Dresden University of Technology
Institute for Theoretical Computer Science
Chair for Automata Theory

LTCS–Report

Solving Language Equations and Disequations Using Looping Tree Automata with Colors

Franz Baader Alexander Okhotin

LTCS-Report 12-01

A short version of this report has also appeared in
Proceedings of LPAR-18, Springer LNCS 7180, 2012.

Postal Address:
Lehrstuhl für Automatentheorie
Institut für Theoretische Informatik
TU Dresden
01062 Dresden

http://lat.inf.tu-dresden.de

Visiting Address:
Nöthnitzer Str. 46
Dresden

Solving Language Equations and Disequations Using Looping Tree Automata with Colors

Franz Baader*
Institute for Theoretical Computer Science,
TU Dresden, D-01062 Dresden, Germany
baader@tcs.inf.tu-dresden.de

Alexander Okhotin†
Department of Mathematics
University of Turku, FIN-20014 Turku, Finland
alexander.okhotin@utu.fi

January 30, 2012

Abstract

We extend previous results on the complexity of solving language equations with one-sided concatenation and all Boolean operations to the case where also disequations (i.e., negated equations) may occur. To show that solvability of systems of equations and disequations is still in ExpTime, we introduce a new type of automata working on infinite trees, which we call looping automata with colors. As applications of these results, we show new complexity results for disunification in the description logic $\mathcal{FL}_0$ and for monadic set constraints with negation. We believe that looping automata with colors may also turn out to be useful in other applications.

*Supported by DFG under grant BA 1122/14-1.
†Supported by the Academy of Finland under grant 134860.

Contents

1 Introduction

Equations with formal languages as constant parameters and unknowns are among the basic notions of *formal language theory*, first introduced by Ginsburg and Rice [8], who gave a characterization of the context-free languages by solutions of systems of equations of the resolved form $X_i = \varphi_i(X_1, \ldots, X_n)$. For equations of the general form $\varphi(X_1, \ldots, X_n) = \psi(X_1, \ldots, X_n)$ built using union and two-sided concatenation, testing their solvability is easily shown to be undecidable [14]. The state-of-the-art in this area as of 2007 is presented in a survey by Kunc [10]. More recent work shows that undecidability already holds for equations over a one-letter alphabet with concatenation as the only operation [9, 11]. In contrast, solvability of language equations with concatenation restricted to *one-sided concatenation with constants* can often be shown to be decidable by encoding the problem into monadic second-order logic on infinite trees (MSO) [15], but this usually does not yield optimal complexity results.

In *logic for programming and artificial intelligence*, language equations with one-sided concatenation are, for instance, relevant in the context of monadic set constraints and unification in description logics (DLs). *Unification in DLs* has been proposed [4] as a novel inference service that can, for example, be used to detect redundancies in ontologies. As a simple example, assume that one knowledge engineer has defined the concept of "women having only daughters" by the concept term $\mathsf{Woman} \sqcap \forall \mathsf{child}.\mathsf{Woman}$. A second knowledge engineer might represent this notion in a somewhat more fine-grained way, e.g., by using the term $\mathsf{Female} \sqcap \mathsf{Human}$ in place of Woman. The concept terms $\mathsf{Woman} \sqcap \forall \mathsf{child}.\mathsf{Woman}$ and $\mathsf{Female} \sqcap \mathsf{Human} \sqcap \forall \mathsf{child}.(\mathsf{Female} \sqcap \mathsf{Human})$ are not equivalent, but they are meant to represent the same concept. The two terms can obviously be made equivalent by viewing the concept name Woman as a concept variable and replacing it in the first term by the concept term $\mathsf{Female} \sqcap \mathsf{Human}$. Unification in DLs checks for the existence of such substitutions, and thus can be used to alert the knowledge engineers to potential redundancies in the ontology. In [4] it was shown that unification in the DL $\mathcal{FL}_0$ can be reduced to *finite solvability*[1] of language equations with one-sided concatenation and union, and that this problem is in turn ExpTime-complete. In [3] it was shown that the same complexity result holds for *solvability*,[2] and in [5] this result was extended to language equations with one-sided concatenation and *all Boolean operations*, and to *other decision problems than just solvability*.

Language equations with one-sided concatenation and all Boolean operations can also be regarded as a particular case of equations on sets of terms, known as *set constraints*, which received significant attention [13] in logic for programming since they can be used in program analysis. In fact, solvability of such language

[1] i.e., existence of a solution consisting of finite languages.

[2] i.e., existence of a solution consisting of arbitrary (not necessarily finite) languages.

equations corresponds to solvability of *monadic* set constraints, where all function symbols are at most unary. In [1] it was already shown that solvability of monadic set constraints is an ExpTime-complete problem.

In the present paper, we extend the existing results for language equations with one-sided concatenation and all Boolean operations to the case of finite systems of language equations and *disequations* (i.e., negated equations). We will show that solvability and finite solvability of such systems are still *in ExpTime*. The motivation comes again from description logics and from set constraints. Set constraints with negation have been investigated in several papers [7, 16, 2], where it is shown that solvability in the general case is NExpTime-complete. The exact complexity of the monadic case has, to the best of our knowledge, not been determined yet. In description logics, it makes sense to consider not only unification, but also disunification problems in order to prevent certain unifiers. For example the concept term $\mathsf{Woman} \sqcap \forall \mathsf{child}.\mathsf{Woman}$ also unifies with $\mathsf{Male} \sqcap \mathsf{Human} \sqcap \forall \mathsf{child}.(\mathsf{Male} \sqcap \mathsf{Human})$, which could, e.g., be prevented by stating that Woman should not become a subconcept of Male, i.e., that $\mathsf{Woman} \sqcap \mathsf{Male}$ must not be unified with Woman.

In Section 2, we formally define language equations and disequations with one-sided concatenation and all Boolean operations, and show that their (finite) solvability can be reduced to the existence of certain runs of a corresponding looping tree automaton. In Section 3, we introduce looping tree automata with colors, which can express the condition on the runs formulated in the previous section, and then analyze the complexity of their emptiness problem. Finally, in Section 4 we use these results to determine the complexity of testing (finite) solvability of the systems of language (dis)equations introduced in Section 2, and then in turn apply this result to identify the complexity of solving disunification problems in $\mathcal{FL}_0$ as well as monadic set constraints with negation.

2 Language (dis)equations with one-sided concatenation

In this section, we first introduce the language (dis)equations that we want to solve, and then we show how solvability can be reduced to a problem for looping automata working on infinite trees.

2.1 The problem definition

Given a finite alphabet Σ and finitely many variables $X_1, \ldots, X_n$, the set of *language expressions* is defined by induction:

- any variable X_i is a language expression;
- the empty word ε is a language expression;
- a concatenation φa of a language expression φ with a symbol $a \in \Sigma$ is a language expression;[3]
- if φ, φ' are language expressions, then so are $(\varphi \cup \varphi')$, $(\varphi \cap \varphi')$ and $(\sim\varphi)$.

Given a mapping $\theta = \{X_1 \mapsto L_1, \ldots, X_n \mapsto L_n\}$ of the variables to languages $L_1, \ldots, L_n$ over Σ, its extension to language expressions is defined as

- $\theta(X_i) := L_i$ for all $i, 1 \leq i \leq n$;
- $\theta(\varepsilon) := \{\varepsilon\}$;
- $\theta(\varphi a) := \theta(\varphi) \cdot \{a\}$ for $a \in \Sigma$;
- $\theta(\varphi \cup \varphi') := \theta(\varphi) \cup \theta(\varphi')$, $\theta(\varphi \cap \varphi') := \theta(\varphi) \cap \theta(\varphi')$, and $\theta(\sim\varphi) := \Sigma^* \setminus \theta(\varphi)$.

We call such a mapping a *substitution*.

A *language equation* is of the form $\varphi = \psi$ and a *language disequation* is of the form $\varphi \neq \psi$, where φ, ψ are language expressions. The substitution θ solves the equation $\varphi = \psi$ (the disequation $\varphi \neq \psi$) iff $\theta(\varphi) = \theta(\psi)$ ($\theta(\varphi) \neq \theta(\psi)$). We are interested in solvability of finite systems of language equations and disequations, where a substitution θ solves such a system iff it solves every (dis)equation in the system. Such a solution is called *finite* iff the languages $L_1 = \theta(X_1), \ldots, L_n = \theta(X_n)$ are finite.

Using the fact that, for any sets M_1, M_2, we have $M_1 = M_2$ iff $(M_1 \setminus M_2) \cup (M_2 \setminus M_1) = \varnothing$ and $M_1 = \varnothing = M_2$ iff $M_1 \cup M_2 = \varnothing$, we can transform a given finite system of language equations and disequations into an equivalent one (i.e., one with the same set of solutions) of the form

$$\varphi = \varnothing, \quad \psi_1 \neq \varnothing, \quad \ldots, \quad \psi_k \neq \varnothing. \tag{1}$$

In order to test such a system for (finite) solvability, we translate it into a looping tree automaton.

2.2 Translation into looping tree automata

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

[3]Note that the concatenation is *one-sided* in the sense that constants ($a \in \Sigma$) are only concatenated from the right to expressions.

with an element $f \in \Gamma$ and has as many successor nodes as is the rank of f. A *looping tree automaton* $\mathcal{A} = (Q, \Gamma, Q_0, \Delta)$ 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. 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 $(q_1, \ldots, q_k)$ 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 the looping tree automaton* $\mathcal{A}$ is defined as

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

It is well-known that the *non-emptiness problem* for looping tree automata, that is, the question whether, given such an automaton $\mathcal{A}$, the accepted language $L(\mathcal{A})$ is non-empty, is decidable in linear time [6].

When reducing a finite system of language (dis)equations of the form (1) to a looping tree automaton, we actually consider a very restricted case of looping tree automata. Assume that the alphabet used in the system is $\Sigma = \{a_1, \ldots, a_m\}$. Then we restrict our attention to a ranked alphabet Γ containing a single symbol γ of rank m. Thus, there is only one infinite tree, and the labeling of its nodes by γ can basically be ignored. Every node in this tree can be uniquely represented by a word $w \in \Sigma^*$, where each symbol a_i selects the ith successor of a node. Consequently, any run on this tree of a looping tree automaton with set of states Q can be represented as a mapping from Σ^* to Q.

Given a finite system of language (dis)equations of the form (1), let Φ denote the set of all subexpressions of $\varphi, \psi_1, \ldots, \psi_k$. We assume that $\varepsilon, X_1, \ldots, X_n \in \Phi$ (otherwise, we simply add them). In [5] we have shown how to construct a looping tree automaton $\mathcal{A}$ with the set of states $Q := 2^\Phi$, and with a 1–1-correspondence between runs of $\mathcal{A}$ and substitutions. To be more precise, given a run $r : \Sigma^* \to Q$ of $\mathcal{A}$, the corresponding substitution $\theta^r = \{X_1 \mapsto L_1^r, \ldots, X_n \mapsto L_n^r\}$ is obtained by defining

$$L_i^r := \{w \in \Sigma^* \mid X_i \in r(w)\}.$$

Conversely, given a substitution $\theta = \{X_1 \mapsto L_1, \ldots, X_n \mapsto L_n\}$, the corresponding run r_θ is

$$r_\theta(w) := \{\xi \in \Phi \mid w \in \theta(\xi)\}.$$

Lemma 2.1 ([5]) *The mapping of runs to substitutions introduced above is a bijection, and the mapping of substitutions to runs is its inverse.*

How do runs that correspond to solutions look like? Given a substitution θ, the corresponding run r_θ satisfies

$$\xi \in r_\theta(w) \text{ iff } w \in \theta(\xi)$$

for all $\xi \in \Phi$. Recall that our system is of the form (1) and that $\varphi, \psi_1, \ldots, \psi_k$ belong to Φ. Thus, θ solves the equation $\varphi = \varnothing$ iff $\varphi \notin r_\theta(w)$ for all $w \in \Sigma^*$, i.e., the run does not use any states containing φ. Consequently, if we remove from $\mathcal{A}$ all states containing φ, then we obtain an automaton whose runs are in a 1–1-correspondence with the solutions of $\varphi = \varnothing$. Let us call the resulting looping tree automaton $\mathcal{A}_\varphi$. Obviously, the size of $\mathcal{A}_\varphi$ is exponential in the size of the input system of language (dis)equations, and this automaton can be constructed in exponential time. To decide solvability of the equation $\varphi = \varnothing$ it is enough to test whether $\mathcal{A}_\varphi$ has a run, which can be done using the (linear-time) emptiness test for looping tree automata.

However, some of the runs of $\mathcal{A}_\varphi$ may correspond to substitutions that do not solve the disequations. If θ solves the disequation $\psi_i \neq \varnothing$, then there is a $w \in \Sigma^*$ such that $w \in \theta(\psi_i)$, which is equivalent to $\psi_i \in r_\theta(w)$.

Lemma 2.2 *A run r of $\mathcal{A}_\varphi$ corresponds to a solution of the whole system (1) iff for every $i, 1 \leq i \leq k$, there is a word $w \in \Sigma^*$ such that $\psi_i \in r_\theta(w)$.*

If we view the indices $1, \ldots, k$ as colors and assign to each state q of $\mathcal{A}_\varphi$ the color set $\kappa(q) := \{i \mid \psi_i \in q\}$, then the condition in the lemma can be reformulated as follows: we are looking for runs in which each color occurs in the color set of at least one state. We will show in the next section how one can check whether a run satisfying such an additional "color condition" exists.

Finiteness of a solution can also easily be expressed by a condition on runs. In fact, since we have $w \in \theta(X_i)$ iff $X_i \in r_\theta(w)$, we need to look for runs in which the variables X_i occur only finitely often. Let us call a state q of $\mathcal{A}_\varphi$ a *variable state* if $X_i \in q$ for some $i, 1 \leq i \leq n$.

Lemma 2.3 *A run r of $\mathcal{A}_\varphi$ corresponds to a* finite *solution of $\varphi = \varnothing$ iff it contains only finitely many variable states, i.e., the set $\{w \in \Sigma^* \mid r(w)$ is a variable state$\}$ is finite.*

3 Looping tree automata with colors

In this section, we first introduce a new type of automata that can express the "color condition" caused by disequations, and then analyze the complexity of the non-emptiness problem for these automata.

Definition 3.1 *A* looping tree automaton with colors *is of the form $\mathcal{A} = (Q, \Gamma, Q_0, \Delta, K, \kappa)$, where $\mathcal{A} = (Q, \Gamma, Q_0, \Delta)$ is a looping tree automaton, K is a finite set (of colors), and $\kappa : Q \to 2^K$ assigns to every state q a set of colors $\kappa(q) \subseteq K$.*

A run of $\mathcal{A} = (Q, \Gamma, Q_0, \Delta, K, \kappa)$ *on a tree* t *is a run of the underlying looping tree automaton* (Q, Γ, Q_0, Δ) *on* t*. The set* $\kappa(r)$ *of* colors of the run r *is defined as*

$$\kappa(r) := \{\nu \in K \mid \textit{there is a node } v \textit{ in } t \textit{ with } \nu \in \kappa(r(v))\}.$$

The run r *satisfies the* color condition *if* $K = \kappa(r)$*. The tree* t *is* accepted by the looping tree automaton with colors $\mathcal{A}$ *if there is a run of* $\mathcal{A}$ *on* t *that satisfies the color condition. The* language $L(\mathcal{A})$ accepted by the looping tree automaton with colors $\mathcal{A}$ *is the set of all trees accepted by* $\mathcal{A}$*.*

3.1 Decidability of the emptiness problem

In order to show decidability of the non-emptiness problem for looping tree automata with colors, we reduce it to the non-emptiness problem for Büchi tree automata. A *Büchi tree automaton* $\mathcal{A} = (Q, \Gamma, Q_0, \Delta, F)$ is a looping tree automaton that additionally is equipped with a set F of final states. A run r of this automaton on a tree t satisfies the *Büchi acceptance condition* if, on every infinite path through the tree, infinitely many nodes are labeled with final states. The tree t is *accepted* by the Büchi tree automaton $\mathcal{A}$ if there is a run of $\mathcal{A}$ on t that satisfies the Büchi acceptance condition. Again, the *language* $L(\mathcal{A})$ *accepted by the Büchi tree automaton* $\mathcal{A}$ is the set of all trees accepted by $\mathcal{A}$. It is well-known that the emptiness problem for Büchi tree automata is decidable in quadratic time [17].

Let $\mathcal{A} = (Q, \Gamma, Q_0, \Delta, K, \kappa)$ be a looping tree automaton with colors. The corresponding Büchi tree automaton $\mathcal{B}_\mathcal{A} = (Q', \Gamma, Q_0', \Delta', F)$ is defined as follows:

- $Q' := Q \times 2^K$;
- $Q_0' := \{(q, K) \mid q \in Q_0\}$;
- for $q \in Q$, $L \subseteq K$, and $f \in \Gamma$ of arity k we define

  $\Delta'((q, L), f) := \{((q_1, L_1), \ldots, (q_k, L_k)) \mid (q_1, \ldots, q_k) \in \Delta(q, f), L \setminus \kappa(q)$ is the union of disjoint sets $L_1, \ldots, L_k\}$;

- $F := Q \times \{\varnothing\}$.

The automaton $\mathcal{B}_\mathcal{A}$ simulates $\mathcal{A}$ in the first components of its states. The second component guesses in which subtree the still required colors are to be found. The Büchi acceptance condition ensures that only runs where these guesses are correct are accepting runs.

Proposition 3.2 $L(\mathcal{A}) = L(\mathcal{B}_\mathcal{A})$.

Proof. First, assume that r is a run of $\mathcal{A}$ on t that satisfies the color condition, i.e., $\kappa(r) = K$. For each color $\nu \in K$, select a node v_ν of t such that $\nu \in \kappa(r(v_\nu))$ and v_ν has minimal distance from the root, i.e., no node u in t strictly above v_ν satisfies $\nu \in \kappa(r(u))$. We now construct a run of $\mathcal{B}_\mathcal{A}$ on t by adding to r the second components of the states of $\mathcal{B}_\mathcal{A}$. Consider an arbitrary node v in t. We assign to this node the color set

$$\lambda(v) := \{\nu \in K \mid v_\nu = v \text{ or } v_\nu \text{ lies below } v\}.$$

The mapping r' from the nodes of t to the states of $\mathcal{B}_\mathcal{A}$ is defined as $r'(v) = (r(v), \lambda(v))$. We claim that this mapping is a run of $\mathcal{B}_\mathcal{A}$ on t that satisfies the Büchi acceptance condition.

To show that r' is indeed a run of $\mathcal{B}_\mathcal{A}$, consider an arbitrary node v of t. Let $v_1, \ldots, v_k$ be the successor nodes of v. We must show that $((r(v), \lambda(v)), t(v)) \to ((r(v_1), \lambda(v_1)), \ldots, (r(v_k), \lambda(v_k)))$ is a valid transition of $\mathcal{B}_\mathcal{A}$. Since r is a run of $\mathcal{A}$, we have $(r(v_1), \ldots, r(v_k)) \in \Delta(r(v), t(v))$, and thus it is sufficient to show that $\lambda(v) \setminus \kappa(r(v))$ is the disjoint union of $\lambda(v_1), \ldots, \lambda(v_k)$. Pairwise disjointness of the sets $\lambda(v_1), \ldots, \lambda(v_k)$ is an immediate consequence of the fact that we have chosen only one node v_ν for each color ν, and such a node can belong only to one of the successor subtrees of v. To show that

$$\lambda(v) \setminus \kappa(r(v)) = \lambda(v_1) \cup \ldots \cup \lambda(v_k),$$

first observe that $\nu \in \lambda(v_i)$ means that $v_\nu = v_i$ or v_ν lies below v_i. Thus, v_ν lies below v, which shows that $\nu \in \lambda(v)$. Since v_ν was chosen so that it has minimal distance from the root, $\nu \in \kappa(r(v))$ is not possible. Thus, we have shown that $\nu \in \lambda(v_i)$ implies $\nu \in \lambda(v) \setminus \kappa(r(v))$. Conversely, assume that $\nu \in \lambda(v) \setminus \kappa(r(v))$. Then $\nu \in \lambda(v)$ means that $v_\nu = v$ or v_ν lies below v. However, $\nu \notin \kappa(r(v))$ shows that the first option is not possible. Consequently, v_ν belongs to one of the subtrees below v, which yields $\nu \in \lambda(v_i)$ for some $i, 1 \leq i \leq k$.

To show that r' satisfies the Büchi acceptance condition, consider the maximal distance of the color nodes v_ν for $\nu \in K$ from the root. Since K is finite, this maximal distance is a well-defined natural number d. Any node v that has a larger distance from the root than d cannot be equal to or have below itself any of the color nodes. Consequently, $\lambda(v) = \varnothing$. This shows that, in any infinite path in t, infinitely many nodes are labeled by r' with a state of $\mathcal{B}_\mathcal{A}$ whose second component is $\varnothing$. Since these are exactly the final states of $\mathcal{B}_\mathcal{A}$, this shows that r' satisfies the Büchi acceptance condition. Thus, we have shown that any tree accepted by $\mathcal{A}$ is also accepted by $\mathcal{B}_\mathcal{A}$, i.e., $L(\mathcal{A}) \subseteq L(\mathcal{B}_\mathcal{A})$.

To show that the inclusion in the other direction also holds, assume that r' is a run of $\mathcal{B}_\mathcal{A}$ on t that satisfies the Büchi acceptance condition. Let r be the mapping from the nodes of t to Q that is obtained from r' by disregarding the second components of states, i.e., if $r'(v) = (q, L)$, then $r(v) = q$. Obviously, r is a run of $\mathcal{A}$. It remains to show that it satisfies the color condition. Assume that

there is a color $\nu \in K$ that does not occur in $\kappa(r)$. We claim that this implies that there is an infinite path in t satisfying the following property: $(*)$ for any node v in this path, the second component of $r'(v)$ contains ν. Since this would imply that r' does not satisfy the Büchi acceptance condition, this then shows that such a color cannot exist, i.e., $K = \kappa(r)$.

To show the existence of an infinite path satisfying property $(*)$, it is sufficient to show the following: if v is a node in t such that the second component L of $r'(v)$ contains ν, then there is a successor node v_i of v such that the second component L_i of $r'(v_i)$ contains ν. The existence of such a successor node is an immediate consequence of the definition of the transition relation of $\mathcal{B}_\mathcal{A}$ and the fact that ν cannot be an element of $\kappa(r(v))$ since we have assumed $\nu \notin \kappa(r)$. □

As an immediate consequence of this proposition we have that the non-emptiness problem for looping tree automata with colors is decidable: given a looping tree automaton with colors $\mathcal{A}$, we can construct $\mathcal{B}_\mathcal{A}$, and then use the quadratic non-emptiness test for Büchi automata. Regarding the complexity of this decision procedure, we can observe that the size of $\mathcal{B}_\mathcal{A}$ is polynomial in the number of states of $\mathcal{A}$, but exponential in the number of colors.

Theorem 3.3 *The non-emptiness problem for looping tree automata with colors can be decided in time polynomial in the number of states, but exponential in the number of colors.*

The non-emptiness for looping tree automata with colors can actually also be reduced to the one for looping tree automata without colors. This reduction will be described in Subsection 3.2. In contrast to the reduction to Büchi tree automata described above, this reduction is not language-preserving, but only emptiness-preserving. In fact, we will show in Subsection 3.3 that looping tree automata with colors are more expressive than looping tree automata.

Finally, in Subsection 3.4 we will determine the exact worst-case complexity of the emptiness problem for looping tree automata with colors w.r.t. the overall size of the input automaton.

3.2 Reduction to looping tree automata without colors

It is, actually, also possible to reduce the non-emptiness problem for looping tree automata with colors to the one for looping tree automata without colors. In this reduction, however, we do not get an automaton accepting the same tree language, but only one that behaves the same w.r.t. the emptiness problem. More precisely, we will show that we can transform the Büchi tree automaton $\mathcal{B}_\mathcal{A}$ constructed

from the looping tree automaton with colors $\mathcal{A}$ into a looping tree automaton without colors $\mathcal{C}_\mathcal{A}$ such that $L(\mathcal{B}_\mathcal{A}) \neq \varnothing$ iff $L(\mathcal{C}_\mathcal{A}) \neq \varnothing$.

Let $\mathcal{A} = (Q, \Gamma, Q_0, \Delta, K, \kappa)$ be a looping tree automaton with colors, and $\mathcal{B}_\mathcal{A} = (Q', \Gamma, Q'_0, \Delta', F)$ the corresponding Büchi tree automaton, as introduced above. Recall that a transition $((q, L), f) \to ((q_1, L_1), \ldots, (q_k, L_k))$ is called *decreasing* if $|L| > |L_i|$ holds for all $i, 1 \leq i \leq k$, and *non-decreasing* otherwise. Lemma 3.8 states that any non-decreasing transition

$$((q, L), f) \to ((q_1, L_1), \ldots, (q_k, L_k))$$

satisfies the following two properties:

- $\kappa(q) \cap L = \varnothing$;
- there is an $i, 1 \leq i \leq k$, such that $L_i = L$ and $L_j = \varnothing$ for all $j \neq i$.

We call a non-decreasing transition $((q, L), f) \to ((q_1, L_1), \ldots, (q_k, L_k))$ *trivial* if $L = \varnothing$.

Let N be the cardinality of Q. The looping tree automaton without colors $\mathcal{C}_\mathcal{A}$ is defined as $\mathcal{C}_\mathcal{A} = (Q' \times \{0, 1, \ldots, N\}, \Gamma, Q'_0 \times \{N\}, \Delta'')$, where $\Delta''((q, L, n), f))$ consists of all tuples $((q_1, L_1, n_1), \ldots, (q_k, L_k, n_k))$ such that

$((q, L), f) \to ((q_1, L_1), \ldots, (q_k, L_k))$ is a transition of $\mathcal{B}_\mathcal{A}$ and

- $n_1 = \cdots = n_k = N$ if this transition is decreasing or trivial
- $n > 0$, $n_i = n - 1$, and $n_j = N$ for all $j \neq i$ if this is a non-decreasing transition with $L_i = L \neq \varnothing$.

Lemma 3.4 *$L(\mathcal{B}_\mathcal{A}) \neq \varnothing$ iff $L(\mathcal{C}_\mathcal{A}) \neq \varnothing$.*

Proof. First, assume that $L(\mathcal{C}_\mathcal{A}) \neq \varnothing$. Let r be a run of $\mathcal{C}_\mathcal{A}$ on some tree t. If we remove the third component of all states in this run, then we obviously obtain a run r' of $\mathcal{B}_\mathcal{A}$. It remains to show that this run satisfies the Büchi acceptance condition. Assume that there is an infinite path in t such that r' does not assign states with empty second component to the nodes of this path from some point on. Then, less than $|K|$ decreasing transitions and no trivial transitions can be applied in this path. Consequently, from some point on, only non-trivial non-decreasing transitions are applied in such a way that the successor node with the non-empty color set is always the one on the path. However, by the definition of Δ'', this is possible at most N times in a row, and not infinitely often.

Second, assume that $L(\mathcal{B}_\mathcal{A}) \neq \varnothing$. Let r' be a run of $\mathcal{B}_\mathcal{A}$ on some tree t. Since the set of final states of $\mathcal{B}_\mathcal{A}$ is *reproducing* in the sense that successor states of

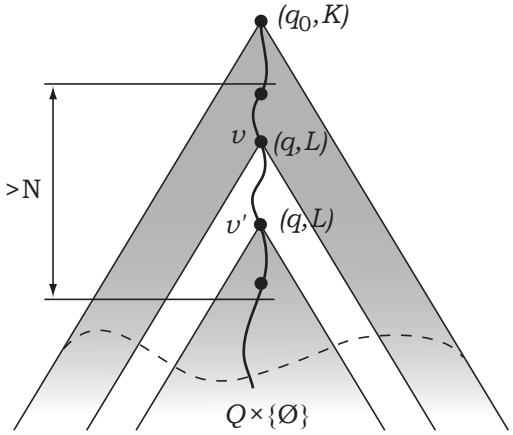

Figure 1: Cutting out a slice between two repeated states.

a final state are also final, we know that every infinite path in r' ends with an infinite sequence of final states. By König's Lemma, this implies that we have only finitely many occurrences of non-final states in r'. We choose r' and t such that the number of occurrences of non-final states is minimal, i.e., there is no tree $\widehat{t}$ and run $\widehat{r}'$ of $\mathcal{B}_\mathcal{A}$ on $\widehat{t}$ that contains fewer occurrences of non-final states.

If such a minimal run r' on a tree t cannot be turned into a run of $\mathcal{C}_\mathcal{A}$ on t by adding appropriate numbers as third components of the states, then there is a path in r' that contains a sequence of consecutive non-trivial non-decreasing transitions of length greater than N. Since the transitions are non-trivial, the states in this sequence are all non-final, and since it is longer than N, at least one of the states occurs twice. We can now modify t and r' by cutting out the slice between these repeating states: if the first occurrence of this state is at node v and the second at node v', then we remove all nodes equal to or below v that are not equal to or below v', as illustrated in Figure 1.

It is easy to see that this yields a tree and a run on this tree that satisfies the Büchi acceptance condition. However, since this new run contains fewer occurrences of non-final states, this contradicts our choice of t and r'. Consequently, r' can be turned into a run of $\mathcal{C}_\mathcal{A}$ by adding appropriate numbers as third components of the states. □

Together with Proposition 3.2, this lemma yields the following proposition.

Proposition 3.5 *For every looping tree automata with colors $\mathcal{A}$ we can effectively construct a looping tree automaton without colors $\mathcal{C}_\mathcal{A}$ such that $L(\mathcal{A}) \neq \varnothing$ iff $L(\mathcal{C}_\mathcal{A}) \neq \varnothing$. This construction can be carried out in time polynomial in the number of states of $\mathcal{A}$, but exponential in the number of colors of $\mathcal{A}$.*

3.3 Expressiveness

We say that a class $\mathcal{C}_2$ of tree automata is *at least as expressive* as another such class $\mathcal{C}_1$ if every tree language that can be accepted by an automaton of class $\mathcal{C}_1$ can also be accepted by an automaton of class $\mathcal{C}_2$. The class $\mathcal{C}_2$ is *more expressive* than the class $\mathcal{C}_1$ if additionally there is a tree language that can be accepted by an automaton of class $\mathcal{C}_2$, but not by one of class $\mathcal{C}_1$.

Theorem 3.6 *Büchi tree automata are more expressive than looping tree automata with colors, and looping tree automata with colors are more expressive than looping tree automata.*

Proof. Proposition 3.2 shows that Büchi tree automata are at least as expressive as looping tree automata with colors. Thus, to show the first statement of the theorem, it remains to exhibit a tree language that can be accepted by a Büchi tree automaton, but not by a looping tree automaton with colors. Let $\Gamma := \{a, b\}$ where a, b are function symbols of arity 1, and define

$$L_{\infty a} := \{t \mid t \text{ is an infinite tree over } \Gamma \text{ containing } a \text{ infinitely often}\}.$$

It is well-known (and easy to see) that $L_{\infty a}$ can be accepted by a Büchi tree automaton. Now assume that $\mathcal{A} = (Q, \Gamma, Q_0, \Delta, K, \kappa)$ is a looping tree automaton with colors that accepts $L_{\infty a}$, and let $n > |Q|$. Consider the infinite tree $t \in L_{\infty a}$ whose only path starts with a, and where between two consecutive occurrences of a there are exactly n occurrences of b. Let r be a run of $\mathcal{A}$ that accepts t, i.e., satisfies the color condition. Thus, there is a node v in t such that all the colors in K have already been seen when the run reaches this node. Without loss of generality we can assume that v is labeled with a (otherwise, we just go to the next a). In the b-block of length $n > |Q|$ after this a, r must label two different nodes with the same state from Q. But then we can modify t into t' such that all nodes after v are labeled with b, and there is a run r' of $\mathcal{A}$ on t' that coincides with r up to the node v. Thus, r' also satisfies the color condition, which shows that $\mathcal{A}$ accepts $t' \notin L_{\infty a}$. This is a contradiction, which shows that a looping tree automaton with colors accepting $L_{\infty a}$ cannot exist.

Regarding the second statement, any looping tree automaton can be viewed as a looping tree automaton with colors where the color set is empty, which shows that looping tree automata with colors are at least as expressive as looping tree automata. Thus, it remains to exhibit a tree language that can be accepted by a looping tree automaton with colors, but not by a looping tree automaton without colors. Let Γ be as above, and define

$$L_{\geq 1a} := \{t \mid t \text{ is an infinite tree over } \Gamma \text{ containing at least one } a\}.$$

The following looping tree automaton with colors $\mathcal{A} = (Q, \Gamma, Q_0, \Delta, K, \kappa)$ obviously accepts $L_{\geq 1a}$:

- $Q := \{q_a, q_b\} =: Q_0$;
- $\Delta(q_a, a) := \{q_a, q_b\}, \Delta(q_b, b) := \{q_a, q_b\}, \Delta(q_a, b) := \varnothing =: \Delta(q_b, a)$;
- $K := \{a\}$ and $\kappa(q_a) := \{a\}, \kappa(q_b) := \varnothing$.

Now assume that $\mathcal{A}' = (Q', \Gamma, Q'_0, \Delta')$ is a looping tree automaton without colors that accepts $L_{\geq 1a}$, and let $n' > |Q'|$. Consider the tree $t' \in L_{\geq 1a}$ that starts with n' nodes labeled with b, followed by a node labeled with a, and after that has only nodes labeled with b. Then there must be a run r' of $\mathcal{A}'$ on t'. Since $n' > |Q'|$, there is a state of Q' that occurs twice in r' on the first n nodes. Consequently, if we modify t' into t'' by replacing the only a by b, we obtain a tree $t'' \notin L_{\geq 1a}$ on which nevertheless there is a run of $\mathcal{A}'$. This is a contradiction, which shows that a looping tree automaton without colors accepting $L_{\geq 1a}$ cannot exist. □

3.4 The exact complexity of the emptiness problem

If we consider the complexity of the emptiness test described in Subsection 3.1 w.r.t. the overall size of the input automaton, then the test yields an ExpTime upper bound for the emptiness problem. In this section, we show that the problem is actually NP-complete.

We show *NP-hardness of the non-emptiness problem for looping tree automata with colors* by a simple reduction from SAT, the satisfiability problem for sets of clauses in propositional logic. Let $\mathcal{P} = \{p_1, \ldots, p_n\}$ be a set of propositional variables, and $\mathcal{L} = \mathcal{P} \cup \{\neg p_1, \ldots, \neg p_n\}$ the corresponding set of literals. Recall that a clause c is a set of literals $\{\ell_1, \ldots, \ell_m\}$, which stands for the disjunction $\ell_1 \vee \ldots \vee \ell_m$ of these literals. A set of clauses $\mathcal{C} = \{c_1, \ldots, c_p\}$ is read conjunctively, i.e., a propositional valuation satisfies $\mathcal{C}$ iff it satisfies all clauses in $\mathcal{C}$. Given a set of clauses $\mathcal{C} = \{c_1, \ldots, c_p\}$ built using literals from $\mathcal{L} = \mathcal{P} \cup \{\neg p_1, \ldots, \neg p_n\}$, we define the corresponding looping tree automaton with colors $\mathcal{A}_\mathcal{C} = (Q, \Gamma, Q_0, \Delta, K, \kappa)$ as follows:

- $\Gamma := \{f\}$ where f has arity 1;
- $Q := \mathcal{L} \cup \{q_{\text{loop}}\}$;
- $Q_0 := \{p_1, \neg p_1\}$;
- for $1 \leq i < n$ and $\ell \in \{p_i, \neg p_i\}$ we define $\Delta(\ell, f) := \{p_{i+1}, \neg p_{i+1}\}$;
- for $\ell \in \{p_n, \neg p_n, q_{\text{loop}}\}$ we define $\Delta(\ell, f) := \{q_{\text{loop}}\}$;
- $K := \mathcal{C}$;
- $\kappa(\ell) := \{c \in \mathcal{C} \mid \ell \in c\}$ for $\ell \in \mathcal{L}$ and $\kappa(q_{\text{loop}}) := \varnothing$.

Obviously, the size of $\mathcal{A}_\mathcal{C}$ is polynomial in the size of $\mathcal{L}$ and $\mathcal{C}$.

A run r of $\mathcal{A}_\mathcal{C}$ on the unique infinite tree over Γ contains, for every $i, 1 \leq i \leq n$, either p_i or $\neg p_i$, i.e., it determines a propositional valuation. If this run satisfies the color condition, then every clause c belongs to $\kappa(r)$, i.e., there is a literal ℓ that occurs in r (i.e., ℓ is true in the valuation determined by r) and that is contained in c. This shows that runs satisfying the color condition determine valuations that satisfy all clauses in $\mathcal{C}$. Conversely, a propositional valuation determines a unique run r, by choosing for every i the literal that is true in this valuation. If the valuation satisfies $\mathcal{C}$, then for each clause c one of its literals is true, and thus occurs in r. Consequently, each clause occurs in the color set $\kappa(r)$, which shows that r satisfies the color condition. Therefore, the clause set $\mathcal{C}$ is satisfiable iff $L(\mathcal{A}_\mathcal{C}) \neq \varnothing$.

Since the satisfiability problem for sets of propositional clauses is NP-hard, this shows that the same is true for the non-emptiness problem for looping tree automata with colors.

Proposition 3.7 *The non-emptiness problem for looping tree automata with colors is NP-hard.*

To show that the *non-emptiness problem for looping tree automata with colors is in NP* we consider the Büchi tree automaton constructed in the previous subsection. But first, we eliminate all states in the given automaton that do not occur in any run: these states can be identified in polynomial time using the emptiness test for looping tree automata [6]. The resulting automaton has the same set of runs on any tree, and thus also accepts the same language.

Let us now assume that all states of the looping tree automaton with colors $\mathcal{A} = (Q, \Gamma, Q_0, \Delta, K, \kappa)$ occur in some run, and that the set of colors K is non-empty.[4] Let $\mathcal{B}_\mathcal{A} = (Q', \Gamma, Q'_0, \Delta', F)$ be the Büchi automaton constructed from $\mathcal{A}$ in the previous section. Call a transition $((q, L), f) \to ((q_1, L_1), \ldots, (q_k, L_k))$ *decreasing* if $|L| > |L_i|$ holds for all $i, 1 \leq i \leq k$. Otherwise, the transition is called *non-decreasing*. The following lemma is an easy consequence of the definition of Δ'.

Lemma 3.8 *If $((q, L), f) \to ((q_1, L_1), \ldots, (q_k, L_k))$ is non-decreasing, then $\kappa(q) \cap L = \varnothing$ and there is an $i, 1 \leq i \leq k$, such that $L_i = L$ and $L_j = \varnothing$ for all $j \neq i$.*

Now, assume that r is a run of $\mathcal{B}_\mathcal{A}$ satisfying the Büchi acceptance condition. This run starts with an initial state $(q_0, K) \in Q'_0 = Q_0 \times \{K\}$. If the first transition that is applied is a non-decreasing transition, then there is exactly one

[4] If $K = \varnothing$, then $\mathcal{A}$ is a normal looping tree automaton, for which the non-emptiness problem is decidable in polynomial time.

successor node n_1 of the root to which r assigns a state with $K \neq \varnothing$ as second component, whereas all the other nodes are assigned states with empty second components (i.e., final states). If another non-decreasing transition is applied to n_1, then there is exactly one successor node of n_1 to which r assigns a state with $K \neq \varnothing$ as second component, etc. Since r satisfies the Büchi acceptance condition, after a finite number of non-decreasing steps we reach a node v to which a decreasing transition is applied. Let this decreasing transition be of the form $((q, K), \_) \to ((q_1, L_1), \ldots, (q_k, L_k))$ (where here and in the following, the alphabet symbol from Γ is irrelevant). Since the transition is decreasing, we have $|K| > |L_i|$ for all $i, 1 \leq i \leq k$. Let $v_1, \ldots, v_k$ be the successor nodes of v, and consider all v_i such that $L_i \neq \varnothing$. We can now apply the same analysis as for the root and K to the nodes v_i and $L_i \neq \varnothing$, i.e., we follow a chain of non-decreasing transitions that reproduce L_i until we find the next decreasing transition. This can be done until all color sets are empty. Basically, this construction yields a finite tree of decreasing transitions satisfying certain easy to check properties (see Definition 3.9 below). Our NP-algorithm guesses such a tree and checks whether the required properties are satisfied. Before we can formally define the relevant properties of this tree, we need to introduce one more notation.

Let $L \subseteq K$ be a non-empty set of colors and let q, q' be states in Q. We say that q' is *directly L-reachable* from q if there is a transition $(q, \_) \to (q_1, \ldots, q_k)$ in Δ such that $q' = q_i$ for some $i, 1 \leq i \leq k$, and $L \cap \kappa(q) = \varnothing$. Note that this implies that there is a non-decreasing transition $((q, L), \_) \to ((q_1, L_1), \ldots, (q_k, L_k))$ with $L_i = L$ and $L_j = \varnothing$ for $j \neq i$ in the transition relation Δ' of $\mathcal{B}_\mathcal{A}$. We say that q' is *L-reachable* from q if there is a sequence of states $p_0, \ldots, p_\ell$ ($\ell \geq 0$) such that $q = p_0$, $q' = p_\ell$, and p_{i+1} is directly L-reachable from p_i for all $i, 0 \leq i < \ell$.

Definition 3.9 *Given a looping tree automaton with colors $\mathcal{A}$ and the corresponding Büchi tree automaton $\mathcal{B}_\mathcal{A}$, a* dt-tree *for $\mathcal{B}_\mathcal{A}$ is a finite tree T whose nodes are decreasing transitions of $\mathcal{B}_\mathcal{A}$ such that the following properties are satisfied:*

- *the root of T is of the form $((q, K), \_) \to ((q_1, L_1), \ldots, (q_k, L_k))$ such that q is K-reachable from some initial state of $\mathcal{A}$;*
- *if $((q, L), \_) \to ((q_1, L_1), \ldots, (q_k, L_k))$ is a node in T and $i_1, \ldots, i_\ell$ are all the indices i with $L_i \neq \varnothing$, then this node has ℓ successor nodes of the form $((q'_{i_j}, L_{i_j}), \_) \to \cdots$ such that q'_{i_j} is L_{i_j}-reachable from q_{i_j} for $j = 1, \ldots, \ell$.*

Note that the leaves of a dt-tree are labeled with transitions

$$((q, L), \_) \to ((q_1, L_1), \ldots, (q_k, L_k))$$

for which $L_1 = \ldots = L_k = \varnothing$.

Lemma 3.10 *We have $L(\mathcal{B}_\mathcal{A}) \neq \varnothing$ iff there exists a dt-tree for $\mathcal{B}_\mathcal{A}$.*

Proof. For the "only if" direction, we have already sketched above how a run of $\mathcal{B}_\mathcal{A}$ satisfying the Büchi acceptance condition on some tree t can be used to construct a dt-tree for $\mathcal{B}_\mathcal{A}$.

For the "if" direction, it also easy to see how a dt-tree T for $\mathcal{B}_\mathcal{A}$ can be turned into a run r_T of $\mathcal{B}_\mathcal{A}$ satisfying the Büchi acceptance condition. If the root of T is of the form $((q, K), \_) \to ((q_1, L_1), \ldots, (q_k, L_k))$, where q is K-reachable from the initial state q_0 of $\mathcal{A}$, then the run r_T labels the root with (q_0, K). It uses the fact that q is K-reachable from q_0 to apply a sequence of non-decreasing transitions leading from the root to a node v labeled with (q, K). The nodes v' branching off from this path from the root to v are labeled with states of the form $(q', \varnothing)$. By our assumptions on $\mathcal{A}$, q' occurs in some run $r_{q'}$ of $\mathcal{A}$. This run can be used to generate the part of r_T below the node v' by using the part of $r_{q'}$ below the node labeled by $r_{q'}$ with q', and extending its states with the second component $\varnothing$.

Now, let us go back to the node v, which is labeled with (q, K) by r_T. Its successor nodes are labeled according to the decreasing transition $((q, K), \_) \to ((q_1, L_1), \ldots, (q_k, L_k))$. For the successors nodes of v labeled in this way, each state whose second component is $\varnothing$ can be treated as described above for the node v'. Each state with a non-empty second component can be processed as described for the root above, etc. Since the leaves of T have empty color sets on their right-hand sides, it is easy to see that this way we actually construct a run r_T satisfying the Büchi acceptance condition. □

The lemma shows that it is enough to design an algorithm that checks for the existence of a dt-tree. For this to be possible in non-deterministic polynomial time, we need to know that the size of dt-trees is polynomial in the size of $\mathcal{A}$. We can actually show the following linear bound in the number of colors.

Lemma 3.11 *The number of nodes of a dt-tree is bounded by $2{\cdot}|K|$.*

Proof. We call a decreasing transition $((q, L), \_) \to ((q_1, L_1), \ldots, (q_k, L_k))$ *removing* if $L \cap \kappa(q) \neq \varnothing$ and *branching* otherwise. Note that, for a branching transition $((q, L), \_) \to ((q_1, L_1), \ldots, (q_k, L_k))$, there must be indices $i \neq j$ such that L_i and L_j are non-empty.

In a dt-tree, for every color there is exactly one transition removing it, and every removing transition removes at least one color. Consequently, a dt-tree can contain at most $|K|$ removing transitions. Since decreasing transitions that are leaves in a dt-tree are necessarily removing, this also shows that the number of leaves of a dt-tree is bounded by $|K|$.

Any branching transition increases the number of leaves by at least one, which shows that a dt-tree can contain at most $|K| - 1$ branching transitions. Since every decreasing transition is either removing or branching, this completes the proof of the lemma. □

Together with Lemma 3.10, this lemma yields the desired NP upper bound.

Proposition 3.12 *The non-emptiness problem for looping tree automata with colors is in NP.*

Proof. After modifying the input automaton (in polynomial time) so that all states are used in some run, the algorithm starts by guessing a decreasing transition $((q, K), \_) \to ((q_1, L_1), \ldots, (q_k, L_k))$ and then checks (in polynomial time) whether q is K-reachable from some initial state. For every index i such that $L_i \neq \varnothing$, the algorithm then guesses a decreasing transition $((q'_i, L_i), \_) \to \ldots$ and checks whether q'_i is L_i-reachable from q_i, etc. The Lemma 3.11 ensures that overall only linearly many decreasing transitions need to be guessed. □

Note that, though the algorithm checks whether $L(\mathcal{B}_\mathcal{A}) \neq \varnothing$, it never explicitly constructs the (exponentially large) Büchi automaton $\mathcal{B}_\mathcal{A}$.

Given the NP-hardness result of Proposition 3.7, we thus have determined the exact worst-case complexity of the non-emptiness problem.

Theorem 3.13 *The non-emptiness problem for looping tree automata with colors is NP-complete.*

4 Applying the results

We will first show that the results obtained so far allow us to determine the exact complexity of (finite) solvability of finite systems of language (dis)equations with one-sided concatenation.

Proposition 4.1 *For a given finite system of language (dis)equations of the form (1), solvability and finite solvability are decidable in ExpTime.*

Proof. Let $\mathcal{A}_\phi = (Q, \Gamma, Q_0, \Delta)$ be the looping tree automaton constructed from the system (1) in Section 2.2, and define $K := \{1, \ldots, k\}$ and $\kappa(q) := \{i \in K \mid \psi_i \in q\}$ for all $q \in Q$. According to Lemma 2.2, the system (1) has a solution iff the looping tree automaton with colors $\mathcal{A} = (Q, \Gamma, Q_0, \Delta, K, \kappa)$ has a

run satisfying the color condition, i.e., accepts a non-empty language. As shown in the previous section, from $\mathcal{A}$ we can construct a Büchi automaton $\mathcal{B}_\mathcal{A}$ such that $L(\mathcal{A}) = L(\mathcal{B}_\mathcal{A})$ and the size of $\mathcal{B}_\mathcal{A}$ is polynomial in the number of states, but exponential in the number of colors of $\mathcal{A}$. Since the number of states of $\mathcal{A}$ is exponential in the size of the system (1), but the number of colors is linear in that size, the size of $\mathcal{B}_\mathcal{A}$ is exponential in the size of the system (1). As the emptiness problem for Büchi automata can be solved in polynomial time, this yields the desired ExpTime upper bound for solvability.

For finite solvability, we also must take the condition formulated in Lemma 2.3 into account, i.e., we are looking for runs of $\mathcal{B}_\mathcal{A}$ such that states of $\mathcal{B}_\mathcal{A}$ whose first components are variable states of $\mathcal{A}$ occur only finitely often. This condition can easily be expressed by modifying the Büchi automaton $\mathcal{B}_\mathcal{A}$, as described in a more general setting in the proof of the next lemma. Since the new Büchi automaton constructed in that proof is linear in the size of the original automaton, this yields the desired ExpTime upper bound for finite solvability. □

Lemma 4.2 *Let $\mathcal{B} = (Q, \Gamma, Q_0, \Delta, F)$ be a Büchi automaton and $P \subseteq Q$. Then we can construct in linear time a Büchi automaton $\mathcal{B}' = (Q', \Gamma, Q'_0, \Delta', F')$ such that*

$$L(\mathcal{B}') = \{t \mid \textit{there is a run of } \mathcal{B} \textit{ on } t \textit{ that contains only finitely many states from } P\}.$$

Proof. We define $Q' := Q \times \{1\} \cup (Q \setminus P) \times \{0\}$, $Q'_0 = Q_0 \times \{1\}$, $F' := (F \setminus P) \times \{0\}$, and

$$\begin{aligned}
\Delta'((q,1),\gamma) &:= \{((q_1,i_1),\ldots,(q_k,i_k)) \mid (q_1,\ldots,q_k) \in \Delta(q,\gamma), \\
&\qquad i_j = 1 \text{ if } q_j \in P, \\
&\qquad i_j \in \{0,1\} \text{ if } q_j \in Q \setminus P\}, \\
\Delta'((q,0),\gamma) &:= \{((q_1,0),\ldots,(q_k,0)) \mid (q_1,\ldots,q_k) \in \Delta(q,\gamma), \\
&\qquad q_1,\ldots,q_k \notin P\}.
\end{aligned}$$

Basically, this Büchi automaton guesses (by decreasing the second component of a state to 0) that from now on only states from $Q \setminus P$ will be seen. In fact, once the second component is 0, it stays 0 in all successor states, and only states from $Q \setminus P$ are paired with 0. Since F' contains only states with second component 0, this enforces that on every path eventually only states with second component 0 (and thus first component in $Q \setminus P$) occur. By König's lemma, this implies that a run of $\mathcal{B}'$ satisfying the Büchi acceptance condition contains only finitely many states with second component 1, and thus only finitely many states whose first component belongs to P. □

Since (finite) solvability of language equations that are simpler than the ones considered here are ExpTime-hard [4, 3], we thus have determined the exact complexity of (finite) solvability of our systems of language (dis)equations.

Syntax	Semantics
$A \in N_c$	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
$r \in N_r$	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
$\top$	$\Delta^{\mathcal{I}}$
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$\forall r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y : (x, y) \in r^{\mathcal{I}} \to y \in C^{\mathcal{I}}\}$

Table 1: Syntax and semantics of $\mathcal{FL}_0$-concept terms.

Theorem 4.3 *The problems of deciding solvability and finite solvability of finite systems of language (dis)equations of the form (1) are ExpTime-complete.*

4.1 Disunification in $\mathcal{FL}_0$

Description logics (DLs) are a well-investigated family of logic-based knowledge representation formalisms. They can be used to represent the relevant concepts of an application domain using concept terms, which are built from concept names and role names using certain concept constructors. From a semantic point of view, concept names and concept terms represent sets of individuals, whereas roles represent binary relations between individuals. The expressive power of a given DL depends on what concept constructors are available. Here, we are concerned with the DL $\mathcal{FL}_0$.

Unification in $\mathcal{FL}_0$ has been investigated in detail in [4]. In particular, it is shown there that solvability of $\mathcal{FL}_0$-unification problems is an ExpTime-complete problem. The ExpTime upper bound is based on a reduction to finite solvability of a restricted form of language equations with one-sided concatenation. In this subsection, we use Theorem 4.3 to show that this upper bound also holds for $\mathcal{FL}_0$-disunification problems. But first, we introduce syntax and semantics of $\mathcal{FL}_0$.

Starting from the finite and disjoint sets N_c of concept names and N_r of role names, $\mathcal{FL}_0$-*concept terms* are built using the concept constructors conjunction ($C \sqcap D$), value restriction ($\forall r.C$), and the top concept ($\top$). For example, the concept term

$$\mathsf{Woman} \sqcap \forall \mathsf{child}.\mathsf{Woman}.$$

describes the concept of "women having only daughters," using the concept name Woman and the role names child.

The semantics of concept terms is defined as in first-order predicate logic, using the notion of an interpretation. Given finite sets N_c of concept names and N_r of role names, an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty set $\Delta^{\mathcal{I}}$

(the domain of $\mathcal{I}$) and an interpretation function $\cdot^{\mathcal{I}}$ that maps each concept name $A \in N_c$ to a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and each role name $r \in N_r$ to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The extension of $\cdot^{\mathcal{I}}$ to arbitrary concept terms is defined inductively, as shown in the second column of Table 1. The concept term D *subsumes* the concept term C ($C \sqsubseteq D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all interpretations $\mathcal{I}$. Two concept terms C, D are *equivalent* ($C \equiv D$) iff they subsume each other. We write $C \not\equiv D$ to indicate that C, D are not equivalent.

In order to define (dis)unification of concept terms, we first have to introduce the notions of *concept patterns* and *substitutions* operating on concept patterns. To this purpose, we need a finite set of *concept variables* N_v (disjoint from $N_c \cup N_r$). $\mathcal{FL}_0$*-concept patterns* are $\mathcal{FL}_0$-concept terms defined over the set $N_c \cup N_v$ of concept names and the set N_r of role names. For example, given $A \in N_c$, $X \in N_v$, and $r \in N_r$, $\forall r.A \sqcap \forall r.\forall r.X$ is an $\mathcal{FL}_0$-concept pattern.

A *substitution* σ is a mapping from N_v into the set of all $\mathcal{FL}_0$-concept terms. This mapping is extended from variables to concept patterns in the obvious way:

- $\sigma(\top) := \top$ and $\sigma(A) := A$ for all $A \in N_c$,
- $\sigma(C \sqcap D) := \sigma(C) \sqcap \sigma(D)$ and $\sigma(\forall r.C) := \forall r.\sigma(C)$.

Definition 4.4 *An* $\mathcal{FL}_0$-disunification problem Γ *is a finite system of* equivalences *of the form* $C \equiv^? D$ *and* disequivalences *of the form* $C' \not\equiv^? D'$*, where* C*,* D*,* C'*,* D' *are* $\mathcal{FL}_0$*-concept patterns. The substitution* σ *is a* solution *of this problem iff* $\sigma(C) \equiv \sigma(D)$ *holds for all equivalences* $C \equiv^? D$ *in* Γ *and* $\sigma(C') \not\equiv \sigma(D')$ *holds for all disequivalences* $C' \not\equiv^? D'$ *in* Γ*. If such a solution exists, then the disunification problem is called* solvable*. An* $\mathcal{FL}_0$-unification problem *is a disunification problem that does not contain disequivalences. Solutions of unification problems are also called* unifiers*.*

For example, the substitutions $\sigma_1 = \{X \mapsto \forall r.A, Y \mapsto A\}$ and $\sigma_2 = \{X \mapsto \forall r.A, Y \mapsto A \sqcap \forall r.A\}$ are both solutions of the unification problem

$$\{X \sqcap \forall r.\forall r.A \equiv^? \forall r.X \sqcap \forall r.Y\},$$

but only σ_2 is a solution of the disunification problem

$$\{X \sqcap \forall r.\forall r.A \equiv^? \forall r.X \sqcap \forall r.Y, \quad X \not\equiv^? \forall r.Y\}.$$

The first step towards transforming $\mathcal{FL}_0$-disunification problems into finite systems of language (dis)equations is to transform concept terms and patterns into an appropriate normal form. By using the equivalence $\forall r.(C \sqcap D) \equiv \forall r.C \sqcap \forall r.D$ as a rewrite rule from left to right, any $\mathcal{FL}_0$-concept pattern can be transformed into an equivalent term that is either $\top$ or a (non-empty) conjunction of patterns of the form $\forall r_1.\cdots\forall r_m.A$ for $m \geq 0$ (not necessarily distinct) role names

$r_1, \ldots, r_m \in N_r$ and a concept name or concept variable $A \in N_c \cup N_v$. We abbreviate $\forall r_1. \cdots \forall r_m.A$ by $\forall r_1 \ldots r_m.A$, where $r_1 \ldots r_m$ is considered as a word over the alphabet of all role names N_r. In addition, instead of $\forall w_1.A \sqcap \ldots \sqcap \forall w_\ell.A$ we write $\forall L.A$ where $L := \{w_1, \ldots, w_\ell\}$ is a finite set of words over N_r. The term $\forall \varnothing.A$ is considered to be equivalent to $\top$. Using these abbreviations, any $\mathcal{FL}_0$-concept pattern C can be rewritten as

$$C \equiv \bigsqcap_{A \in N_c} \forall S_A.A \sqcap \bigsqcap_{X \in N_v} \forall S_X.X \tag{2}$$

for finite sets of words S_A, S_X ($A \in N_c, X \in N_v$) over the alphabet N_r. If C is a concept term, i.e., a pattern that does not contain concept variables, then we have $S_X = \varnothing$ for all $X \in N_v$. Recall that the term $\forall \varnothing.X$ is equivalent to $\top$, and can thus be removed from the conjunction.

Correctness of our (yet to be defined) reduction from disunification to solvability of language (dis)equations depends on the following (easily provable) characterization of (in)equivalence between $\mathcal{FL}_0$-concept terms:

Lemma 4.5 ([4]) *Let C, D be $\mathcal{FL}_0$-concept terms such that*

$$C \equiv \bigsqcap_{A \in N_c} \forall S_A.A \quad \textit{and} \quad D \equiv \bigsqcap_{A \in N_c} \forall T_A.A.$$

Then $C \equiv D$ iff $S_A = T_A$ for all $A \in N_c$, and thus $C \not\equiv D$ iff $S_A \neq T_A$ for some $A \in N_c$.

Next, let us analyze the application of a substitution to a concept pattern in terms of the normal form introduced above.

Lemma 4.6 *Let C be a pattern with normal form as described in* (2) *and let σ be a substitution such that*

$$\sigma(X) \equiv \bigsqcap_{A \in N_c} \forall S_{X,A}.A \tag{3}$$

for all $X \in N_v$. Then we have

$$\sigma(C) \equiv \bigsqcap_{A \in N_c} \forall (S_A \cup \textstyle\bigcup_{X \in N_v} S_X S_{X,A}).A,$$

where $S_X S_{X,A}$ stands for the concatenation of the language S_X with the language $S_{X,A}$.

For example, if we consider the concept pattern $C = \forall r.X \sqcap \forall r.Y$ and the substitution $\sigma = \{X \mapsto \forall r.A,\ Y \mapsto A \sqcap \forall r.A\}$, then we have $S_A = \varnothing$, $S_X = \{r\} = S_Y$, $S_{X,A} = \{r\}$, and $S_{Y,A} = \{\varepsilon, r\}$. The normal form of $\sigma(C) = \forall r.\forall r.A \sqcap \forall r.(A \sqcap \forall r.A)$ is $\forall \{r, rr\}.A$, and $\{r, rr\} = \varnothing \cup \{r\}\{r\} \cup \{r\}\{\varepsilon, r\}$.

Given patterns

$$C \equiv \bigsqcap_{A \in N_c} \forall S_A.A \sqcap \bigsqcap_{X \in N_v} \forall S_X.X \quad \text{and} \quad D \equiv \bigsqcap_{A \in N_c} \forall T_A.A \sqcap \bigsqcap_{X \in N_v} \forall T_X.X, \qquad (4)$$

we translate the equivalence $C \equiv^? D$ into the language equations

$$\widehat{E}_{C,D}(A): \qquad S_A \cup \bigcup\nolimits_{X \in N_v} S_X X_A = T_A \cup \bigcup\nolimits_{X \in N_v} T_X X_A$$

for all $A \in N_c$, and the disequivalence $C \not\equiv^? D$ into the language disequations

$$\widehat{E}_{C,D}(A): \qquad S_A \cup \bigcup\nolimits_{X \in N_v} S_X X_A \neq T_A \cup \bigcup\nolimits_{X \in N_v} T_X X_A$$

for all $A \in N_c$. Note the coefficients S_A, S_X, T_a, T_X in these (dis)equations are finite languages. The variables X_A in these (dis)equations are renamed copies of the variables $X \in N_v$. A (finite) solution of such a (dis)equation replaces the variables X_A by (finite) languages such that (in)equality holds.

The following proposition is an immediate consequence of Lemma 4.5 and Lemma 4.6.

Proposition 4.7 *Let C, D be patterns of the form* (4) *and σ be a substitution as described in* (3)*. Then we have*

1. *σ solves the equivalence $C \equiv^? D$ iff for every $A \in N_c$, the substitution $\{X_A \mapsto S_{X,A} \mid X \in N_v\}$ solves the language equation $\widehat{E}_{C,D}(A)$.*
2. *σ solves the disequivalence $C \not\equiv^? D$ iff for some $A \in N_c$, the substitution $\{X_A \mapsto S_{X,A} \mid X \in N_v\}$ solves the language disequation $\widehat{D}_{C,D}(A)$.*

As shown in [4], a finite set of equivalences can be encoded into a single equivalence. Thus, it is sufficient to consider disunification problems that contain only one equivalence. The following lemma is an immediate consequence of the above proposition.

Lemma 4.8 *The disunification problem $\{C_0 \equiv^? D_0, C_1 \not\equiv^? D_1, \ldots, C_k \not\equiv^? D_k\}$ has a solution iff for every $A \in N_c$, there is a substitution θ_A such that*

- *$\theta_A(X_A)$ is a finite language over N_r for all $A \in N_c$ and all variables $X \in N_v$;*
- *θ_A solves the language equation $\widehat{E}_{C_0,D_0}(A)$ for all $A \in N_c$;*
- *for every index $i \in \{1, \ldots, k\}$ there is a concept name $A \in N_c$ such that θ_A solves the language disequation $\widehat{D}_{C_i,D_i}(A)$.*

The only remaining problem is that the language (dis)equations $\widehat{E}_{C_0,D_0}(A)$ and $\widehat{D}_{C_i,D_i}(A)$ used in this lemma are actually not language equations with one-sided concatenation as introduced in Section 2, since concatenation of constants is performed from the left rather than from the right. However, this problem can easily be overcome by reversing all concatenations. For a finite language S, its mirror image S^{mir} is obtained in the obvious way, by mirroring all words in it, where $(a_1 \dots a_n)^{mir} := a_n \dots a_1$. Given the language equation

$$\widehat{E}_{C,D}(A): \qquad S_A \cup \bigcup_{X \in N_v} S_X X_A = T_A \cup \bigcup_{X \in N_v} T_X X_A,$$

its mirror image is

$$E_{C,D}(A): \qquad S_A^{mir} \cup \bigcup_{X \in N_v} X_A S_X^{mir} = T_A^{mir} \cup \bigcup_{X \in N_v} X_A T_X^{mir}.$$

The mirror image $D_{C,D}(A)$ of the language disequation $\widehat{D}_{C,D}(A)$ is defined in the same way. Obviously, (finite) solutions of $\widehat{E}_{C,D}(A)$ can be transformed into (finite) solutions of $E_{C,D}(A)$ by mirroring the languages in these solutions, and vice versa. The same is true for the disequations $\widehat{D}_{C,D}(A)$ and $D_{C,D}(A)$.

The following lemma is thus an immediate consequence of Lemma 4.8.

Lemma 4.9 *The disunification problem* $\{C_0 \equiv^? D_0, C_1 \not\equiv^? D_1, \dots, C_k \not\equiv^? D_k\}$ *has a solution iff for every* $A \in N_c$*, there is a substitution* θ_A *such that*

- $\theta_A(X_A)$ *is finite for all* $A \in N_c$ *and all variables* X *occurring in the problem;*
- θ_A *solves the language equation* $E_{C_0,D_0}(A)$ *for all* $A \in N_c$*;*
- *for every index* $i \in \{1, \dots, k\}$ *there is a concept name* $A \in N_c$ *such that* θ_A *solves the language disequation* $D_{C_i,D_i}(A)$*.*

In order to take care of the last condition of the lemma, we consider functions $f : \{1, \dots, k\} \to N_c$. Given such a function f, we define, for each $A \in N_c$, the system of language (dis)equations $DE_f(A)$ as

$$DE_f(A) := \{E_{C_0,D_0}(A)\} \cup \{D_{C_i,D_i}(A) \mid f(i) = A\}.$$

The following theorem is then an immediate consequence of Lemma 4.9.

Theorem 4.10 *The disunification problem* $\{C_0 \equiv^? D_0, C_1 \not\equiv^? D_1, \dots, C_k \not\equiv^? D_k\}$ *has a solution iff there is a function* $f : \{1, \dots, k\} \to N_c$ *such that, for every concept names* $A \in N_c$*, the system of language (dis)equations* $DE_f(A)$ *has a finite solution.*

Since there are exponentially many functions $f : \{1, \ldots, k\} \to N_c$ and finite solvability of each system of language (dis)equations $DE_f(A)$ can be tested in exponential time by Theorem 4.3, this yields an overall exponential time complexity. ExpTime-hardness already holds for the special case of unification [4].

Corollary 4.11 *Solvability of $\mathcal{FL}_0$-disunification problems is ExpTime-complete.*

4.2 Monadic set constraints

Set constraints [1] are constraint systems in which the variables range over sets of terms. They can, for example, be used for program analysis. Here, we are concerned with set constraints with negation, as considered in [7, 16, 2].

As already mentioned in [3] and [5], there is a close connection between language equations with one-sided concatenation and monadic set constraints, i.e., set constraints where all function symbols are unary or nullary. For the case of set constraints without negation (i.e., where only inclusions between sets are allowed), it has been known for a long time [1] that the unrestricted case is NExpTime-complete and the monadic one (with at least two unary symbols and at least one nullary symbol) is ExpTime-complete. For the case of set constraints with negation (i.e., where inclusions and negated inclusions between sets are allowed), NExpTime-completeness for the unrestricted case has been shown by several authors [7, 16, 2], but to the best of our knowledge, the monadic case has not been investigated. In this subsection, we use Theorem 4.3 to show the ExpTime upper bound for the monadic case also holds *with negation*. But first, we need to introduce set constraints with negation.

Given a finite signature Ω, i.e., a finite set of function symbols with associated arities, the *set $T(\Omega)$ of ground terms over Ω* is defined in the usual way: every nullary function symbol $a \in \Omega$ belongs to $T(\Omega)$, and if $f \in \Omega$ is an n-ary function symbol ($n \geq 1$) and $t_1, \ldots, t_n \in T(\Omega)$, then $f(t_1, \ldots, t_n) \in T(\Omega)$.

Definition 4.12 *Let $\mathcal{X}$ be a finite set of variables and Ω be a finite signature, which is disjoint from $\mathcal{X}$.* Set expressions over Ω and $\mathcal{X}$ *are defined inductively:*

- *Every variable $X \in \mathcal{X}$ and every nullary function symbol $a \in \Omega$ is a set expression.*
- *If $f \in \Omega$ is an n-ary function symbol ($n \geq 1$) and $E, E', E_1, \ldots, E_n$ are set expressions, then $E \cap E', E \cup E', \overline{E}, f(E_1, \ldots, E_n)$ are also set expressions.*

If E, E' are set expressions, then $E \subseteq E'$ is a positive set constraint *and $E \not\subseteq E'$ is a* negative set constraint*. A* finite system of set constraints with negation *is a finite set of positive and negative set constraints.*

Substitutions map the variables in $\mathcal{X}$ to sets of ground terms. Given a substitution $\theta : \mathcal{X} \to 2^{T(\Omega)}$, we define its *extension to set expressions* by induction:

- $\theta(a) := \{a\}$ for every nullary function symbol $a \in \Omega$;
- $\theta(E \cap E') := \theta(E) \cap \theta(E')$, $\theta(E \cup E') := \theta(E) \cup \theta(E')$, $\theta(\overline{E}) := T(\Omega) \setminus \theta(E)$, and $\theta(f(E_1, \ldots, E_n)) := \{f(t_1, \ldots, t_n) \mid t_1 \in \theta(E_1), \ldots, t_n \in \theta(E_n)\}$.

The substitution θ *solves* the positive set constraint $E \subseteq E'$ iff $\theta(E) \subseteq \theta(E')$ and the negative set constraint $E \not\subseteq E'$ iff $\theta(E) \not\subseteq \theta(E')$. It solves a finite system of set constraints with negation iff it solves all the (positive and negative) set constraints occurring in this system.

If the signature Ω contains only nullary and unary function symbols, then we call Σ a *monadic signature*. Systems of set constraints with negation over monadic signatures are called *systems of monadic set constraints with negation*.

Let $\Omega = \Sigma \cup \mathcal{C}$ be a monadic signature, where Σ is the set of unary function symbols in Ω, and $\mathcal{C}$ is the set of nullary function symbols in Ω, which we call constants. Any ground term is then of the form $f_1(f_2(\cdots f_k(a)\cdots))$ for $k \geq 0$ unary function symbols $t = f_1, \ldots, f_k \in \Sigma$ and a constant $a \in \mathcal{C}$. Using postfix notation, we can write t as a word $w_t := af_k \ldots f_1$ over the alphabet Ω. More precisely, we have $w_t \in \{a\}\Sigma^*$. Thus, sets of ground terms S correspond to languages $L_S := \{w_t \mid t \in S\} \subseteq \mathcal{C}\Sigma^*$. Conversely, for every language $L \subseteq \mathcal{C}\Sigma^*$ there exists a unique set of ground terms T_L such that $L = L_{T_L}$.

Set expressions E, as introduced in Definition 4.12, can now be translated into language expressions L_E, as defined in Section 2:

- if $E = a$ for $a \in \mathcal{C}$, then $L_E := a$;
- if $E = f(E')$, then $L_E := L_{E'}f$;
- $L_{E \cap E'} := L_E \cap L_{E'}$, $L_{E \cup E'} := L_E \cup L_{E'}$, and $L_{\overline{E}} := \overline{L_E}$.

A positive set constraint C: $E \subseteq E'$ can be translated into the language equation L_C: $L_E \cap \overline{L_{E'}} = \varnothing$ and a negative set constraint C': $E \not\subseteq E'$ into the language disequation $L_{C'}$: $L_E \cap \overline{L_{E'}} \neq \varnothing$. A finite system $\mathcal{S}$ of monadic set constraints with negation can thus be translated into a finite system of language (dis)equations $L_{\mathcal{S}} := \{L_C \mid C \in \mathcal{S}\}$.

Lemma 4.13 *Let $\mathcal{S}$ be a finite system of monadic set constraints with negation, and assume that $\mathcal{X} = \{X_1, \ldots, X_n\}$.*

1. *If $\theta = \{X_1 \mapsto S_1, \ldots, X_n \mapsto S_n\}$ is a solution of $\mathcal{S}$, then*

$$L_\theta := \{X_1 \mapsto L_{S_1}, \ldots, X_n \mapsto L_{S_n}\}$$

is a solution of $L_{\mathcal{S}}$, which satisfies $L_{S_1} \cup \ldots \cup L_{S_n} \subseteq \mathcal{C}\Sigma^*$.

2. *If* $\sigma = \{X_1 \mapsto L_1, \ldots, X_n \mapsto L_n\}$ *is a solution of* $L_{\mathcal{S}}$ *such that* $L_1 \cup \ldots \cup L_n \subseteq \mathcal{C}\Sigma^*$*, then*

$$T_\sigma := \{X_1 \mapsto T_{L_1}, \ldots, X_n \mapsto T_{L_n}\}$$

is a solution of $\mathcal{S}$*.*

The following lemma shows that the condition $L_{S_1} \cup \ldots \cup L_{S_n} \subseteq \mathcal{C}\Sigma^*$ can be expressed using language equations.

Lemma 4.14 *Let* $\mathcal{C} = \{a_1, \ldots, a_m\}$ *and* $\Sigma = \{f_1, \ldots, f_k\}$*. Any solution* σ *of the language equation*

$$X = Xf_1 \cup \ldots \cup Xf_k \cup a_1 \cup \ldots \cup a_k$$

satisfies $\sigma(X) = \mathcal{C}\Sigma^*$*.*

As an easy consequence of these two lemmas we thus obtain that solvability of finite systems of monadic set constraints with negation can be reduced in polynomial time to solvability of finite systems of language (dis)equations.

Theorem 4.15 *Let* $\mathcal{S}$ *be a finite system of monadic set constraints with negation, and assume that* X *is a variable that does not occur in* $\mathcal{X} = \{X_1, \ldots, X_n\}$*. Then* $\mathcal{S}$ *is solvable iff the finite system of language (dis)equations*

$$L_{\mathcal{S}} \cup \{X = Xf_1 \cup \ldots \cup Xf_k \cup a_1 \cup \ldots \cup a_k\} \cup \{(X_1 \cup \ldots \cup X_n) \cap \overline{X} = \varnothing\}$$

is solvable.

Since ExpTime-hardness already holds for monadic set constraints without negation [1], we have thus shown the following theorem.

Corollary 4.16 *Solvability of monadic set constraints with negation is ExpTime-complete.*

5 Conclusion

We have shown that solvability and finite solvability of systems of language (dis)equations are ExpTime-complete, in contrast to their undecidability (Σ_2^0-completeness) in the case of unrestricted concatenation [12]. We have used these results to obtain new complexity results for solving monadic set constraints with negation, and for disunification problems in the DL $\mathcal{FL}_0$. As a tool, we have introduced looping tree automata with colors. Though the results of Section 3 show that a direct reduction to the emptiness problem for Büchi tree automata would be possible, using looping tree automata with colors as intermediate formalism makes the presentation much clearer and easier to comprehend. In addition, we believe that these automata may be of interest also for other applications in logic.

References

[1] A. Aiken, D. Kozen, M.Y. Vardi, E.L. Wimmers, "The complexity of set constraints", *Computer Science Logic* (CSL'93, Swansea, UK, 1993), LNCS 832, 1–17.

[2] A. Aiken, D. Kozen, E.L. Wimmers, "Decidability of systems of set constraints with negative constraints" *Information and Computation*, 122(1) (1995), 30–44.

[3] F. Baader, R. Küsters, "Unification in a description logic with transitive closure of roles", *Logic for Programming, Artificial Intelligence, and Reasoning* (LPAR'01, Havana, Cuba, 2001), LNCS 2250, 217–232.

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

[5] F. Baader, A. Okhotin, "On Language Equations with One-sided Concatenation", *LTCS-Report LTCS-06-01*, Chair for Automata Theory, Institute for Theoretical Computer Science, TU Dresden, 2006. Available at http://lat.inf.tu-dresden.de/research/reports.html. A short version has been published in the Proceedings of the *20th International Workshop on Unification (UNIF'06)*.

[6] F. Baader, S. Tobies, "The inverse method implements the automata approach for modal satisfiability", *International Joint Conference on Automated Reasoning* (IJCAR'01, Siena, Italy, 2001), LNCS 2083, 92–106.

[7] W. Charatonik, L. Pacholski, "Negative set constraints with equality", *Logic in Computer Science* (LICS'94, Paris, France, 1994), 128–136.

[8] S. Ginsburg, H.G. Rice, "Two families of languages related to ALGOL", *J. of the ACM*, 9 (1962), 350–371.

[9] A. Jeż, A. Okhotin, "On the computational completeness of equations over sets of natural numbers", *Automata, Languages and Programming* (*ICALP 2008*, Reykjavík, Iceland, 2008), LNCS 5126, 63–74.

[10] M. Kunc, "What do we know about language equations?", *Developments in Language Theory* (DLT'07, Turku, Finland, 2007), LNCS 4588, 23–27.

[11] 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, 2010), LNCS 6224, 291–302.

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

[13] L. Pacholski and A. Podelski, "Set constraints: A pearl in research on constraints", *Principles and Practice of Constraint Programming* (CP'97, Linz, Austria, 1997), LNCS 1330, 549—562.

[14] 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.

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

[16] K. Stefánsson, "Systems of set constraints with negative constraints are NEXPTIME-complete", *Logic in Computer Science* (LICS'94, Paris, France, 1994), 137–141.

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