}, \textit{diff?}, \textit{initial}\}$. The idea underlying the third component M is the following:

- if M contains *qfirst?*, then we are looking for a node in the subtree below with q in the first component: this is a search for v_1 in the definition of essentially different runs;
- if M contains *qsecond?*, then we are looking for a q in the second component of some node in the subtree below;
- *diff?* says that we are looking for a node in the subtree below with different first and second components;
- $\textit{initial}$ is present only in the initial state.

The automaton $\mathcal{B}_q$ starts with the initial state $(q, q, \{\textit{qfirst?}, \textit{qsecond?}, \textit{diff?}, \textit{initial}\})$. If it is in the state (q_1, q_2, M) and it reads the corresponding symbol (q_1, q_2), then it can make the following transitions:[4]

$$(q_1, q_2, M), (q_1, q_2) \to ((p_{11}, p_{21}, M_1), \dots, (p_{1m}, p_{2m}, M_m))$$

whenever the following conditions are satisfied:

1. $q_1 \to (p_{11}, \dots, p_{1m})$ and $q_2 \to (p_{21}, \dots, p_{2m})$ are transitions in the ILTA (now represented as a tree automaton, not an NFA).
2. $M_1, \dots, M_n$ are subsets of $M \setminus \{\textit{initial}\}$.
3. If $\textit{qfirst?} \in M$, then
    - *qfirst?* belongs to exactly one of $M_1, \dots, M_m$, or
    - $q_1 = q$ and $\textit{diff?}, \textit{initial} \notin M$, and *qfirst?* belongs to none of $M_1, \dots, M_m$.

   Note that the choice of which alternative to take realizes the non-deterministic decision whether the current node is v_1 (second case) or not (first case). In case we have $\textit{diff?} \in M$, we cannot choose the second case since this would then violate the condition that v_1 cannot be a prefix of w. The same is true if *initial* is in M, because this would violate the condition that v_1 is not the empty word. In the first case, we also make a non-deterministic decision, in which a successor tree containing v_1 will be guessed.

[4]If it reads a symbol different from the first two components of its state, then no transition is possible.

4. If *qsecond?* $\in M$, then

   - *qsecond?* belongs to exactly one of $M_1, \dots, M_m$, or
   - $q_2 = q$ and *diff?*$, initial \notin M$, and *qsecond?* belongs to none of $M_1, \dots, M_m$.

   The explanation for this is analogous to the one for *qfirst?*. Note that we can, of course, also decide that $v_1 = v_2$ if both q_1 and q_2 are equal to q.

5. If *diff?* in M then

   - *diff?* belongs to exactly one of $M_1, \dots, M_m$, or
   - $q_1 \neq q_2$ and *diff?* belongs to none of $M_1, \dots, M_m$.

   We are looking for the difference in the first or second subtree. If q_1 is different from q_2, we can also decide that this is w.

This completes the description of the transition relation of $\mathcal{B}_q$. The set of final states of $\mathcal{B}_q$ consists of all the states (q_1, q_2, M) where M is empty.

It is easy to see that this automaton indeed accepts exactly the trees (r_1, r_2) where r_1 and r_2 are essentially different q-runs of A. In fact, in a run of $\mathcal{B}_q$ we eventually get rid of all states with non-empty M in all paths, if appropriate nodes v_1, v_2, w are found. □

As an immediate consequence of this proposition we obtain:

Theorem 2. The problem of testing whether a language equation with one-sided concatenation has countably many solutions is decidable in exponential time.

Let us apply our method to determine the cardinality of the set of solutions of the equation (8), whose trim ILTA is given in Figure 1. The ILTA is non-empty, and hence the equation has solutions. It is non-deterministic (actually, it has two initial states, and each of its states has multiple transitions by a), and hence the equation has multiple solutions. There are paths from cyclic states to branching states. For example, consider the state $\{X, X \cup Ya\}$, which is cyclic because of the self-loop, and which is itself branching. Consequently, there are infinitely many solutions.

Finally, let us construct a pair of essentially different q-runs, corresponding to the condition of Proposition 4. Let $q = \{X, X \cup Ya\}$, $w = a$, $v_1 = aa$ and $v_2 = aaa$. The required runs are as follows (since the branching is unary, trees degrade to paths):

$$\begin{tikzcd}
\{X, X \cup Ya\} \arrow[r, "a"] & \{X, X \cup Ya\} \arrow[r, "a"] & \{X, X \cup Ya\} \arrow[r, "a"] & \cdots \\
 & w \updownarrow & v_1 \uparrow & v_2 \downarrow \\
\{X, X \cup Ya\} \arrow[r, "a"] & \{X, Y, X \cup Ya\} \arrow[r, "a"] & \{X, Ya, X \cup Ya\} \arrow[r, "a"] & \{X, X \cup Ya\} \arrow[r, "a"] & {}
\end{tikzcd}$$

The existence of these paths implies that the ILTA has uncountably many runs, and therefore the equation has uncountably many solutions.

3.3. Least and greatest solutions

As pointed out at the end of Subsection 2.1, we must compare solution vectors not on all components, but only on those components corresponding to a set of focus variables. Let $\varphi(X_1, \ldots, X_n, Y_1, \ldots, Y_\ell) = \varnothing$ be a language equation with one-sided concatenation, and $\{X_1, \ldots, X_n\}$ be the set of focus variables. Given vectors of languages $L = (L_1, \ldots, L_n, L_{n+1}, \ldots, L_{n+\ell})$ and $L' = (L'_1, \ldots, L'_n, L'_{n+1}, \ldots, L'_{n+\ell})$, we define $L \preceq L'$ if and only if $L_i \subseteq L'_i$ for all $i = 1, \ldots, n$. Unless $\ell = 0$, the relation $\preceq$ is not a partial order, but only a pre-order, because any two vectors of languages that coincide on the focus variables are deemed equivalent. Accordingly, a language equation may have multiple least or greatest solutions with respect to $\preceq$, but these least or greatest solutions will always coincide on the focus variables.

Let $A_\varphi = (\Sigma, Q, Q_0, \delta)$ be the ILTA corresponding to the above language equation with focus variables $X_1, \ldots, X_n$. We define a preorder on its set of states Q as follows:

$$q \preccurlyeq q' \text{ if and only if } q \cap \{X_1, \ldots, X_n\} \subseteq q' \cap \{X_1, \ldots, X_n\}.$$

This preorder on states defines the following preorder on runs of A: for any $r, r' : \Sigma^* \to Q$ we say that $r \preccurlyeq r'$ if $r(w) \preccurlyeq r'(w)$ for all $w \in \Sigma^*$.

As an easy consequence of the definition of the mapping $L \mapsto r_L$ we obtain that this mapping is a preorder isomorphism:

Lemma 8. Let L, L' be vectors of languages. Then $L \preceq L'$ if and only if $r_L \preccurlyeq r_{L'}$.

Consequently, to decide whether the equation $\varphi = \varnothing$ has a least/greatest solution with respect to $\preceq$, it is enough to decide whether A_φ has a least/greatest run with respect to $\preccurlyeq$. In the following, we show how to decide in polynomial time whether a given ILTA has a least run with respect to any preorder on its states. Greatest runs can be represented by applying the same construction to the preorder $\succcurlyeq$.

Definition 3. Let $A = (\Sigma, Q, Q_0, \delta)$ be an ILTA, let $\preccurlyeq$ be a preorder on Q. Define another relation $\sqsubset$ on Q as follows: $q \sqsubset q'$ if and only if there exists a run r with root label q, such that, for every run r' with root label q', we have $r \preccurlyeq r'$.

The relation $\sqsubset$ is transitive, but it is not necessarily reflexive, and hence, strictly speaking, is not a preorder. Nevertheless, we will use it as if it were a preorder, and, in particular, least elements with respect to $\sqsubset$ will be considered. A subset $S \subseteq Q$ is said to have a least element $q \in S$, if $q \sqsubset q'$ for all $q' \in S$. Note that this condition also covers the case where $q = q'$, i.e., for a least element q we require in particular $q \sqsubset q$.

The following lemma is an immediate consequence of the definition of $\sqsubset$.

Lemma 9. An ILTA $A = (\Sigma, Q, Q_0, \delta)$ has a least run with respect to the preorder $\preccurlyeq$ on Q if and only if Q_0 has a least element with respect to $\sqsubset$.

The next lemma presents an algorithm for constructing $\sqsubset$.

Lemma 10. For every trim ILTA $A = (\Sigma, Q, Q_0, \delta)$ and for every polynomial-time decidable preorder $\preccurlyeq$ on Q, the corresponding relation $\sqsubset$ on Q can be constructed in time polynomial in $|Q|$.

Proof:
We show that the complement of $\sqsubset$ can be computed as

$$R = \bigcup_{k=0}^{\infty} R_k, \tag{10a}$$

where

$$R_0 = \{(q,q') \mid q \not\preccurlyeq q'\}, \tag{10b}$$
$$R_{k+1} = R_k \cup \{(q,q') \mid \exists a \in \Sigma.\forall q_a \in \delta(q,a).\exists q'_a \in \delta(q',a).\ R_k(q_a,q'_a)\} \tag{10c}$$

From this definition it immediately follows that R, and thus also its complement $\sqsubset$, can be computed in polynomial time.

By (10b,10c), R satisfies the following equation

$$R = \{(q,q') \mid q \not\preccurlyeq q' \text{ or } \exists a \in \Sigma.\forall q_a \in \delta(q,a).\exists q'_a \in \delta(q',a).\ R(q_a,q'_a)\},$$

and therefore

$$\neg R = \{(q,q') \mid q \preccurlyeq q' \text{ and } \forall a \in \Sigma.\exists q_a \in \delta(q,a).\forall q'_a \in \delta(q',a).\ \neg R(q_a,q'_a)\}. \tag{11}$$

It is sufficient to prove that (i) $R(q,q')$ implies $q \not\sqsubset q'$ and (ii) $\neg R(q,q')$ implies $q \sqsubset q'$.

Part i: We prove that $R_k(q,q')$ for some $k \geqslant 0$ implies $q \not\sqsubset q'$ by induction on k.

Base case: if $R_0(q,q')$, then $q \not\preccurlyeq q'$ by (10b). Therefore, for every run r with root label q and for every run r' with root label q' we know that $r(\varepsilon) = q \not\preccurlyeq q' = r'(\varepsilon)$, and thus, clearly, $q \not\sqsubset q'$.

Induction step: let $(q,q') \in R_{k+1} \setminus R_k$, and let $a \in \Sigma$ be the symbol promised in (10c). In order to show that $q \not\sqsubset q'$, consider an arbitrary run r starting from q. Let $q_a = r(a)$ and define a run r_a with root q_a as $r_a(u) = r(au)$ for all $u \in \Sigma^*$.

According to (10c), for the state q_a there exists a state $q'_a \in \delta(q',a)$, such that $R_k(q_a,q'_a)$. By the induction hypothesis, this implies $q_a \not\sqsubset q'_a$. That is, for the run r_a with root q_a there exists a run r'_a with root q'_a such that $r_a \not\preccurlyeq r'_a$, that is, $r_a(w) \not\preccurlyeq r'_a(w)$ for some $w \in \Sigma^*$. Construct a run r' with root q', such that $r'(au) = r'_a(u)$ for all $u \in \Sigma^*$; since A is trim, this run can also be defined in some way for strings beginning from symbols other than a. We thus have $r(aw) = r_a(w) \not\preccurlyeq r'_a(w) = r'(aw)$, and hence $r \not\preccurlyeq r'$, which completes the proof that $q \not\sqsubset q'$.

Part ii: We show that $\neg R(q,q')$ implies $q \sqsubset q'$.

Let us construct a run r starting from q and simultaneously verify that for every run r' starting from q' we have $r \preccurlyeq r'$. The run r is defined inductively on the length of w such that, for every r' starting from q', we have $\neg R(r(u), r'(u))$ for all prefixes u of w.

Base case: $w = \varepsilon$. Here $r(\varepsilon) = q$, $r'(\varepsilon) = q'$ and $\neg R(q,q')$ by assumption.

Induction step. Consider a string $w \in \Sigma^*$ and assume $\neg R(r(u), r'(u))$ for all prefixes of w. In particular, $\neg R(r(w), r'(w))$, and, by (11), for every $a \in \Sigma$ there exists a certain state $q_a \in \delta(r(w), a)$ satisfying the property stated in (11). Define $r(wa)$ as q_a, and consider the state $q'_a := r'(wa) \in \delta(r'(w), a)$: because of our choice of q_a we have for this q'_a that $\neg R(q_a, q'_a)$ holds. This finishes the induction step.

Having constructed such a run r, it is left to notice that, by (11), $\neg R(r(w), r'(w))$ for all $w \in \Sigma^*$ implies $r(w) \preccurlyeq r'(w)$ for all w, and hence $r \preccurlyeq r'$, which proves $q \sqsubset q'$. $\square$

Since the size of A_φ is exponential in the size of φ, we thus obtain the following complexity upper bound for deciding the existence of a least solution. (Greatest solutions can be treated analogously.)

Theorem 3. The problem of testing whether a language equation with one-sided concatenation has a least (greatest) solution is decidable in ExpTime.

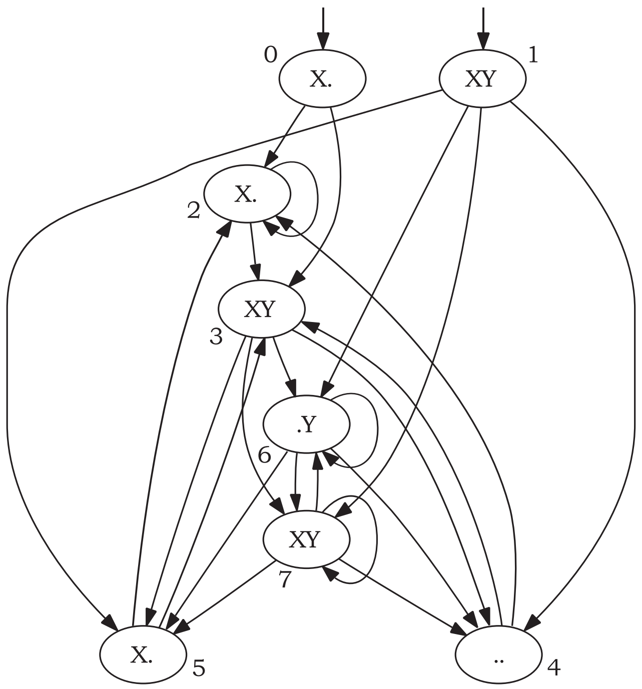

Figure 3. The automaton from Figure 1 with its states numbered, showing the variables only.

Let us return to our example: the equation (8) and the corresponding ILTA given in Figure 1. In order to determine whether the ILTA has a least run, we need to construct the preorder $\preccurlyeq$ and the corresponding relation $\sqsubset$. Let us name the states of this automaton by numbers, as shown in Figure 3. Note that we have only represented the variables contained in each state, since this is the relevant information for determining both relations.

The preorder $\preccurlyeq$ is computed simply by containment of variable components, and hence, for instance, $0 \preccurlyeq 0$, $0 \preccurlyeq 1$, $0 \preccurlyeq 3$, $0 \preccurlyeq 5$, $0 \preccurlyeq 7$, $1 \preccurlyeq 3$, $1 \preccurlyeq 7$, etc. On the other hand, $0 \not\preccurlyeq 4$, $0 \not\preccurlyeq 6$, $1 \not\preccurlyeq 0$, etc.

The computation of $\not\sqsubseteq$ begins with computing the negation of $\preccurlyeq$:

$$\begin{aligned} R_0 \quad = \quad & \{(0,4),(0,6),(1,0),(1,2),(1,4),(1,5),(1,6),(2,4),(2,6),(3,0), \\ & (3,2),(3,4),(3,5),(3,6),(5,4),(5,6),(6,0),(6,2),(6,4),(6,5), \\ & (7,0),(7,2),(7,4),(7,5),(7,6)\}. \end{aligned}$$

More elements are added to R in the next steps of the iteration. For instance, consider the pair $(2,3) \notin R_0$ and consider all transitions (with a) from 2, which yield the states 2 and 3. For the transition from 2 to

3, there exists a transition from 3 to 6, and we have $(3, 6) \in R_0$. For the transition from 2 to 2, the same transition from 3 to 6 yields the pair $(2, 6) \in R_0$. Therefore, $(2, 3) \in R_1$.

Using this pair, we can determine that $(0, 2) \in R_2$. Indeed, for the transition from 0 to 2 there is a transition from 2 to 3, and we have $(2, 3) \in R_1$. For the transition from 0 to 3 there exists a transition from 2 to 2, and we have $(3, 2) \in R_0 \subseteq R_1$.

Proceeding in this way, we eventually conclude that $R = Q \times Q$, that is, $\sqsubset\ = \varnothing$. Therefore, the elements of Q_0 are incomparable with respect to $\sqsubset$ (this would be the case even if there were only one state in Q_0), and hence Lemma 9 implies that the automaton does not have a least run. Consequently, the equation does not have a least solution.

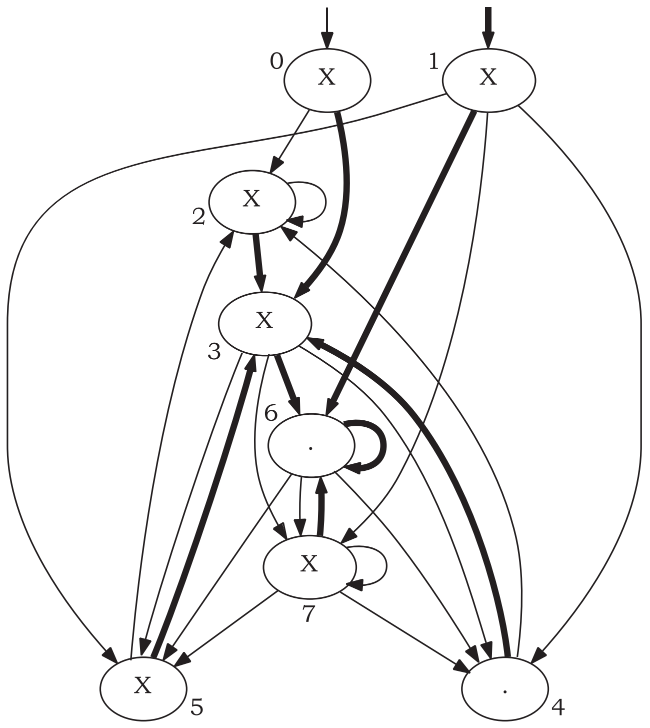

Figure 4. The automaton from Fig. 3, but now showing only the focus variable X.

Let us now consider the case where X is the only focus variable. The revised version of Figure 3 is given in Figure 4, where only the focus variable X is shown (thick lines will be explained later in Section 3.4). The values of the relation $\preccurlyeq$ are presented on the left-hand side of Table 1. In this case, $q \preccurlyeq q'$ for most pairs of states, except for those, where q contains X, while q' does not. Thus, we have

$$\begin{aligned} R_0 \ = \ &\{(0,4),(0,6),(1,4),(1,6),(2,4),(2,6), \\ &(3,4),(3,6),(5,4),(5,6),(7,4),(7,6)\}. \end{aligned}$$

We can determine that $(2, 3) \in R_1$ in the same way as in the previous case. However, $(3, 2) \notin R_0$, and thus $(0, 2)$ is not put into R_2. Overall, the iteration adds only the following pairs to R:

$$\begin{aligned} R \setminus R_0 \ = \ &\{(0,1),(0,3),(0,7),(2,1),(2,3),(2,7),(4,1), \\ &(4,3),(4,6),(4,7),(5,1),(5,3),(5,7)\}. \end{aligned}$$

The relation $\sqsubset$ contains the remaining 39 pairs. It is given on the right-hand side of Table 1.

$\preccurlyeq$	0	1	2	3	4	5	6	7
0	+	+	+	+	−	+	−	+
1	+	+	+	+	−	+	−	+
2	+	+	+	+	−	+	−	+
3	+	+	+	+	−	+	−	+
4	+	+	+	+	+	+	+	+
5	+	+	+	+	−	+	−	+
6	+	+	+	+	+	+	+	+
7	+	+	+	+	−	+	−	+

$\sqsubset$	0	1	2	3	4	5	6	7
0	+	−	+	−	−	+	−	−
1	+	+	+	+	−	+	−	+
2	+	−	+	−	−	+	−	−
3	+	+	+	+	−	+	−	+
4	+	−	+	−	+	+	−	−
5	+	−	+	−	−	+	−	−
6	+	+	+	+	+	+	+	+
7	+	+	+	+	−	+	−	+

Table 1. The relations $\preccurlyeq$ and $\sqsubset$ for the case of the focus variable X.

Since $1 \sqsubset 0$ and $1 \sqsubset 1$, the set $Q_0 = \{0, 1\}$ has 1 as least element. By Lemma 9, this implies that the automaton has a least run, and thus the equation has a least solution with respect to the focus variable X.

3.4. Computing regular solutions

Until now, we have considered only decision problems, which require a *yes* or *no* answer. If a language equation has a (unique, least, greatest) solution, one might also be interested not just in knowing that it exists, but also in computing such a solution. However, solutions are vectors of possibly infinite languages, so how can one represent such solutions in a finite way? If the solution is *regular*, that is, if all its components are regular languages, then it can be represented by finite automata for the component languages. Although, in general, solutions of language equations with one-sided concatenation need not be regular, one can show that a solvable language equation always has a regular solution, and that least and greatest solutions are always regular. One way of showing this is to express (least, greatest) solutions in Rabin's monadic second-order logic [33], and use well-known results for this logic. Our representation of solutions by runs of an effectively constructable ILTA provides an easy and natural way of determining regular solutions. It also yields a standalone proof of regularity of unique/least/greatest solutions of language equations with one-sided concatenation.

Given a non-empty trim ILTA $A = (\Sigma, Q, Q_0, \delta)$, the deterministic ILTA $B = (\Sigma, Q, \{q_0\}, \delta')$ is called a *deterministic subautomaton* of A if $q_0 \in Q_0$ and $\delta'(q, a) \subseteq \delta(q, a)$ with $|\delta'(q, a)| = 1$ for all $q \in Q$ and $a \in \Sigma$. Obviously, a non-deterministic ILTA can have exponentially many deterministic subautomata, but we can compute a single one of them by choosing an arbitrary element q_0 of Q_0 and, for every $q \in Q$ and $a \in \Sigma$, an arbitrary element of q' of $\delta(q, a)$ and defining $\delta'(q, a) := \{q'\}$.

Lemma 11. Assume that the language equation $\varphi(X_1, \ldots, X_n) = \varnothing$ is solvable. Let B_φ be a deterministic subautomaton of A_φ and let $B_\varphi^{(i)}$ be the DFA obtained from B_φ by using the set $F_i := \{q \mid X_i \in q\}$ $(i = 1, \ldots, n)$ as the set of final states. Then the vector $(L_1, \ldots, L_n)$ with $L_i = L(B_\varphi^{(i)})$ is a regular solution of $\varphi(X_1, \ldots, X_n) = \varnothing$.

Proof:
Since B_φ is deterministic, it has a unique run r, and since it is a subautomaton of A_φ, r is also a run of A_φ. Thus, the corresponding vector of languages $L^r = (L_1^r, \dots, L_n^r)$ with $L_i^r = \{w \mid X_i \in r(w)\}$ is a solution of $\varphi(X_1, \dots, X_n) = \varnothing$. Obviously, we have $L_i^r = L(B_\varphi^{(i)})$, and thus $(L_1, \dots, L_n) = (L_1^r, \dots, L_n^r)$ is a regular solution of $\varphi(X_1, \dots, X_n) = \varnothing$. □

The following theorem is an immediate consequence of this lemma.

Theorem 4. Let $\varphi(X_1, \dots, X_n, Y_1, \dots Y_\ell) = \varnothing$ be a solvable language equation with one-sided concatenation. Then it has a regular solution $(L_1, \dots, L_n)$ such that deterministic finite automata recognizing the languages $L_1, \dots, L_n$ can be constructed in exponential time.

For the case of a unique solution, the trim ILTA A_φ is deterministic, and thus is the only deterministic subautomaton of itself.

Theorem 5. Assume that the language equation $\varphi(X_1, \dots, X_n) = \varnothing$ has a unique solution $(L_1, \dots, L_n)$. Then this solution is regular, and deterministic automata accepting the languages L_i can be obtained as follows: for each $i = 1, \dots, n$, let $A_\varphi^{(i)}$ be the DFA obtained from A_φ by using the set $F_i := \{q \mid X_i \in q\}$ as the set of final states; then $L_i = L(A_\varphi^{(i)})$.

In order to obtain automata representing a least solution, we must modify the constructed ILTA into an ILTA that has a unique least run. Let $A = (\Sigma, Q, Q_0, \delta)$ be an ILTA, which has one or more least runs with respect to the preorder $\preccurlyeq$. Define the corresponding relation $\sqsubset$ as in Section 3.3.

Construct another ILTA $B = (\Sigma, Q, Q_0', \delta')$ as follows. Its initial set is defined as $Q_0' := \{q_0\}$, where $q_0 \in Q_0$ is any least element of Q_0 with respect to $\sqsubset$: such a q_0 exists by Lemma 9. The transitions are defined by

$$\delta'(q, a) := \begin{cases} \{q'\}, & \text{where } q' \text{ is any least element of } \delta(q,a) \text{ with respect to } \sqsubset, \\ & \text{if such an element exists} \\ \varnothing, & \text{if } \delta(q,a) \text{ has no least element} \end{cases}$$

for each $q \in Q$ and $a \in \Sigma$.

Lemma 12. Let A be an ILTA that has at least one least run with respect to $\preccurlyeq$, and let B be an ILTA constructed from A as defined above. Then the automaton B has a unique run, which is among the least runs of A.

Proof:
Since $|Q_0'| \leqslant 1$ and $|\delta'(q,a)| \leqslant 1$ for all q and a, B has at most one run. It has to be proved that B has a run r and this run is one of the least runs of A.

We define $r(w)$ for $w \in \Sigma^*$ by induction on the length of w. Simultaneously, we prove that, for every $n \geqslant 0$, there exists a least run $\widehat{r}$ of A such that $r(u) = \widehat{r}(u)$ for very $u \in \Sigma^*$ with $|u| \leqslant n$.

Base case. Let $\widehat{r}$ be any least run of A and define $r(\varepsilon) := \widehat{r}(\varepsilon)$.

Induction step. Let $r(u) = \widehat{r}(u)$ for all $u \in \Sigma^*$ such that $|u| \leqslant n$, where $\widehat{r}$ is a least run of A. We need to define $r(wa)$ for every $w \in \Sigma^*$ of length n and for every $a \in \Sigma$. For every such w and a, consider the state $q := r(w)$. (Note that $r(w) = \widehat{r}(w)$ by the induction hypothesis.)

Let us first show that $\delta(q, a)$ contains a least element with respect to $\sqsubset$. Let $\widehat{q}_a = \widehat{r}(wa)$ and define the run $\widehat{r}_a$ from $\widehat{q}_a$ as $\widehat{r}_a(v) = \widehat{r}(wav)$ for all $v \in \Sigma^*$. If $\delta(q, a)$ does not contain a least element, then for this state $\widehat{q}_a$ there exists a state q'_a, such that $\widehat{q}_a \not\sqsubset q'_a$. The latter implies that for the run $\widehat{r}_a$ from $\widehat{q}_a$ there exists a run r'_a from q'_a, such that $\widehat{r}_a \not\preccurlyeq r'_a$, i.e., $\widehat{r}_a(v_0) \not\preccurlyeq r'_a(v_0)$ for some $v_0 \in \Sigma^*$. Construct a new run r' as follows: $r'(wav) = r'_a(v)$ for all $v \in \Sigma^*$ and $r'(u) = \widehat{r}(u)$ for all $v \in \Sigma^* \setminus wa\Sigma^*$. Then $\widehat{r}(wav_0) \not\preccurlyeq r'(wav_0)$, and therefore $\widehat{r} \not\preccurlyeq r'$, which contradicts the assumption that $\widehat{r}$ is one of the least runs.

We have thus proved that there are least elements with respect to $\sqsubset$ in $\delta(q, a)$, and therefore $\delta'(q, a) = \{q_a\}$, where q_a is one of these least elements. Then there exists a run r_a from q_a such that $r_a \preccurlyeq r'_a$ for every run r'_a from q'_a; in particular, $r_a \preccurlyeq \widehat{r}_a$. Define $r(wa)$ as $r_a(\varepsilon)$ and also denote $r_{w,a} := r_a$.

In order finish the induction step, we need to show that there exists another least run $\widetilde{r}$ of A, such that $r(u) = \widetilde{r}(u)$ for all u of length up to $n + 1$. Define such a run $\widetilde{r}$ as follows:

$$\widetilde{r}(u) = \widehat{r}(u) \quad (\text{for all } u \in \Sigma^* \text{ such that } |u| \leqslant n),$$
$$\widetilde{r}(wav) = r_{w,a}(v) \quad (\text{for all } w \in \Sigma^n,\ a \in \Sigma \text{ and } v \in \Sigma^*).$$

Then $\widetilde{r} \preccurlyeq \widehat{r}$, and since $\widehat{r}$ is one of the least runs of A, $\widetilde{r}$ is also one of the least runs of A. This completes the induction step. □

Using this lemma, one can construct finite automata for components of a least or greatest solution of a language equation by processing the corresponding ILTA. This is stated in the following theorem.

Theorem 6. Let $\varphi(X_1, \ldots, X_n, Y_1, \ldots Y_\ell) = \varnothing$ be a language equation with one-sided concatenation, and assume that it has a least (greatest) solution with respect to focus variables $X_1, \ldots, X_n$, with $X_1 = L_1, \ldots, X_n = L_n$. Then the languages $L_1, \ldots, L_n$ are regular, and finite automata recognizing them can be constructed in exponential time.

Proof:
Consider the preorder $\preceq$ on $(n + \ell)$-tuples of languages defined as $(L_1, \ldots, L_n, L_{n+1}, \ldots, L_{n+\ell}) \preceq (L'_1, \ldots, L'_n, L'_{n+1}, \ldots, L'_{n+\ell})$ if $L_i \subseteq L'_i$ for all $i \in \{1, \ldots, n\}$ and assume that $\varphi = \varnothing$ has one or more least solutions with respect to this preorder. Consider the ILTA A_φ corresponding to this language equation, and define the preorder $\preccurlyeq$ on the set of states of A_φ, corresponding to the preorder $\preceq$, as introduced above. By Lemma 8, A_φ has one or more least runs with respect to $\preccurlyeq$, and these runs correspond to the least solutions of $\varphi = \varnothing$.

According to Lemma 10, construct the relation $\sqsubset$ on the set of states of A_φ, corresponding to $\preccurlyeq$. Using this relation, construct an ILTA B from A_φ as described in the above Lemma 12. By this lemma, the unique run of B is one of the least runs of A_φ, and accordingly represents the languages $L_1, \ldots, L_n$ from the least solutions of the language equation $\varphi = \varnothing$. Since B is a deterministic subautomaton of A_φ, Lemma 11 asserts the regularity of these languages and yields DFAs recognizing them. □

Let us now conclude our example by constructing a DFA for the least solution of the equation (8) with respect to the focus variable X. The relevant ILTA is shown in Figure 4. The least element in $Q_0 = \{0, 1\}$ is 1.

Let us determine least elements in $\delta(q, a)$ for all states q. Consider the transitions from 1 to 4, 5, 6 and 7. According to Table 1, $6 \sqsubset 4$, $6 \sqsubset 5$, $6 \sqsubset 6$ and $6 \sqsubset 7$, and therefore 6 is a least element in $\delta(1, a)$ (in fact the only least element). Hence, the deterministic transition from 1 by a is routed to 6. Proceeding in this way, we obtain the transitions marked by thick lines in Figure 4. Further setting the states 4 and 6 as non-accepting and the rest as accepting yields a DFA recognizing the value of X in the least solution.

Once the unreachable states are eliminated, we obtain an automaton with the states 1 and 6, which recognizes the language $\{\varepsilon\}$. This is the value of X in the least solution, with respect to the set of focus variables $\{X\}$. Note that, in this solution, the non-focus variable Y receives the value a^*.

4. The complexity lower-bounds

We show that the decision problems for language equations introduced in Section 2 are EXPTIME-hard already for language equations with one-sided concatenation and union as the only Boolean operation. For solvability, this was already established by Baader and Küsters [2]. In the latter paper, it was also proved that such an equation has a solution if and only if it has a greatest solution, and hence the EXPTIME-hardness of the existence of a greatest solution follows from this result as well.

In the following we will concentrate on the remaining decision problems. Similarly to Baader and Küsters [2], we show their EXPTIME-hardness by a reduction from the intersection emptiness problem for deterministic looping tree automata. A tree processed by such an automaton is represented as a language, and the acceptance of the tree by the automaton is defined by language equations. First, we define this representation and establish its basic properties used in later proofs.

4.1. Representing infinite trees by languages

Given a ranked alphabet Γ, we use the alphabet $\Sigma_\Gamma := \{f^{[i]} \mid f \in \Sigma, 1 \leqslant i \leqslant \operatorname{rank} f\}$ as the alphabet underlying our language equations. For every infinite tree t over Γ, we define a representation of t as a string language over Σ_Γ:

$$S(t) = \{f_1^{[i_1]} \dots f_\ell^{[i_\ell]} \mid \ell \geqslant 0, t \text{ has a path with labels } f_1, \dots, f_\ell, f_{\ell+1}, \text{ in which } f_1 \text{ labels the root of } t, \text{ and each } f_{j+1} \text{ labels the } i_j\text{-th successor of the node with label } f_j\}. \quad (12)$$

The strings in $S(t)$ unambiguously encode finite prefixes of paths in t. The representation of every infinite tree $f(t_1, \dots, t_k)$ can be expressed through the representations of its subtrees $t_1, \dots, t_k$ as follows:

$$S(f(t_1, \dots, t_k)) = \{\varepsilon\} \cup \bigcup_{i=1}^{k} \{f^{[i]}u \mid u \in S(t_i)\}$$

This representation is illustrated in Figure 5, where $\Gamma = \{f, g\}$, with $\operatorname{rank} f = 2$ and $\operatorname{rank} g = 1$.

The following lemma characterizes the languages of the form $S(t)$:

Lemma 13. The language $L \subseteq \Sigma_\Gamma^*$ is of the form $L = S(t)$ for some infinite tree t iff

I. $\varepsilon \in L$;

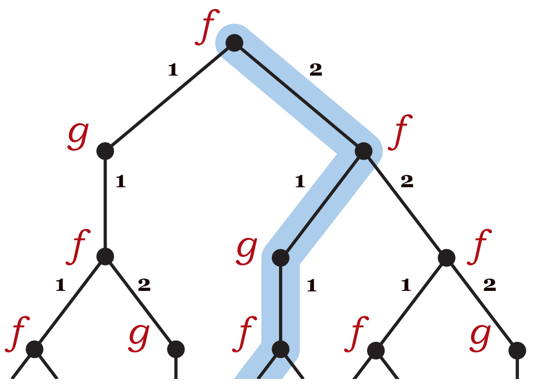

Figure 5. Representing an infinite tree t as a language $S(t)$: the nodes in the selected path are represented by the strings ε, $f^{[2]}$, $f^{[2]}f^{[1]}$, $f^{[2]}f^{[1]}g^{[1]}$.

II. for every $w \in L$ there exists a unique symbol $f \in \Gamma$, such that $wf^{[1]} \in L$;

III. if $wf^{[i]} \in L$, then $wf^{[j]} \in L$ for every j ($1 \leqslant j \leqslant \operatorname{rank} f$);

IV. for every $w \in \Sigma_\Gamma^*$ and $f^{[i]} \in \Sigma_\Gamma$, $wf^{[i]} \in L$ implies $w \in L$.

Proof:
First, we show the *"only-if" direction*. Thus, assume that $L = S(t)$.
(I) $\varepsilon \in S(t)$ by (12).
(II) If $w = f_1^{[i_1]} \dots f_\ell^{[i_\ell]} \in S(t)$, then, by (12), there exists a corresponding path in t; the symbol f we are looking for is the i_ℓ-th successor of the last vertex in this path, that is, the one labeled with f_ℓ. Since f has rank at least one, $wf^{[1]}$ also belongs to $S(t)$.
(III) If $wf^{[i]} \in S(t)$, then the condition in (12) is met, and it is the same for $wf^{[j]}$.
(IV) If the condition in (12) is met for $wf^{[i]}$, then this obviously implies that the condition is also satisfied for w.

Second, we show the *"if" direction*. Thus assume that L satisfies the conditions I–IV. Let us construct an infinite tree t with vertices labeled with Γ, maintaining the following invariant:

For every constructed vertex v labeled with f, consider the path leading to this vertex. If this path is labeled with $f_1, \dots, f_\ell, f_{\ell+1}$ where f_1 labels the root, $f_{\ell+1} = f$ labels v, and each f_{j+1} labels the i_j-th successor of f_j, then the string $f_1^{[i_1]} \dots f_\ell^{[i_\ell]} f^{[1]}$ belongs to L.

Base case. By conditions I and II, there is a unique symbol $f_0 \in \Gamma$, such that $f_0^{[1]} \in L$. Let us label the root with f_0.

Induction step. Consider a path to any vertex labeled with a symbol f of rank n and the corresponding string $f_1^{[i_1]} \dots f_\ell^{[i_\ell]} f^{[1]} \in L$, and let $w = f_1^{[i_1]} \dots f_\ell^{[i_\ell]}$. By condition III, $wf^{[2]}, \dots, wf^{[n]} \in L$. By condition II applied n times, there exist unique symbols $g_1, \dots, g_n \in \Gamma$, such that $wf^{[i]}g_i^{[1]} \in L$ for all i. Let us supply the vertex labeled with f with n successors, which are respectively labeled with $g_1, \dots, g_n$. By our choice of the symbols g_i, the invariant is also satisfied for the paths leading to these new vertices.

This completes our description of the inductive definition of the tree t. It remains to be shown that $L = S(t)$. First note that, by construction, all strings corresponding to finite paths in t belong to L. Hence we have $S(t) \subseteq L$. Second, assume that $L \neq S(t)$, and let $wf^{[i]}$ be the shortest string in L that is not in $S(t)$. Then, $wf^{[1]} \in L$ by condition III. In addition, by condition IV, we have $w \in L$, and also $w \in S(t)$ since it is shorter than $wf^{[i]}$. Thus, when extending the vertex in t corresponding to the last node on the path represented by w, we would have chosen the (unique) symbol f with $wf^{[1]} \in L$ to label the corresponding successor node. But then $wf^{[i]} \in S(t)$. □

The mapping S is extended in the obvious way to sets of trees: $S(T) := \bigcup_{t \in T} S(t)$. We also consider the set of pre-images of this operation

$$S^{-1}(L) := \{t \mid S(t) \subseteq L\}, \tag{13}$$

which acts as an "inverse" of S, in the sense given in the following lemma.

Lemma 14. For every set of trees T, $T \subseteq S^{-1}(S(T))$ and $S(S^{-1}(S(T))) = S(T)$.

Proof:
(I) If $t \in T$, then $S(t) \subseteq S(T)$ by the definition of $S(T)$, and hence $t \in S^{-1}(S(t))$ according to (13).

(II) "$\subseteq$" If $w \in \Sigma_\Gamma^*$ is in $S(S^{-1}(S(T)))$, then there exists a tree $t \in S^{-1}(S(T))$, such that $w \in S(t)$. Hence, $S(t) \subseteq S(T)$, and therefore $w \in S(T)$.

"$\supseteq$" By the first part of the proof, $T \subseteq S^{-1}(S(T))$, which implies $S(T) \subseteq S(S^{-1}(S(T)))$ by the monotonicity of S. □

4.2. Representing looping tree automata by language equations

Let $\mathcal{A} = (Q, \Gamma, \{q_0\}, \Delta)$ be a *deterministic* looping tree automaton over Γ, where Δ is represented as a partial function from $Q \times \Gamma$ to Q^*. We introduce another partial function $\mathfrak{q} : \Sigma_\Gamma^* \to Q$ that simulates the operation of $\mathcal{A}$ on a finite prefix of a single path encoded as in (12). Define $\mathfrak{q}(w)$ inductively on the length of w:

- $\mathfrak{q}(\varepsilon) = q_0$, and
- $\mathfrak{q}(uf^{[i]})$ is defined as the i-th component of $\Delta(\mathfrak{q}(u), f)$ if this transition is defined, and undefined otherwise.

Basically, if $\mathfrak{q}(u)$ is defined, then it gives the unique label of the node corresponding to u in a run of $\mathcal{A}$ on a tree containing the path encoded by u.

Now define a system of language equations (14) over the alphabet $\Sigma_\Gamma \cup Q$, which simulates the computation of the automaton $\mathcal{A}$. The set of variables of this equation is $\{X_{q,f} \mid \Delta(q, f) \text{ is defined}\} \cup \{X_0\}$, and the system consists of the two equations

$$\bigcup_{\Delta(q,f) \text{ is defined}} X_{q,f} \cdot \{q\} = \{q_0\} \cup \bigcup_{\Delta(q,f)=(q_1,\ldots,q_k)} X_{q,f} \cdot \{f^{[1]}q_1, \ldots, f^{[k]}q_k\}, \tag{14a}$$

$$X_0 = \bigcup_{\Delta(q,f) \text{ is defined}} X_{q,f}. \tag{14b}$$

The following lemma establishes some basic properties of solutions of this system.

Lemma 15. Let $(\dots, L_{q,f}, \dots, L_0)$ be any solution of (14). Then, for all $w \in \Sigma_\Gamma^*$, $q \in Q$ and $f \in \Gamma$,

I. $w \in L_{q,f}$ if and only if $\mathfrak{q}(w) = q$ and $wf^{[i]} \in L_0$ for all $i \in \{1, \dots, \operatorname{rank} f\}$.

II. If $w \in L_{q,f}$ for some q, then there exists an infinite tree t, such that $\{wf^{[1]}, \dots, wf^{[\operatorname{rank} f]}\} \subseteq S(t) \subseteq L_0$.

Proof:
Denote by $L \subseteq \Sigma_\Gamma^* Q$ the common value of the left-hand side and the right-hand side of (14a) under the substitution $X_{q,f} = L_{q,f}$.

Part I. Let us first show by induction on the length of w that $w \in L_{q,f}$ implies that $\mathfrak{q}(w)$ is defined and equals q.

Base case. If $\varepsilon \in L_{q,f}$, then, by the left-hand side of the equation, $q \in L$. According to the right-hand side, this implies $q = q_0 = \mathfrak{q}(\varepsilon)$.

Induction step. Let $w = uh^{[i]}$, where $u \in \Sigma_\Gamma^*$ and $h^{[i]} \in \Sigma_\Gamma$. Since $uh^{[i]} \in L_{q,f}$, we obtain $uh^{[i]}q \in L$ by the left-hand side. Therefore, by the right-hand side, there exists a state $p \in Q$ such that $u \in L_{p,h}$, $\Delta(p,h) = (q_1, \dots, q_k)$ and $q = q_i$. By the induction hypothesis, $u \in L_{p,h}$ implies $\mathfrak{q}(u) = p$. Combining this with the value of $\Delta(p,h)$, we obtain $\mathfrak{q}(uh^{[i]}) = q_i = q$.

Now let us demonstrate that $w \in L_{q,f}$ implies $wf^{[i]} \in L_0$ for all i. Since the variable $X_{q,f}$ exists, the transition $\Delta(q,f)$ is defined, and thus yields a tuple $(q_1, \dots, q_k) \in Q^k$, where $k = \operatorname{rank} f$. By the right-hand side of the equation, $wf^{[i]}q_i \in L$, and therefore, by the left-hand side, there exists a symbol $g \in \Gamma$, such that $wf^{[i]} \in L_{q_i,g} \subseteq L_0$.

It remains to prove the converse claim that $wf^{[i]} \in L_0$ implies $w \in L_{\mathfrak{q}(w),f}$. If $wf^{[i]} \in L_0$, then $wf^{[i]} \in L_{p,h}$ for some $p \in Q$ and $h \in \Gamma$ such that $\Delta(p,h)$ is defined. Hence, by the left-hand side, $wf^{[i]}p \in L$. By the right-hand side, there exists a state $q \in Q$, such that $\Delta(q,f) = (q_1, \dots, q_k)$, where $p = q_i$ and $w \in L_{q,f}$. As shown in the proof of the "only-if" direction, $w \in L_{q,f}$ implies $\mathfrak{q}(w) = q$, and thus we have $w \in L_{\mathfrak{q}(w),f}$. This concludes the proof of Part I of the lemma.

Part II. Let $w \in L_{q,f}$ for some $q \in Q$. We construct a tree t by induction on the length of paths, while satisfying as an *invariant* that all paths in t are represented by strings in L_0.

Base case. The empty path is represented by ε. The empty word ε is in L_0 since the right-hand side of the equation yields $q_0 \in L$, and thus there must be a symbol f such that $\Delta(q_0,f)$ is defined and $\varepsilon \in L_{q_0,f} \subseteq L_0$.

Induction step. Let the tree contain a finite path represented by $x \in \Sigma_\Gamma^*$. By the induction hypothesis, $x \in L_0$, and hence $x \in L_{p,h}$ for some $p \in Q, h \in \Gamma$. By Part I, $p = \mathfrak{q}(x)$. The symbol h is defined not necessarily uniquely. If wf has a prefix xg, then $xg \in L_0$. In fact, one can use the implication from right to left of Part I to show that any prefix of an element of L_0 also belongs to L_0. But then $xg \in L_0$ and $p = \mathfrak{q}(x)$ yield $x \in L_{p,g}$ by Part I. Hence, we can choose $h = g$. Otherwise the choice of h is arbitrary.

We continue the path represented by x with a vertex with label h. This yields $\operatorname{rank} h$ longer strings of the form $xh^{[j]}$ ($1 \leqslant j \leqslant \operatorname{rank} h$). Since $x \in L_{p,h}$, by Part I, $xh^{[i]} \in L_0$, which shows that the invariant of the construction remains true. □

Based on this lemma and the properties of the mapping S mentioned above, we can show the following characterization of the solutions of (14). In particular, this characterization shows that the value L_0 of the variable X_0 determines the whole solution $(\dots, L_{q,f}, \dots, L_0)$.

Lemma 16. A vector of languages $(\dots, L_{q,f}, \dots, L_0)$ is a solution of (14) if and only if

$$\varnothing \subset S^{-1}(L_0) \subseteq L(\mathcal{A}), \tag{15}$$

$$L_{q,f} = \{w \mid \mathfrak{q}(w) = q,\ wf^{[i]} \in L_0 \text{ for all } i\} \quad (q \in Q, f \in \Gamma), \tag{16}$$

and there exists a set of trees T, such that $L_0 = S(T)$.

Proof:
First, we show the *"only-if" direction*. Consider an arbitrary solution $(\dots, L_{q,f}, \dots, L_0)$ of (14). For every $w \in L_0$, let T_w be the set of all trees t such that $w \in S(t) \subseteq L_0$. According to Part II of Lemma 15, there exists at least one such tree, and thus we obtain

$$\{w\} \subseteq S(T_w) \subseteq L_0.$$

Summing up these inequalities for all $w \in L_0$, we obtain

$$\begin{array}{ccccc} \bigcup_{w\in L_0}\{w\} & \subseteq & \bigcup_{w\in L_0} S(T_w) & \subseteq & \bigcup_{w\in L_0} L_0 \\ \| & & \| & & \| \\ L_0 & \subseteq & S(\bigcup_{w\in L_0} T_w) & \subseteq & L_0, \end{array}$$

which shows that $L_0 = S(\bigcup_{w\in L} T_w)$. Thus, if we define $T := \bigcup_{w\in L_0} T_w$, then the last assertion in the statement of the lemma is satisfied. It remains to show that the conditions (15–16) hold as well.

- To see that $S^{-1}(L_0) \neq \varnothing$, note that $q_0 \in \bigcup_{q,f} L_{q,f} \cdot \{q\}$ by (14a), and hence $\varepsilon \in \bigcup_f L_{q_0,f} \subseteq L_0$. The tree t_ε associated with ε by Part II of Lemma 15 is in $T_\varepsilon \subseteq T$, and hence, by Lemma 14, $t_\varepsilon \in S^{-1}(S(T)) = S^{-1}(L_0)$.
- Next, we prove the containment $S^{-1}(L_0) \subseteq L(\mathcal{A})$. Suppose there exists a tree $t \in S^{-1}(L_0)$ (that is, $S(t) \subseteq L_0$), such that $t \notin L(\mathcal{A})$. Because t is not accepted by the looping automaton $\mathcal{A}$, there is no run of $\mathcal{A}$ on t, that is, when trying to construct the (unique) run of $\mathcal{A}$ on t, starting from the root, we encounter a node in which there are no possible transitions. By the definition of the function $\mathfrak{q}$, this means that there is a string $w \in S(t)$, on which $\mathfrak{q}(w)$ is undefined. However, we have $w \in S(t) \subseteq L_0$, and thus there exists a pair (q, f) with $w \in L_{q,f}$. But then, according to Part I of Lemma 15, $\mathfrak{q}(w) = q$, which yields a contradiction.
- The condition (16) is given by Part I of Lemma 15.

To show the *"if" direction*, let us start by considering the case where $L_0 = S(t)$ for a tree $t \in L(\mathcal{A})$. The first claim is that substituting

$$L^t_{q,f} := \{w \mid \mathfrak{q}(w) = q,\ wf^{[i]} \in S(t) \text{ for all } i\} \tag{17}$$

for $X_{q,f}$ (for $q \in Q$ and $f \in \Gamma$ with $\Delta(q, f)$ defined) turns (14a) into an equality.

The value of the left-hand side of (14a) under this substitution is

$$\bigcup_{\Delta(q,f) \text{ is defined}} L^t_{q,f} \cdot \{q\} = \{w \cdot \mathfrak{q}(w) \mid \exists f.\forall i.\ wf^{[i]} \in S(t)\} = \{w \cdot \mathfrak{q}(w) \mid w \in S(t)\}.$$

The latter equality follows from Lemma 13: if $wf^{[i]} \in S(t)$, then $w \in S(t)$ by Part IV of that lemma; conversely, if $w \in S(t)$, then $wf^{[1]} \in S(t)$ for some f by Part II, and then $wf^{[i]} \in S(t)$ for all i by Part III.

The right-hand side of (14a) looks as follows under this substitution:

$$\begin{aligned}
&\{q_0\} \cup \textstyle\bigcup_{\Delta(q,f)=(q_1,\ldots,q_k)} L^t_{q,f} \cdot \{f^{[1]}q_1, \ldots, f^{[k]}q_k\} = \\
&= \varepsilon \cdot \mathfrak{q}(\varepsilon) \cup \{uf^{[i]} \cdot \mathfrak{q}(uf^{[i]}) \mid \forall i.\ uf^{[i]} \in S(t)\} = \\
&= \{w \cdot \mathfrak{q}(w) \mid w \in S(t)\}.
\end{aligned}$$

This proves that the substitution that replaces $X^t_{q,f}$ by $L^t_{q,f}$ satisfies the equation (14a). In order to show that the equation (14b) is satisfied as well if we replace X_0 by $L_0 = S(t)$, we must prove that $S(t) = \bigcup_{q,f} L^t_{q,f}$:

"$\subseteq$" If $w \in S(t)$, then $\mathfrak{q}(w)$ is defined, since $t \in L(\mathcal{A})$. By Lemma 13 (Parts II and III), there exists a symbol $f \in \Gamma$ with $wf^{[i]} \in S(t)$ for all $i \in \{1, \ldots, \operatorname{rank} f\}$. Therefore, $w \in L^t_{\mathfrak{q}(w),f}$.

"$\supseteq$" If $w \in L^t_{q,f}$ for some pair (q, f), then $wf^{[1]} \in S(t)$. Hence, by Part IV of Lemma 13, we have $w \in S(t)$.

This completes the proof of the "if" direction for the case where $T = \{t\}$ for a tree $t \in L(\mathcal{A})$.

Now, let $L_0 = S(T)$ for an arbitrary set of trees T such that (15) holds. Then $T \subseteq S^{-1}(S(T)) = S^{-1}(L_0) \subseteq L(\mathcal{A})$, where the first inclusion holds by Lemma 14 and the second by the assumption (15), and therefore $T \subseteq L(\mathcal{A})$.

For every $t \in T \subseteq L(\mathcal{A})$, consider the vector of languages $(\ldots, L^t_{q,f}, \ldots, L^t_0)$ corresponding to t, defined by (17) and by $L^t_0 := S(t)$. We have shown above that this vector is a solution of the system (14). Consider the componentwise union of these vectors for all $t \in T$, that is, the vector $(\ldots, L_{q,f}, \ldots, L_0)$ defined as $L_{q,f} := \bigcup_{t \in T} L^t_{q,f}$ and $L_0 := \bigcup_{t \in T} L^t_0 = S(T)$. As a union of solutions, it is a solution as well.[5]

It remains to show that the components $L_{q,f}$ defined above are indeed the ones induced by $L_0 = S(T)$ according to (16):

$$\begin{aligned}
L_{q,f} &= \bigcup_{t \in T} \{w \mid \mathfrak{q}(w) = q,\ wf^{[i]} \in S(t) \text{ for all } i\} \\
&= \{w \mid \mathfrak{q}(w) = q,\ wf^{[i]} \in S(T) \text{ for all } i\}.
\end{aligned}$$

This completes the proof of the "if" direction. □

4.3. Complexity of the decision problems

The next theorem summarizes the main results of this paper.

Theorem 7. The problems of testing, for a given system of language equations with one-sided concatenation and any set of Boolean operations containing union, whether

[5]Note that the system (14) is a system of language equations with one-sided concatenation *and union*, for which this property is well-known [2].

1. it has a solution,
2. it has a unique solution,
3. it has finitely many solutions,
4. it has countably many solutions,
5. it has a least (greatest) solution with respect to componentwise inclusion

are all EXPTIME-complete.

Given the results shown in Section 3 and in [1, 2], it is enough to prove that testing whether a language equation with one-sided concatenation *and union* has a unique solution, finitely many solutions, countably many solutions, and a least solution, respectively, are EXPTIME-hard problems.

All four cases are proved by a single reduction from the EXPTIME-complete intersection emptiness problem for deterministic looping tree automata [35, 2]. Let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be deterministic looping tree automata over a common ranked alphabet Γ, and assume without loss of generality that their sets of states $Q_1, \ldots, Q_n$ are pairwise disjoint and that the initial state $q_0^{(i)}$ of every $\mathcal{A}_i$ is not reenterable, that is, it never occurs on the right-hand side of a transition.

We augment Γ with a new unary symbol f_{triv}, and transform each automaton $\mathcal{A}_i$ into an automaton $\mathcal{A}'_i$ over the alphabet $\Gamma' = \Gamma \cup \{f_{triv}\}$ by adding the extra transition $(q_0^{(i)}, f_{triv}) \to q_0^{(i)}$. The set of trees accepted by $\mathcal{A}'_i$ equals $\{f_{triv}^{\ell}(t)) \mid \ell \geqslant 0, t \in L(\mathcal{A}_i)\} \cup \{t_{triv}\}$, where t_{triv} denotes an infinite branch with all vertices labeled by f_{triv}. Consequently, the intersection $\bigcap_{i=1}^{n} L(\mathcal{A}'_i)$ is equal to $\{f_{triv}^{\ell}(t)) \mid \ell \geqslant 0, t \in \bigcap_{i=1}^{n} L(\mathcal{A}_i)\} \cup \{t_{triv}\}$.

For each automaton $\mathcal{A}'_i$, construct two language equations of the form (14), and consider the resulting system of $2n$ equations, which share a common variable X_0. It is easy to show that the vector of languages $L_{triv} := (\ldots, L_{q,f}^{(i)}, \ldots, L_0)$ defined by

$$L_0 := S(t_{triv}) \text{ and}$$
$$L_{q,f}^{(i)} \text{ determined by } L_0 \text{ and } \mathcal{A}'_i \text{ according to (16) in Lemma 16}$$

is always a solution of the system. In fact, $S(t_{triv}) = (f_{triv}^{[1]})^*$, and therefore $S^{-1}(L_0) = S^{-1}((f_{triv}^{[1]})^*) = \{t_{triv}\}$, which is a subset of $L(\mathcal{A}'_i)$ for all i. Thus, the condition (15) in Lemma 16 is also satisfied, and n applications of that lemma show that the constructed vector satisfies each pair of equations, and is therefore a solution of the whole system.

Whether the system has any other solutions depends on whether $\bigcap_{i=1}^{n} L(\mathcal{A}_i)$ is empty or not.

Lemma 17. If $\bigcap_{i=1}^{n} L(\mathcal{A}_i) = \varnothing$, then the system of language equations has a unique solution.

Proof:
If $\bigcap_{i=1}^{n} L(\mathcal{A}_i) = \varnothing$, then $\bigcap_{i=1}^{n} L(\mathcal{A}'_i) = \{t_{triv}\}$. We prove that in this case the system has the unique solution L_{triv}.

Consider any solution $(\ldots, L^{(i)}_{q,f}, \ldots, L_0)$, and let us apply Lemma 16 to the i-th pair of equations. We obtain:

$$\varnothing \subset S^{-1}(L_0) \subseteq L(\mathcal{A}'_i) \quad (1 \leqslant i \leqslant n), \tag{18a}$$

$$L_0 = S(T_i) \quad \text{(for some set of trees } T_i) \tag{18b}$$

and that all the languages $L^{[i]}_{q,f}$ are completely determined by L_0.

Intersecting (18a) for all i yields $S^{-1}(L_0) \subseteq \bigcap_{i=1}^{n} L(\mathcal{A}'_i)$, where the latter equals $\{t_{triv}\}$ by assumption. The inclusions $\varnothing \subset S^{-1}(L_0) \subseteq \{t_{triv}\}$ imply

$$S^{-1}(L_0) = \{t_{triv}\}.$$

Application of S to both sides yields

$$S(S^{-1}(L_0)) = S(t_{triv}).$$

Recalling that $L_0 = S(T_i)$ for some set of trees T_i (where $i \in \{1, \ldots, n\}$ is arbitrary), we obtain

$$S(S^{-1}(S(T_i))) = S(t_{triv}).$$

The left-hand side of the last equality equals $S(T_i)$ by Lemma 14, and hence we have

$$L_0 = S(T_i) = S(t_{triv}).$$

Therefore, L_0 is uniquely determined, and since the rest of the components of the solution are in turn completely determined by L_0, the solution is unique. □

Lemma 18. If $\bigcap_{i=1}^{n} L(\mathcal{A}_i) \neq \varnothing$, then the system of language equations has uncountably many solutions.

Proof:
If $\bigcap_{i=1}^{n} L(\mathcal{A}_i) \neq \varnothing$, then there exists a tree $t_0 \in \bigcap_{i=1}^{n} L(\mathcal{A}_i)$, and $f^{\ell}_{triv}(t_0) \in \bigcap_{i=1}^{n} L(\mathcal{A}'_i)$ for all $\ell \geqslant 0$. We construct uncountably many solutions of the system as follows.

For every non-empty set of integers $\varnothing \subset N \subseteq \mathbb{N}$, define the set of trees

$$T_N = \{f^{\ell}_{triv}(t_0) \mid \ell \in N\}. \tag{19}$$

Note that T_N is a subset of $\bigcap_{i=1}^{n} L(\mathcal{A}'_i)$. We prove that the vector of languages $(\ldots, L^{(i)}_{q,f,N}, \ldots, L_{0,N})$ determined by $L_{0,N} := S(T_N)$ according to (16) is a solution of the system, and that different sets N yield different solutions.

First, let us show that this vector satisfies the conditions of Lemma 16 for every i-th pair of equations constructed with respect to $\mathcal{A}'_i$, and hence is a solution of the system:

- $S^{-1}(L_{0,N}) \neq \varnothing$ since $\exists \ell \in N$, and hence $f^{\ell}_{triv}(t_0) \in T_N \subseteq S^{-1}(S(T_N)) = S^{-1}(L_{0,N})$, where the inclusion is by Lemma 14.

- Let us prove that $S^{-1}(L_{0,N}) \subseteq L(\mathcal{A}'_i)$. Consider any tree $t \in S^{-1}(L_{0,N})$ and let us consider its starting chain (possibly empty) of nodes labeled f_{triv}. There are two cases: either the chain of f_{triv} is infinite, in which case $t = t_{triv} \in L(\mathcal{A}'_i)$ and the claim is proved, or the tree can be represented in the form $f^{\ell}_{triv}(t')$, where $\ell \geqslant 0$ and the root node of t' is not f_{triv}.

  By definition, $t \in S^{-1}(L_{0,N})$ implies $S(t) \subseteq L_{0,N} = S(T_N)$. On the other hand, $S(t) = (f^{[1]}_{triv})^{\ell} \cdot S(t')$, where none of the strings in $S(t')$ begins with $f^{[1]}_{triv}$. Then, by (19), all strings in $S(t)$ must be in $(f^{[1]}_{triv})^{\ell} \cdot S(t_0) = S(f^{\ell}_{triv}(t_0))$, and therefore $S(t') \subseteq S(t_0)$. We claim that $t' = t_0$.

  Suppose to the contrary that $t' \neq t_0$. Then there is a (possibly empty) common finite path in t' and t_0, encoded as $w = f^{[i_1]}_1 \dots f^{[i_k]}_k$, which is extended with a node labeled g in t', and with a node labeled $h \neq g$ in t_0. Then $wg^{[1]}$ must be in $S(t') \subseteq S(t_0)$. Thus we obtain $wg^{[1]} \in S(t_0)$, which means that the path w in t_0 is extended with both a node labeled g and a node labeled $h \neq g$, which contradicts Part II of Lemma 13.

  Now, $t' = t_0$ implies $t = f^{\ell}_{triv}(t') = f^{\ell}_{triv}(t_0) \in L(\mathcal{A}'_i)$.

The second claim is that solutions corresponding to different sets of integers are different. It has to be proved that, for any sets $N \neq N'$, $S(T_N) \neq S(T_{N'})$. Let $w \in S(t_0) \setminus \{\varepsilon\}$. Consider any number ℓ in the symmetric difference of N and N', and suppose without loss of generality that $\ell \in N$ and $\ell \notin N'$. Then $(f^{[1]}_{triv})^{\ell} w \in S(T_N) \setminus S(T_{N'})$. Thus we have constructed uncountably many pairwise distinct solutions of the system. □

Since the constructed system of language equations has either exactly one solution or uncountably many solutions, we can conclude that it has a unique solution (finitely many solutions, countably many solutions) if and only if the intersection of the languages generated by the n given deterministic looping tree automata is empty. This proves that the problem of deciding whether a system of language equations with one-sided concatenation and union has a unique solution (finitely many solutions, countably many solutions) is EXPTIME-hard. It remains to consider the case of a least solution.

Lemma 19. If $\bigcap_{i=1}^{n} L(\mathcal{A}_i) \neq \varnothing$, then the system of language equations does not have a least solution.

Proof:
Consider the two solutions induced by $L_0 := S(t_{triv})$ and $L'_0 := S(T_{\{0\}}) = S(t_0)$, where t_0 and $T_{\{0\}}$ are defined as in the proof of the previous lemma. If the system has a least solution, then its X_0-component must be a subset of both L_0 and L'_0, that is, less or equal to

$$\underbrace{S(t_{triv})}_{=(f^{[1]}_{triv})^*} \cap \underbrace{S(t)}_{\subseteq \Sigma^*_\Gamma} = \{\varepsilon\}.$$

However, according to Lemma 16, this component must be of the form $S(T)$ for some non-empty set of trees T, and thus has to be infinite, which yields a contradiction. Therefore, no solution of the system can be less than both solutions given above, which shows that there is no least solution among the solutions of the system. □

Since the constructed system of language equations has either exactly one solution (and thus a least solution) or no least solution, we can conclude that it has a least solution if and only if the intersection of the languages generated by the n given deterministic looping tree automata is empty. This proves that the problem of deciding whether a system of language equations with one-sided concatenation and union has a least solution is EXPTIME-hard. This completes the proof of Theorem 7.

5. Conclusion

We have shown that several interesting decision problems for language equations with one-sided concatenation are EXPTIME-complete: solvability, existence of a unique (least, greatest) solution, and determining the cardinality (finite, countable, uncountable) of the set of solutions. The complexity upper-bounds are shown for all decision problems by a uniform translation into a looping tree automaton with independent transitions, i.e., a non-deterministic finite automaton that is viewed as a looping tree automaton. Accordingly, the complexity lower-bounds are shown by a uniform reduction from the intersection emptiness problem for deterministic looping tree automata. Though the translation of deterministic looping tree automata into language equations is identical to the one given in [2], we believe that the proof of correctness of the reduction is simpler and much easier to comprehend than the one given there. In addition, our translation is also used to show EXPTIME-hardness of decision problems other than solvability.

The decision procedures based on the construction of an ILTA have been implemented. This implementation does not just answer yes or no. In case there is a unique (least, greatest) solution, we know that its components are regular languages, and the implementation constructs deterministic finite automata for these components (see Section 3.4).

In [4] we have shown that the bijection between solutions of a language equation $\varphi(X_1, \dots, X_n) = \varnothing$ with one-sided concatenation and runs of the corresponding trim ILTA A_φ (Proposition 2) can also be used to solve disequations, i.e., systems consisting of one equation $\varphi(X_1, \dots, X_n) = \varnothing$ and finitely many inequations of the form $\psi(X_1, \dots, X_n) \neq \varnothing$. However, to take care of the inequations, a new type of automata working on infinite trees, called looping automata *with colors*, had to be introduced.

Acknowledgment

We thank Thomas Wilke for alerting us to the work of Niwiński, and Moshe Vardi for suggesting the name “looping tree automata with *independent* transitions”.

References

[1] A. Aiken, D. Kozen, M. Y. Vardi, E. L. Wimmers, “The complexity of set constraints”, *Computer Science Logic* (CSL 1993, Swansea, UK, September 13–17, 1993), LNCS 832, 1–17.

[2] F. Baader, R. Küsters, “Unification in a description logic with transitive closure of roles”, *Logic for Programming, Artificial Intelligence, and Reasoning* (LPAR 2001, Havana, Cuba, December 3–7, 2001), LNCS 2250, 217–232.

[3] F. Baader, P. Narendran, “Unification of concept terms in description logic”, *Journal of Symbolic Computation*, 31 (2001), 277–305.

[4] F. Baader, A. Okhotin, “Solving language equations and disequations with applications to disunification in description logics and monadic set constraints”, *Logic for Programming, Artificial Intelligence, and Reasoning* (LPAR 2012, Mérida, Venezuela, March 11–15, 2012), LNCS 7180, 107121.

[5] F. Baader and W. Snyder. Unification theory. In *Handbook of Automated Reasoning*, volume I. Elsevier, 2001.

[6] F. Baader, S. Tobies, “The inverse method implements the automata approach for modal satisfiability”, In *Proc. IJCAR’01*, Springer LNCS 2083, 2001.

[7] S. Bala, “Complexity of regular language matching and other decidable cases of the satisfiability problem for constraints between regular open terms”, *Theory of Computing Systems*, 39:1 (2006), 137–163.

[8] V. G. Bondarchuk, “Sistemy uravnenii v algebre sobytii” (Systems of equations in the event algebra), in Russian, *Zhurnal vychislitel’noi matematiki i matematicheskoi fiziki* (Journal of Computational Mathematics and Mathematical Physics), 3:6 (1963), 1077–1088.

[9] W. Büttner. Unification in finite algebras is unitary(?). In *Proc. CADE-9*, Springer LNCS 310, 1988.

[10] W. Charatonik, “Set constraints in some equational theories”, *Information and Computation*, 142 (1998), 40–75.

[11] W. Charatonik, L. Pacholski, “Set constraints with projections”, *Journal of the ACM*, 57:4 (2010), article 23.

[12] J. H. Conway, *Regular Algebra and Finite Machines*, Chapman and Hall, 1971.

[13] A. E. Frid, “Simple equations on binary factorial languages”, *Theoretical Computer Science*, 410:30–32 (2009), 2947–2956.

[14] S. Ginsburg, H. G. Rice, “Two families of languages related to ALGOL”, *Journal of the ACM*, 9 (1962), 350–371.

[15] R. Gilleron, S. Tison, M. Tommasi, “Set constraints and automata”, *Information and Computation*, 149:1 (1999), 1–41.

[16] A. Jeż, A. Okhotin, “On the computational completeness of equations over sets of natural numbers”, *Automata, Languages and Programming* (ICALP 2008, Reykjavík, Iceland, July 6–13, 2008), part II, LNCS 5126, 63–74.

[17] A. Jeż, A. Okhotin, “Equations over sets of natural numbers with addition only”, *STACS 2009* (Freiburg, Germany, 26–28 February, 2009), 577–588.

[18] M. Kunc, “The power of commuting with finite sets of words”, *Theory of Computing Systems*, 40:4 (2007), 521–551.

[19] M. Kunc, “What do we know about language equations?”, *Developments in Language Theory* (DLT 2007, Turku, Finland, July 3–6, 2007), LNCS 4588, 23–27.

[20] T. Lehtinen, A. Okhotin, “On language equations $XXK = XXL$ and $XM = N$ over a unary alphabet”, *Developments in Language Theory* (DLT 2010, London, Ontario, Canada, 17–20 August 2010), LNCS 6224, 291–302.

[21] G. S. Makanin, The problem of solvability of equations in a free semigroup. *Math. Sbornik* 103:147–236. English translation in Math. USSR Sbornik 32, 1977.

[22] W. Martens, M. Niewerth, T. Schwentick, “Schema design for XML repositories: complexity and tractability”, *Symposium on Principles of Database Systems* (PODS 2010, Indianapolis, USA, June 6–11, 2010), 239–250.

[23] T. Nipkow. Unification in primal algebras, their powers and their varieties. *J. of the ACM*, 37(1):742–776, 1990.

[24] D. Niwiński, "On the cardinality of sets of infinite trees recognizable by finite automata", *Mathematical Foundations of Computer Science* (MFCS 1991, Kazimierz Dolny, Poland, September 9–13, 1991), LNCS 520, 1991, 367–376.

[25] A. Okhotin, "Conjunctive grammars and systems of language equations", *Programming and Computer Software*, 28:5 (2002), 243–249.

[26] A. Okhotin, "Decision problems for language equations with Boolean operations", *Automata, Languages and Programming* (ICALP 2003, Eindhoven, The Netherlands, 30 June–4 July 2003), LNCS 2719, 239–251.

[27] A. Okhotin, "Unresolved systems of language equations: expressive power and decision problems", *Theoretical Computer Science*, 349:3 (2005), 283–308.

[28] A. Okhotin, "Strict language inequalities and their decision problems", *Mathematical Foundations of Computer Science* (MFCS 2005, Gdańsk, Poland, August 29–September 2, 2005), LNCS 3618, 708–719.

[29] A. Okhotin, "Decision problems for language equations", *Journal of Computer and System Sciences*, 76:3–4 (2010), 251–266.

[30] A. Okhotin, P. Rondogiannis, "On the expressive power of univariate equations over sets of natural numbers", *Information and Computation*, 212 (2012), 1–14.

[31] A. Okhotin, O. Yakimova, "Language equations with complementation: Decision problems", *Theoretical Computer Science*, 376:1–2 (2007), 112–126.

[32] R. Parikh, A. Chandra, J. Halpern, A. Meyer, "Equations between regular terms and an application to process logic", *SIAM Journal on Computing*, 14:4 (1985), 935–942.

[33] M. O. Rabin, "Decidability of second-order theories and automata on infinite trees", *Transactions of the American Mathematical Society*, 141 (1969), 1–35.

[34] A. Salomaa, *Theory of Automata*, Pergamon Press, Oxford, 1969.

[35] H. Seidl, "Haskell overloading is DEXPTIME-complete", *Information Processing Letters*, 52(2):57–60, 1994.

[36] L. R. Stockmeyer, *The Complexity of Decision Problems in Automata Theory and Logic*, Ph.D. thesis, Dept. of Electrical Engineering, MIT, 1974.

[37] A. Tarski, "A lattice-theoretical fixpoint theorem and its applications", *Pacific Journal of Mathematics*, 5 (1955), 285–309.

[38] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 133–191. Elsevier Science Publishers, Amsterdam, 1990.

[39] M. Y. Vardi and P. Wolper, "Automata-theoretic techniques for modal logics of programs", *Journal of Computer and System Sciences*, 32 (1986), 183–221.