JMLR: Workshop and Conference Proceedings 21:54–68, 2012 The 11th ICGI

Clearing Restarting Automata and Grammatical Inference*

Peter Černo PETERCERNO@GMAIL.COM
Department of Computer Science
Charles University, Faculty of Mathematics and Physics
Malostranské nám. 25, 118 00 PRAHA 1, Czech Republic

Editors: Jeffrey Heinz, Colin de la Higuera and Tim Oates

Abstract

Clearing and subword-clearing restarting automata are linguistically motivated models of automata. We investigate the problem of grammatical inference for such automata based on the given set of positive and negative samples. We show that it is possible to identify these models in the limit. In this way we can learn a large class of languages. On the other hand, we prove that the task of finding a clearing restarting automaton consistent with a given set of positive and negative samples is NP-hard, provided that we impose an upper bound on the width of its instructions.

Keywords: grammatical inference, clearing restarting automata, subword-clearing restarting automata, formal languages.

1. Introduction

Restarting automata (Jančar et al., 1995) were introduced as a tool for modeling some techniques used for natural language processing. In particular they are used for analysis by reduction which is a method for checking (syntactical) correctness or non-correctness of a sentence. While restarting automata are quite general (see Otto (2006) for an overview), they still lack some properties which could facilitate their wider use. One of their drawbacks is, for instance, the lack of some intuitive way how to infer their instructions.

Clearing restarting automata were introduced in Černo and Mráz (2009, 2010) as a new restricted model of restarting automata which, based on a limited context, can only delete a substring of the current content of its tape. The model is motivated by the need for simpler definitions and simultaneously by aiming for efficient machine learning of such automata. In Černo and Mráz (2010) it was shown that this model is effectively learnable from positive samples of reductions and that in this way it is even possible to infer some non-context-free languages. Here we introduce a more general model, called subword-clearing restarting automata, which, based on a limited context, can replace a substring z of the current content of its tape by a proper substring of z. In this paper we focus on the grammatical inference of clearing and subword-clearing restarting automata from the given set of positive and negative samples.

* This work was partially supported by the Grant Agency of Charles University under Grant-No. 272111/A-INF/MFF and by the Czech Science Foundation under Grant-No. P103/10/0783 and Grant-No. P202/10/1333.

The used inference algorithm is inspired by strictly locally testable languages (McNaughton, 1974; Zalcstein, 1972). The idea of strictly locally testable languages rests in the assumption that in order to verify the membership of the given input word we only need to verify all the local parts of this word. Unfortunately the class of strictly locally testable languages is only a subclass of regular languages, which limits their wider use. Our models work in a similar local way. The main difference is that our automata can locally modify the content of their input tape by using rewriting instructions. The inference algorithm itself is responsible only for deciding, which rewriting instructions are justified, based on the given set of positive and negative samples. In the first phase, the algorithm uses the set of positive samples to infer all possible instruction candidates. In the second phase, it uses the set of negative samples for filtering out all "bad" instructions that violate the so-called error preserving property (i.e. instructions that allow a reduction from a negative sample to a positive sample). The output is the set of all surviving instructions. We show that, under certain assumptions, this algorithm runs in a polynomial time and can infer all clearing and subword-clearing restarting automata in the limit. In contrast with this result we show that the task of finding a clearing restarting automaton consistent with the given set of positive and negative samples is NP-hard, provided that we impose an upper bound on the width of instructions. This result resembles the famous result of Gold (1978) who showed that the construction of a minimum state automaton consistent with the given data is, in general, computationally difficult. Indeed, for every n-state finite automaton there exists an equivalent clearing restarting automaton that has the width of instruction bounded from above by $O(n)$ (Černo and Mráz, 2010).

Although clearing and subword-clearing restarting automata have their roots in restarting automata, we will study them from the perspective of the so-called string-rewriting systems. Our approach is reminiscent of the delimited string-rewriting systems introduced in Eyraud et al. (2007), which are expressive enough to define a nontrivial class of languages containing all regular languages and some context-free languages. Eyraud et al. (2007) presented a novel algorithm LARS (Learning Algorithm for Rewriting Systems) which identifies a large subclass of these languages in a polynomial time. In fact, a simplified version of LARS (de la Higuera, 2010) identifies any delimited string-rewriting system in the limit. The main difference between delimited string-rewriting systems and clearing (subword-clearing) restarting automata is that delimited string-rewriting systems use a specific order relation over the set of all terms and rules in order to make always only one single rule eligible for application for any given input string. This makes them an efficient (often linear) parsing device for strings with the membership problem decidable in a polynomial time. Our models, on the other hand, are nondeterministic and do not use any ordering. We also added an important concept of contexts into the definitions of our models by using the so-called context rewriting systems as a common framework for all our models. Although our inference algorithm is able to identify most of the languages presented in Eyraud et al. (2007), the exact relationship between the corresponding classes of languages is not clear at all.

The paper has the following structure. Section 2 introduces the used automata models. The new learning algorithm for identification of clearing and subword-clearing restarting automata in the limit from positive and negative samples is proposed in Section 3. In Section 4 we show that the task of finding a clearing restarting automaton consistent with a given set of positive and negative samples is NP-hard under the constraint that there is an

upper bound on the width of instructions. Conclusions are presented in Section 5. Because of the page limit we omit some of the proofs and refer the interested reader to the technical report Černo (2012). An implementation of the inference algorithm can be found on the following website: http://code.google.com/p/clearing-restarting-automata/.

2. Theoretical Background

We use the standard notation from the theory of automata and formal languages. As our reference concerning this field we use the monograph Hopcroft and Ullman (1969).

In the following we define our central concept, called *context rewriting systems*, which will serve us as a framework for clearing and subword-clearing restarting automata and other similar models.

Definition 1 (Černo and Mráz (2010)) *Let k be a positive integer. A* k-context rewriting system *(k-CRS for short) is a system $M = (\Sigma, \Gamma, I)$, where Σ is an input alphabet, $\Gamma \supseteq \Sigma$ is a working alphabet not containing the special symbols ¢ and \$, called* sentinels*, and I is a finite set of* instructions *of the form $(x, z \to t, y)$, where x is called the* left context*, $x \in LC_k = \Gamma^k \cup \{¢\} \cdot \Gamma^{\leq k-1}$, y is called the* right context*, $y \in RC_k = \Gamma^k \cup \Gamma^{\leq k-1} \cdot \{\$\}$ and $z \to t$ is called the* instruction-rule*, $z, t \in \Gamma^*$. The* width *of the instruction $i = (x, z \to t, y)$ is $|i| = |xzty|$.*

A word $w = uzv$ can be rewritten *into utv (denoted as $uzv \vdash_M utv$) if and only if there exists an instruction $i = (x, z \to t, y) \in I$ such that x is a suffix of $¢ \cdot u$ and y is a prefix of $v \cdot \$$. We often underline the rewritten part of the word w, and if the instruction i is known we use $\vdash_M^{(i)}$ instead of $\vdash_M$, i.e. $u\underline{z}v \vdash_M^{(i)} utv$. The relation $\vdash_M \subseteq \Gamma^* \times \Gamma^*$ is called the* rewriting relation*.*

Let $l \in \{\lambda, ¢\} \cdot \Gamma^*$, and $r \in \Gamma^* \cdot \{\lambda, \$\}$. A word $w = uzv$ can be rewritten in the context (l, r) *into utv (denoted as $uzv \vdash_M utv$* in the context (l, r)*) if and only if there exists an instruction $i = (x, z \to t, y) \in I$, such that x is a suffix of $l \cdot u$ and y is a prefix of $v \cdot r$. Unless told otherwise, we will use the* standard context $(l, r) = (¢, \$)$*.*

The language *associated with M is defined as $L(M) = \{w \in \Sigma^* \mid w \vdash_M^* \lambda\}$, where $\vdash_M^*$ is the reflexive and transitive closure of $\vdash_M$. Note that, by definition, $\lambda \in L(M)$.*

The characteristic language *associated with M is defined as $L_C(M) = \{w \in \Gamma^* \mid w \vdash_M^* \lambda\}$. Similarly, by definition, $\lambda \in L_C(M)$. Obviously, $L(M) = L_C(M) \cap \Sigma^*$.*

Remark 2 *We also include a special case $k = 0$ in Definition 1. In this case we define $LC_0 = RC_0 = \{\lambda\}$, and the rest of the definition remains the same.*

Observe that $\lambda \in L(M)$ holds for each k-CRS $M = (\Sigma, \Gamma, I)$. In the following, we call two languages $L_1, L_2 \subseteq \Sigma^*$ equivalent if $L_1 \cap \Sigma^+ = L_2 \cap \Sigma^+$. We simply ignore the empty word in this setting. The implicit acceptance of the empty word can be easily avoided by a slight modification of the definition of context rewriting systems, but in principle, we would not get a more powerful model.

All k-CRS's have the following basic property.

Lemma 3 (Error Preserving Property, Černo and Mráz (2010)) *Let $M = (\Sigma, \Gamma, I)$ be a k-CRS and u, v be two words over Γ. If $u \vdash_M^* v$ and $u \notin L(M)$, then $v \notin L(M)$.*

Context rewriting systems are very similar to the so-called string-rewriting systems (Book and Otto, 1993). A *string-rewriting system* S on Σ consists of finitely many pairs of strings from Σ^*, called *rewrite rules*, which are written $(l \to r)$. The *reduction relation* $\Rightarrow_S^*$ on Σ^* induced by S is the reflexive and transitive closure of the *single-step reduction relation* $\Rightarrow_S = \{(ulv, urv) \mid (l \to r) \in S, u, v \in \Sigma^*\}$. For a string $u \in \Sigma^*$, if there exists a string v such that $u \Rightarrow_S v$ holds, then u is called *reducible* modulo S. Otherwise, if such a string does not exist, then u is called *irreducible* modulo S. By $\mathsf{IRR}(S)$ we denote the set of all irreducible strings modulo S. It is easy to see that $\mathsf{IRR}(S)$ is a regular language. A string-rewriting system is called *length-reducing* if $|l| > |r|$ for each rule $(l \to r) \in S$.

String-rewriting systems play a central role in the broad concept of the so-called McNaughton families (Beaudry et al., 2003). A language $L \subseteq \Sigma^*$ is called *McNaughton language* if there exists a finite alphabet Γ strictly containing Σ, a finite string-rewriting system S on Γ, strings $t_1, t_2 \in (\Gamma \setminus \Sigma)^* \cap \mathsf{IRR}(S)$ and a letter $Y \in (\Gamma \setminus \Sigma) \cap \mathsf{IRR}(S)$ such that, for all $w \in \Sigma^*$: $w \in L \Leftrightarrow t_1 w t_2 \Rightarrow_S^* Y$. Here the symbols of Σ are *terminals*, while those of $\Gamma \setminus \Sigma$ can be seen as *nonterminals*. The language L is said to be *specified* by the four-tuple (S, t_1, t_2, Y) and this fact will be expressed as $L = L(S, t_1, t_2, Y)$. By placing various restrictions on the string-rewriting systems used we obtain different families of languages.

In this paper we follow a similar approach, but instead of using general strings $t_1, t_2 \in (\Gamma \setminus \Sigma)^*$ we use only the single letters $t_1 = ¢$, $t_2 = \$$, called the sentinels, and instead of the symbol Y we use the empty word. It is easy to see that context rewriting systems can be simulated by string-rewriting systems. Indeed, suppose that $M = (\Sigma, \Gamma, I)$ is a k-CRS and $Y \notin \Gamma$. Let us define a string-rewriting system $S(M) = \{(xzy \to xty) \mid (x, z \to t, y) \in I\} \cup \{(¢\$, Y)\}$. Apparently, $L(M) = L(S(M), ¢, \$, Y)$. This basically allows us to extend most of the terminology concerning string-rewriting systems also to context rewriting systems. We say, for instance, that a k-CRS M is *length-reducing* if the reduction relation $\to_R$ of the string-rewriting system $S = S(M)$ is length-reducing.

Since general k-CRS can simulate any type 0 grammar (according to the Chomsky hierarchy (Hopcroft and Ullman, 1969)), we consider only length-reducing k-CRS in this paper. The first model we introduce is called *clearing restarting automaton* which is a k-CRS such that $\Sigma = \Gamma$ and all its instruction-rules are of the form $z \to \lambda$, where $z \in \Sigma^+$.

Definition 4 (Černo and Mráz (2010)) *Let k be a nonnegative integer. A k-*clearing restarting automaton *(*k-cl-RA *for short) is a k-*CRS *$M = (\Sigma, \Sigma, I)$ (or $M = (\Sigma, I)$, for short), where for each instruction $i = (x, z \to t, y) \in I$: $z \in \Sigma^+$ and $t = \lambda$. Since t is always the empty word, we use the notation $i = (x, z, y)$.*

Remark 5 *Speaking about a k-*cl-RA *M we use "automata terminology," e.g. we say that M* accepts *a word w if $w \in L(M)$. By definition, each k-*cl-RA *accepts λ. If we say that a k-*cl-RA *M* recognizes *(or* accepts*) a language L, we always mean that $L(M) = L \cup \{\lambda\}$.*

Example 1 *Let $M = (\Sigma, I)$ be a* 1-cl-RA *with $\Sigma = \{a, b\}$ and I consisting of the following two instructions:*

(1) (a, ab, b),
(2) $(¢, ab, \$)$.

Then we have $aaa\underline{ab}bbb \vdash_M^{(1)} aa\underline{ab}bb \vdash_M^{(1)} a\underline{ab}b \vdash_M^{(1)} \underline{ab} \vdash_M^{(2)} \lambda$ *which means that* $aaaabbbb \vdash_M^* \lambda$*. So the word* $aaaabbbb$ *is accepted by* M*. It is easy to see that* M *recognizes the language* $L(M) = \{a^n b^n \mid n \geq 0\}$*.*

Clearing restarting automata are studied in Černo and Mráz (2010). We only mention that they can recognize all regular languages, some context-free languages and even some non-context-free languages.

Here we propose yet another model called *subword-clearing restarting automaton*, which can be useful in some grammatical inference scenarios.

Definition 6 *Let* k *be a nonnegative integer. A* k-subword-clearing restarting automaton *(*k-scl-RA *for short) is a* k-CRS $M = (\Sigma, \Sigma, I)$*, where for each instruction* $i = (x, z \to t, y) \in I$*:* $z \in \Sigma^+$ *and* t *is a proper subword of* z*.*

Subword-clearing restarting automata are strictly more powerful than clearing restarting automata. They can, for instance, recognize the language $\{a^n c b^n \mid n \geq 0\} \cup \{\lambda\}$, which lies outside the class of languages accepted by clearing restarting automata. However, they still cannot recognize all context-free languages. (Consider e.g. the language $\{ww^R \mid w \in \Sigma^*\}$).

3. Learning Schema

In this section we introduce a learning schema for clearing and subword-clearing restarting automata, which can be used to identify any hidden target model in the limit from positive and negative samples. In the following, the term *automaton* refers primarily to the clearing and subword-clearing restarting automaton, but the presented ideas can be easily generalized to other similar models obtained from context rewriting systems.

An input for our problem consists of two finite sets of words over an alphabet Σ: the set of positive samples S^+ and the set of negative samples S^-. Our goal is to find an automaton M, such that: $S^+ \subseteq L(M)$ and $S^- \cap L(M) = \emptyset$. We assume that $S^+ \cap S^- = \emptyset$ and $\lambda \in S^+$.

Without further restrictions, the task becomes trivial even for clearing restarting automata. Just consider the set of instructions $I = \{(¢, w, \$) \mid w \in S^+, w \neq \lambda\}$. Trivially, $L(M) = S^+$, where $M = (\Sigma, I)$. Therefore, in the following we impose an upper limit $l \geq 1$ on the width of instructions and a specific length $k \geq 0$ of contexts for the instructions of the resulting automaton. Note, that if we can effectively enumerate all automata satisfying these restrictions then the identification in the limit from positive and negative samples can be easily deduced from the classical positive result of Gold on the identification in the limit of the class of primitive recursive languages (see e.g. Lange et al. (2008) for a nice survey on learning indexed families of recursive languages). Nevertheless, we propose Algorithm 1, which, under certain conditions, works in a polynomial time.

Algorithm 1 deserves some explanation. First, the function Assumptions(S^+, l, k) returns some set of instruction candidates. Let us assume, for a moment, that this set already contains all instructions of the hidden target automaton. Then in Cycle 2 we gradually remove each instruction that allows a reduction from a negative sample to a positive sample (such instruction violates the error preserving property from Lemma 3). In Step 5 we remove redundant instructions and in Step 6 we check if the remaining set of instructions is consistent with the given input set of positive and negative samples. In other words, we

Algorithm 1: Learning schema Infer(S^+, S^-, l, k)

Input: The set of positive S^+ and negative S^- samples over Σ, $S^+ \cap S^- = \emptyset$, $\lambda \in S^+$. The maximal width of instructions $l \geq 1$. The length of contexts of instructions $k \geq 0$.

Output: An automaton consistent with (S^+, S^-), or **Fail**.

1 $\Phi \leftarrow$ Assumptions(S^+, l, k);
2 **while** $\exists w_- \in S^-, w_+ \in S^+, \phi \in \Phi : w_- \vdash^{(\phi)} w_+$ **do**
3 $\quad \Phi \leftarrow \Phi \setminus \{\phi\}$;
4 **end**
5 $\Phi \leftarrow$ Simplify(Φ);
6 **if** Consistent(Φ, S^+, S^-) **then**
7 $\quad$ **return** *Automaton with the set of instructions* Φ;
8 **end**
9 **Fail**;

check if (1) for all $w_+ \in S^+ : w_+ \vdash_\Phi^* \lambda$ and (2) for all $w_- \in S^- : w_- \nvdash_\Phi^* \lambda$. The condition (1) always holds, if we assume that in Step 1 we already have all instructions of the hidden target automaton. However, the condition (2) may fail. Therefore, the success of the above algorithm, depends both on the initial assumptions obtained in Step 1, and on the given set of positive and negative samples. Nevertheless, we will show that if we have a reasonable implementation of the function Assumptions, then there is always a set of positive samples S_0^+ and a set of negative samples S_0^- such that the above schema returns a correct solution for all sets of positive samples $S^+ \supseteq S_0^+$ and negative samples $S^- \supseteq S_0^-$ consistent with the hidden target automaton.

The time complexity of Algorithm 1 depends both on the time complexity of the function Assumptions in Step 1 and on the time complexity of the simplification procedure and the consistency check in Steps 5 and 6. There are correct implementations of the function Assumptions (both for clearing and subword-clearing restarting automata) that run in a polynomial time. In fact, they run in a linear time, if the maximal width of instructions l and the length of contexts k is considered to be a fixed constant. If the function Assumptions runs in a polynomial time then also the size of the set Φ is polynomial (with respect to the size of the input) and therefore also Cycle 2 runs in a polynomial time. It is also possible to implement the function Simplify so that it runs in a linear time with respect to the size of Φ, provided that both the maximal width of instructions l and the length of contexts k are considered to be fixed constants. Algorithm 2 shows a possible implementation of the function Simplify both for clearing and subword-clearing restarting automata.

Unfortunately, it is an open problem, whether the consistency check can be done in a polynomial time (both for clearing and subword-clearing restarting automata). But from the point of view of the identification in the limit, we can completely omit both the simplification step and the consistency check, because they do not affect the correctness of the inference algorithm. In the limit, the algorithm always returns a correct solution.

In the following Definition 7 we define precisely what we mean by the term *correct* implementation of the function Assumptions.

Algorithm 2: Implementation of Simplify(Φ)

Input: The set of instructions Φ.
Output: The simplified set of instructions Ψ.

```
Ψ ← ∅;
foreach φ = (x, z → t, y) ∈ Φ in some fixed order do
    if z ⊬*_Ψ t in the context (x, y) then
        Ψ ← Ψ ∪ {(x, z → t, y)};
    end
end
return Ψ;
```

Definition 7 *We call a function* Assumptions correct *with respect to the model of clearing restarting automata (correct with respect to k-cl-RA, for short), if the following holds:*

1. *For every set $S^+ \subseteq \Sigma^*$ the set of instructions $\Phi =$ Assumptions(S^+, l, k) is finite. Moreover, for every instruction $(x, z, y) \in \Phi : x \in LC_k, y \in RC_k, |z| > 0, |xzy| \leq l$.*
2. *For every k-cl-RA $M = (\Sigma, I)$, with the maximal width of instructions bounded from above by $l \geq 1$, there exists an equivalent k-cl-RA $N = (\Sigma, J)$, $J \subseteq I$, and a finite set $S_0^+ \subseteq L(N) = L(M)$, such that for every $S^+ \supseteq S_0^+$ the following holds: $J \subseteq$ Assumptions(S^+, l, k).*

The reason why we consider an equivalent automaton N in the second condition and do not state the above definition directly by using M is because M could contain also some useless instructions, i.e. instructions that M would never use in any accepting computation. We cannot expect from the function Assumptions to give us such useless instructions.

It is easy to see that similar definitions can be formulated also for other models, e.g. subword-clearing restarting automata. In the following we show some examples of functions Assumptions that are correct with respect to k-cl-RA.

Example 2 *The most trivial implementation of the function* Assumptions *is to return all possible instructions with the width bounded from above by l. It follows trivially that such a function is correct. However, the number of such instructions is in general exponential with respect to l, therefore such a function would be of little interest in real applications.*

Example 3 *Here we define two sample functions* Assumptions *correct with respect to k-cl-RA.*

1. Assumptions$_{cl1}(S^+, l, k) := \{(x, z, y) \mid x \in LC_k, y \in RC_k, |z| > 0, |xzy| \leq l$ *and* $\exists w_1, w_2 \in S^+ : xzy$ *is a subword of* ¢w_1\$ *and* xy *is a subword of* ¢w_2\$$\}$.

   The basic intuition behind this function is the assumption that if both patterns xzy and xy occur in the set of positive samples, then it is somehow justified to clear the word z based on the context (x, y). Note that the more we increase the length of contexts k the smaller (or equal) number of patterns we will find. The contexts serve here as a safety cushion against the inference of incorrect instructions.

2. $\mathsf{Assumptions}_{cl2}(S^+, l, k) := \{(x, z, y) \mid x \in LC_k, y \in RC_k, |z| > 0, |xzy| \leq l$ *and* $\exists w_1, w_2 \in S^+ : w_1 = \alpha z \beta, w_2 = \alpha\beta$, *$x$ is a suffix of $¢\alpha$ and y is a prefix of $\beta\$$}.

   This condition is even more restrictive than the previous one. It basically says that the instruction (x, z, y) is justified only in the case when there are positive samples $w_1, w_2 \in S^+$ such that we can obtain w_2 from w_1 by using this instruction.

Both of these functions can be computed in a polynomial time with respect to $\mathsf{size}(S^+) = \sum_{w \in S^+} |w|$. In fact, if l and k are fixed constants, then these functions can be computed in a linear time, since we need to consider only subwords of length bounded from above by the constant l. (See the technical report Černo (2012) for the implementation details).

In the following we prove the correctness of all functions Assumptions from Example 3 with respect to Definition 7. Since $\mathsf{Assumptions}_{cl2}(S^+, k, l) \subseteq \mathsf{Assumptions}_{cl1}(S^+, k, l)$, we only need to prove the correctness of the more restrictive function $\mathsf{Assumptions}_{cl2}$. The correctness of the less restrictive function $\mathsf{Assumptions}_{cl1}$ follows immediately. Let $M = (\Sigma, I)$ be any k-cl-RA with instructions of width at most $l \geq 1$, and let $N = (\Sigma, J)$ be any minimal k-cl-RA with respect to $|J|$ equivalent with M such that $J \subseteq I$. The minimality of N implies that for every instruction $\phi \in J$ there is a word $w_\phi \in L(N)$ such that the instruction ϕ is used in every accepting computation $w_\phi \vdash_N^* \lambda$. Without the loss of generality we may assume that the instruction ϕ must be used in the first step of an accepting computation for the word w_ϕ. (Note that there may also be some other instructions applicable to w_ϕ, but they definitely do not lead to any accepting computation). Let us fix for every $\phi \in J$ some accepting computation $w_\phi \vdash_N^{(\phi)} w'_\phi \vdash_N^* \lambda$. Now define $S_0^+ := \bigcup_{\phi \in J} \{w_\phi, w'_\phi\}$. Apparently $S_0^+ \subseteq L(N)$. Moreover, we can easily see that S_0^+ contains enough words to justify all instructions $\phi \in J$, i.e. $J \subseteq \mathsf{Assumptions}_{cl2}(S_0^+, l, k)$. The correctness follows easily from the monotonicity of the function $\mathsf{Assumptions}_{cl2}$ with respect to S^+ and the set inclusion relation. In general, however, $\mathsf{size}(S_0^+)$ is not polynomially bounded by $\mathsf{size}(N)$, i.e. it may happen that for some instructions $\phi \in J$ the length of the word w_ϕ is at least exponentially large with respect to the size of N, where the size of $N = (\Sigma, J)$ is the sum of widths of all its instructions, $\mathsf{size}(N) = \sum_{i \in J} |i|$.

The above examples can be easily extended to the model of k-scl-RA – instead of patterns xzy and xy we would consider the patterns xzy and xty, where t is a proper subword of z. We would basically get the same results as in the case of k-cl-RA.

In the following Theorem 8 we state our first positive result concerning the grammatical inference of clearing restarting automata. This theorem and its proof can be easily extended to subword-clearing restarting automata and other similar models.

Theorem 8 *Let a function* Assumptions *be correct with respect to k-cl-RA. Then, for every k-cl-RA $M = (\Sigma, I)$, with instructions of width at most $l \geq 1$, there exists a finite set of positive samples S_0^+ and a finite set of negative samples S_0^- consistent with M such that for every finite set of positive samples $S^+ \supseteq S_0^+$ and every finite set of negative samples $S^- \supseteq S_0^-$ consistent with M the algorithm $\mathsf{Infer}(S^+, S^-, l, k)$ will succeed and return an equivalent automaton k-cl-RA $N = (\Sigma, J)$ such that $L(N) = L(M)$.*

Proof Let $M = (\Sigma, I)$ be any k-cl-RA with instructions of width at most $l \geq 1$. According to Definition 7, there exist $J \subseteq I$ and $S_0^+ \subseteq L(M)$ such that k-cl-RA $N = (\Sigma, J)$ is equivalent

to M and for each $S^+ \supseteq S_0^+$ it holds $J \subseteq \mathsf{Assumptions}(S^+, l, k)$. We will show how to construct a finite set of negative samples S_0^- such that the algorithm $\mathsf{Infer}(S^+, S^-, l, k)$ will always succeed and return an automaton equivalent to N for every finite set of positive and negative samples $S^+ \supseteq S_0^+$, $S^- \supseteq S_0^-$ consistent with N. Let Φ denote the set of all instructions (x, z, y) such that $x \in LC_k, y \in RC_k, |z| > 0, |xzy| \leq l$. We say that the instruction $\phi \in \Phi$ is *bad* if there exist $w_- \notin L(N), w_+ \in L(N) : w_- \vdash^{(\phi)} w_+$. We say that the instruction ϕ is *disabled* by (S_0^+, S_0^-) if $\exists w_- \in S_0^-, w_+ \in S_0^+ : w_- \vdash^{(\phi)} w_+$. Now consider the following Algorithm 3:

Algorithm 3: Extension of Samples (S_0^+, S_0^-)

$S_0^- \leftarrow \emptyset$;
while $\exists$ *bad instruction* $\phi \in \Phi$ *such that* ϕ *is not disabled by* (S_0^+, S_0^-) **do**
  Let $w_- \notin L(N), w_+ \in L(N) : w_- \vdash^{(\phi)} w_+$;
  $S_0^+ \leftarrow S_0^+ \cup \{w_+\}$;
  $S_0^- \leftarrow S_0^- \cup \{w_-\}$;
end

Every added pair w_+, w_- effectively disables at least one instruction from Φ, so the Algorithm 3 is definitely finite. Now consider any finite set of positive samples $S^+ \supseteq S_0^+$ and any finite set of negative samples $S^- \supseteq S_0^-$ consistent with N. If we run the algorithm $\mathsf{Infer}(S^+, S^-, l, k)$, then in Step 1 we obtain some set of instructions covering all instructions from J. Note that no instruction from J is bad. In the following cycle the algorithm gradually removes all bad instructions. After this cycle we are left with correct instructions including all instructions from J, so the resulting automaton is apparently equivalent to the automaton N, and therefore also to the original target automaton M. ■

Example 4 *Consider the* 1-cl-RA $M = (\{a, b\}, \{(¢, ab, \$), (a, ab, b)\})$ *recognizing the language* $L(M) = \{a^n b^n \mid n \geq 0\}$ *(see Example 1). Let us take* $S^+ = \{\lambda, ab, aabb\}$. *First, we would like to estimate the set of instructions* $\Phi = \mathsf{Assumptions}_{cl1}(S^+, l, k)$ *for* $k = 1$ *and* $l = 6$ *(see Example 3 for definition). The set of all subwords of the delimited positive samples* $¢S^+\$$ *is:* $SW^+ = \{\lambda, ¢, a, b, \$, ¢\$, ¢a, aa, ab, bb, b\$, ¢aa, ¢ab, aab, abb, ab\$, bb\$, ¢ab\$, ¢aab, aabb, abb\$, ¢aabb, aabb\$, ¢aabb\$\}$. *An instruction* (x, z, y), *where* $x \in LC_1 = \{a, b, ¢\}$, $y \in RC_1 = \{a, b, \$\}$, $|z| > 0$ *and* $|xzy| \leq l$, *is justified, according to the definition of* $\mathsf{Assumptions}_{cl1}$, *if and only if both* $xzy, xy \in SW^+$. *Thus, only the following reductions are justified:* $¢\underline{a}a \vdash ¢a$, $a\underline{a}b \vdash ab$, $a\underline{b}b \vdash ab$, $b\underline{b}\$ \vdash b\$$, $¢\underline{ab}\$ \vdash ¢\$$, $a\underline{aabb} \vdash ab$, $¢\underline{aabb}\$ \vdash ¢\$$. *Therefore* $\mathsf{Assumptions}_{cl1}(S^+, l, k) = \{(¢, \underline{a}, a), (a, \underline{a}, b), (a, \underline{b}, b), (b, \underline{b}, \$), (¢, \underline{ab}, \$), (a, \underline{ab}, b), (¢, \underline{aabb}, \$)\}$. *Apparently, all instructions of* M *are included. However, the following instructions are bad:* $(¢, \underline{a}, a)$, $(a, \underline{a}, b)$, $(a, \underline{b}, b)$, $(b, \underline{b}, \$)$. *We can disable them easily by taking* $S^- = \{aab, abb\}$. *We do not need to add anything else to* S^+, *so the function* $\mathsf{Infer}(S^+, S^-, l, k)$ *will correctly output the automaton* $N = (\{a, b\}, \{(¢, \underline{ab}, \$), (a, \underline{ab}, b)\})$, *which is equivalent to the hidden target automaton* M. *The function* Simplify *removed the instruction* $(¢, aabb, \$)$ *from the inferred automaton as it can be simulated by the remaining two instructions.*

We should emphasize, that the set $S^- = \{aab, abb\}$ used in Example 4 is not big enough to guarantee the properties expressed in Theorem 8. If we take, for instance, a bigger set of positive samples, such as $S^+ = \{\lambda, ab, aabb, aaabbb\}$ then the function $\mathsf{Assumptions}_{cl1}(S^+, l, k)$ would give us a set of instructions containing a bad instruction $(a, \underline{aa}, b)$ not disabled by the pair (S^+, S^-). However, it can be shown that the following sets of positive and negative samples: $S_0^+ = \{a^n b^n \mid 0 \leq n \leq 6\}$ and $S_0^- = \{aab,\ abb,\ aaaab,\ abbb,\ aaaaab,\ aaabb,\ aabbb,\ abbbb,\ aaaaaab,\ aaaabb,\ aabbbb,\ abbbbb,\ aaaaabb,\ aabbbbb,\ aaaaaabb,\ aabbbbbb\}$ satisfy the properties of Theorem 8, for $k = 1$ and $l = 6$.

Also note that if we used the function $\mathsf{Assumptions}_{cl2}$ in Example 4 instead of the function $\mathsf{Assumptions}_{cl1}$ (see Example 3 for definition) we would need no negative samples at all.

In the following Example 5 we show how the inference algorithm can be used also for a more general subword-clearing restarting automata.

Example 5 *Consider the* 1-scl-RA $M = (\{a, b, c\}, \{(a, acb \to c, b), (¢, acb, \$)\})$ *recognizing the language* $L(M) = \{a^n c b^n \mid n > 0\} \cup \{\lambda\}$. *(We use the abbreviation* (x, z, y) *instead of* $(x, z \to \lambda, y)$*). This language cannot be recognized by any clearing restarting automaton, therefore we have to use the inference algorithm for subword-clearing restarting automata. We will use the following function* Assumptions*:* $\mathsf{Assumptions}_{scl1}(S^+, l, k) := \{(x, z \to t, y) \mid x \in LC_k, y \in RC_k, |z| > 0, |xzty| \leq l,\ t$ *is a proper subword of* z *and* $\exists w_1, w_2 \in S^+ :$ xzy *is a subword of* $¢w_1\$$ *and* xty *is a subword of* $¢w_2\$\}$. *It can be shown (similarly as in Example 3) that this function is correct with respect to the model of subword-clearing restarting automata. Let us take* $S^+ = \{\lambda, acb, aacbb\}$. *Then* $\mathsf{Assumptions}_{scl1}(S^+, l, k) =$ $\{(¢, a, a),\ (a, a, c),\ (b, b, \$),\ (c, b, b),\ (¢, aa \to a, c),\ (a, ac \to c, b),\ (c, bb \to b, \$),\ (a, cb \to c, b),$ $(¢, acb, \$),\ (a, acb \to c, b)\}$, *where* $k = 1$ *and* $l = 6$. *In this case, there are only two correct instructions:* $(¢, acb, \$)$ *and* $(a, acb \to c, b)$. *In order to filter all the other bad instructions, the following set of negative samples is sufficient:* $S^- = \{aacb, acbb\}$.

Similarly as in Example 4, the set $S^- = \{aacb, acbb\}$ is not big enough to guarantee the correctness of the inference algorithmin in all cases. However, it can be shown that the following sets of positive and negative samples: $S_0^+ = \{\lambda,\ acb,\ aacbb,\ aaacbbb,\ aaaacbbbb,\ aaaaacbbbbb,\ aaaaaacbbbbbb\}$ and $S_0^- = \{aacb,\ acbb,\ aaacb,\ acbbb,\ aaaacb,\ aaacbb,\ aacbbb,\ acbbbb,\ aaaaacb,\ aaaacbb,\ aacbbbb,\ acbbbbb,\ aaaaacbb,\ aaaacbbb,\ aaacbbbb,\ aacbbbbb,\ aaaaaacbb,\ aaaaacbbb,\ aaacbbbbb,\ aacbbbbbb\}$ meet our criteria stated in Theorem 8 (modified for the model of subword-clearing restarting automata), for $k = 1$ and $l = 6$.

In the following Example 6 we illustrate how to execute an active learning approach for the inference of a more realistic language of simplified arithmetical expressions.

Example 6 *Our goal is to infer a model of a subword-clearing restarting automaton recognizing the language of simplified arithmetical expressions over the alphabet* $\Sigma = \{a, +, (,)\}$, *where the letter* a *is used as a placeholder for any variable or numeric value. Correct arithmetical expressions are, for instance:* $a + (a + a)$, $(a + a)$, $((a))$ *etc., whereas the following expressions are all incorrect:* $a+$, $)a$, $(a + a$ *etc. For the sake of readability, we omit many cumbersome details, e.g. we do not list all the inferred instructions etc. We fix the length of contexts to* $k = 1$ *and the maximal width of instructions to* $l = 6$. *Let us start with some initial set of positive and negative samples* S_1^+ *and* S_1^-, *as in Table 1.*

Table 1: The Initial Set of Positive and Negative Samples.

Positive Samples S_1^+			**Negative Samples** S_1^-					
a	(a)	$((a+a))$	$+$	$a+$	$++$	$(+$	$)+$	$+a$
$a+a$	$((a))$	$a+(a+a)$	$($	$a($	$+($	$(($	$)($	$(a$
$a+a+a$	$(a+a)$	$(a+a)+a$	$)$	$a)$	$+)$	$()$	$))$	$)a$

These samples give us a good initial essence of what the correct arithmetical expressions might look like. The function $\mathsf{Assumptions}_{scl1}(S_1^+, l, k)$ *(see Example 5 for definition) gives us altogether* 64 *instruction candidates, where only two of them:* $[\text{ç}, (, a]$ *and* $[a,), \$]$ *will be filtered out due to the set of negative samples. The resulting automaton* M_1 *(our first hypothesis) is consistent with the given input set of positive and negative samples* S_1^+ *and* S_1^-*, and contains exactly* 21 *instructions after simplification (see Table 2). Note that here we use square brackets* [*and*] *instead of parentheses to border the instructions.*

Table 2: The Instructions of the Resulting Automaton M_1 After Simplification.

$[(, (, a]$	$[\text{ç}, (, (]$	$[+, (, a]$	$[),), \$]$	$[a,),)]$	$[a,), +]$	$[\text{ç}, a, \$]$
$[\text{ç}, ((, a]$	$[\text{ç}, (a \to a, +]$	$[a,)), \$]$	$[), +a, \$]$	$[a, +a, \$]$	$[a, +a,)]$	$[a, +a, +]$
$[+, a) \to a, \$]$	$[(, a+, a]$	$[\text{ç}, a+, (]$	$[\text{ç}, a+, a]$	$[+, a+, a]$	$[\text{ç}, (a), \$]$	$[\text{ç}, (a) \to a, \$]$

Let us generate some expressions recognized by this automaton M_1*. The following Table 3 lists the set of all expressions recognized by* M_1 *up to length* 5*.*

Table 3: The Set of Expressions Recognized by M_1.

λ	$\mathbf{a}+\mathbf{a}$	$\mathbf{a}+(\mathbf{a})$	$a)))$	$(a+a$	$((((a$
$\mathbf{a}$	$((a)$	$(((a$	$a))+a$	$\mathbf{a}+\mathbf{a}+\mathbf{a}$	$a))))$
$(\mathbf{a})$	$(a))$	$((\mathbf{a}))$	$a+a))$	$a+a)$	$a)+(a$
$((a$	$(\mathbf{a})+\mathbf{a}$	$((a+a$	$a+(a$	$(((a)$	$(a+(a$
$a))$	$(\mathbf{a}+\mathbf{a})$	$a+((a$	$a)+a$	$(a)))$	$a)+a)$

As we can see, there are both correct and incorrect arithmetical expressions (we have highlighted all the correct ones). Note that the automaton was able to recognize even some correct arithmetical expressions that it had not seen before, e.g. $a+(a)$*. Our next step will be to add the incorrect arithmetical expressions to the set of negative samples. Let* $S_2^+ := S_1^+$ *and let* S_2^- *be the set of all negative samples in* S_1^- *extended by the incorrect arithmetical expressions from Table 3. The inference algorithm returns on the input* (S_2^+, S_2^-, l, k) *a consistent automaton* M_2 *having* 16 *instructions after simplification.*

Up to length 5*, the automaton* M_2 *recognizes only correct arithmetical expressions. However, it recognizes also some incorrect arithmetical expressions beyond this length, e.g. the following expressions:* $((a+a), (a+a))$*,* $a+(a+a$ *and* $a+a)+a$*.*

Again, let $S_3^+ := S_2^+$ and let S_3^- be the set of all negative samples in S_2^- extended by the above four incorrect arithmetical expressions. This time the inference algorithm returns on the input (S_3^+, S_3^-, l, k) a consistent automaton M_3 having 12 instructions (see Table 4) that recognizes only correct arithmetical expressions. This can be verified easily by observing that every single instruction of the automaton M_3 preserves the correctness when applied to a correct arithmetical expression.

Table 4: The Instructions of the Resulting Automaton M_3 After Simplification.

$[¢, a, \$]$	$[), +a, \$]$	$[a, +a, \$]$	$[a, +a,)]$
$[a, +a, +]$	$[(, a+, a]$	$[¢, a+, (]$	$[¢, a+, a]$
$[+, a+, a]$	$[(, (a) \to a,)]$	$[¢, (a), \$]$	$[¢, (a) \to a, \$]$

Unfortunately, the automaton is not complete yet. It does not recognize, for instance, the following correct arithmetical expression: $a + (a + (a))$. This time we would need to extend the set of positive samples. We will not do it because our goal was only to sketch the principles of the active learning. We only mention that if we had chosen the maximal width of instructions to be $l = 5$ then we would not have been able to infer any subword-clearing restarting automaton recognizing the language of correct arithmetical expressions, because simply there is no such automaton.

4. Negative Results

In this section we show that, in general, the task of finding a consistent clearing restarting automaton with the given set of positive and negative samples is NP-hard, provided that we impose an upper bound on the width of instructions. First, we prove this result for the simplest case of 0-clearing restarting automata.

Theorem 9 *Let $l \geq 2$ be a fixed integer. Consider the following task: We are given a finite set of positive and negative samples: S^+, S^-, $S^+ \cap S^- = \emptyset$, $\lambda \in S^+$. Our task is to find a* 0-cl-RA *$M = (\Sigma, I)$ consistent with S^+ and S^- such that the width of instructions of M is at most l. This task is* NP*-complete.*

Proof Consider a 3-SAT formula $\psi = \bigwedge_{i=1}^{n} C_i$, where clause $C_i = \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$, and $\ell_{i,1}, \ell_{i,2}, \ell_{i,3}$ are literals having pairwise different variables, for all $i = 1, 2, \ldots, n$. Let $\Omega = \{a_1, a_2, \ldots, a_m\}$ be the set of all variables occurring in ψ. In the following, we specify a finite set of positive samples S^+ and a finite set of negative samples S^-, $S^+ \cap S^- = \emptyset$, $\lambda \in S^+$, such that the following holds: the formula ψ is satisfiable if and only if there exists a 0-cl-RA $M = (\Sigma, I)$ consistent with S^+ and S^-, such that the width of instructions of M is bounded from above by l.

Our alphabet will be $\Sigma = \Omega \cup \overline{\Omega}$, where $\overline{\Omega} = \{\overline{a_i} \mid a_i \in \Omega\}$, and $\Omega \cap \overline{\Omega} = \emptyset$. First set $S^+ := \{\lambda\}, S^- := \emptyset$. For each clause $C_i = \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$ add the negative sample $w_{C_i}^- = \overline{\ell_{i,1}}\ \overline{\ell_{i,2}}\ \overline{\ell_{i,3}}$ to the set of negative samples S^-. (We define $\overline{\overline{a}} = a$ for all $a \in \Omega$). For each variable $a \in \Omega$ add the following positive samples: $w_0^+ = (a\overline{a})^l$, $w_1^+ = a^l$, $w_2^+ = \overline{a}^l$ to the set of positive samples S^+. Finally, for each $a \in \Omega$ add all words $w \in \{a, \overline{a}\}^{\leq l}$, such

that $|w|_a \geq 1$ and $|w|_{\overline{a}} \geq 1$, to the set of negative samples S^-. Note that, for fixed l, there is only finite number of such words for every $a \in \Sigma$. Thus the size of the constructed set of positive and negative samples is, in fact, linear with respect to the size of the formula ψ.

($\Rightarrow$) Suppose that ψ is satisfiable, i.e. there exists an assignment $v : \Omega \to \{0,1\}$ such that $v^*(C_i) = 1$ for all $i \in \{1,2,\ldots,n\}$, where v^* denotes the natural extension of v to formulas. We will construct a 0-cl-RA $M = (\Sigma, I)$ consistent with S^+ and S^-, and with instructions of width at most l. Let $I = I_1 \cup I_2$, where $I_1 = \{(\lambda, a, \lambda) \mid a \in \Omega : v(a) = 1\} \cup \{(\lambda, \overline{a}, \lambda) \mid a \in \Omega : v(a) = 0\}$ and $I_2 = \{(\lambda, a^l, \lambda), (\lambda, \overline{a}^l, \lambda) \mid a \in \Omega\}$. It can be easily observed that, for each literal $\ell \in \Omega \cup \overline{\Omega} : \ell \vdash_M \lambda \Leftrightarrow v(\ell) = 1$, or equivalently: $\overline{\ell} \vdash_M \lambda \Leftrightarrow v(\ell) = 0$. Therefore no negative sample $w^-_{C_i} = \overline{\ell_{i,1}}\ \overline{\ell_{i,2}}\ \overline{\ell_{i,3}}$, for $i = 1,2,\ldots,n$, can be reduced to the empty word λ by using the instructions from I_1, because otherwise it would mean that $v(\ell_{i,1}) = v(\ell_{i,2}) = v(\ell_{i,3}) = 0$. As the literals used in $w^-_{C_i}$ have all pairwise different variables, no instruction from I_2 can be applied to it. Therefore, the resulting automaton M cannot reduce any negative sample of the form $w^-_{C_i} = \overline{\ell_{i,1}}\ \overline{\ell_{i,2}}\ \overline{\ell_{i,3}}$ to the empty word λ. Moreover, all positive samples of the form $w^+_0 = (a\overline{a})^l$, $w^+_1 = a^l$, $w^+_2 = \overline{a}^l$ can be reduced to the empty word λ. For each $a \in \Omega$ there is either the instruction (λ, a, λ), or the instruction $(\lambda, \overline{a}, \lambda)$ in I_1. Therefore, we can always reduce the positive sample $w^+_0 = (a\overline{a})^l$ either to the word a^l, or $\overline{a}^l$. After that we can use one of the instructions in I_2 to reduce it further to the empty word λ. Therefore, for each $a \in \Omega$: $(a\overline{a})^l \vdash^*_M \lambda$, and also trivially $a^l \vdash_M \lambda$, and $\overline{a}^l \vdash_M \lambda$. Finally, for each $a \in \Omega$ the word $w \in \{a, \overline{a}\}^{\leq l}$, such that $|w|_a \geq 1$ and $|w|_{\overline{a}} \geq 1$, cannot be reduced to the empty word λ. This is because there is only one of the instructions: (λ, a, λ), $(\lambda, \overline{a}, \lambda)$ available in I_1, and we will never be able to use any of the instructions: (λ, a^l, λ), $(\lambda, \overline{a}^l, \lambda)$ from I_2, since $|w|_a < l$ and $|w|_{\overline{a}} < l$.

($\Leftarrow$) Now suppose that there exists a 0-cl-RA $M = (\Sigma, I)$ consistent with S^+ and S^-, such that the width of instructions of M is bounded from above by l. We will show that ψ is satisfiable, i.e. we will construct an assignment $v : \Omega \to \{0,1\}$ such that $v^*(C_i) = 1$ for all $i \in \{1,2,\ldots,n\}$. First observe, that for each $a \in \Omega$: either $(\lambda, a, \lambda) \in I$, or $(\lambda, \overline{a}, \lambda) \in I$. Consider the positive sample $w^+_0 = (a\overline{a})^l \in S^+$. We know that $(a\overline{a})^l \vdash^*_M \lambda$. Let $\phi \in I$ be the first instruction used in any such accepting computation. The instruction ϕ is of the form (λ, z, λ), where z is a subword of the word $(a\overline{a})^l$. However, the only allowed options here are $\phi \in \{(\lambda, a, \lambda), (\lambda, \overline{a}, \lambda)\}$, because if $|z| > 1$, then we would immediately get that $|z|_a \geq 1$, $|z|_{\overline{a}} \geq 1$, and thus also $z \in S^-$, which is a contradiction to $z \vdash^{(\phi)} \lambda$. Moreover, both instructions (λ, a, λ) and $(\lambda, \overline{a}, \lambda)$ cannot be in I simultaneously, because it would mean that $a\overline{a} \vdash^*_M \lambda$, where $a\overline{a} \in S^-$. Now, for each $a \in \Omega$ let $v(a) = 1$ if $(\lambda, a, \lambda) \in I$, and let $v(a) = 0$ if $(\lambda, \overline{a}, \lambda) \in I$. For each clause $C_i = \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$ we have a negative sample $w^-_{C_i} = \overline{\ell_{i,1}}\ \overline{\ell_{i,2}}\ \overline{\ell_{i,3}} \in S^-$. Therefore, $\overline{\ell_{i,1}} \not\vdash_M \lambda$ or $\overline{\ell_{i,2}} \not\vdash_M \lambda$ or $\overline{\ell_{i,3}} \not\vdash_M \lambda$, which is equivalent to $v(\ell_{i,1}) = 1$ or $v(\ell_{i,2}) = 1$ or $v(\ell_{i,3}) = 1$. This means that ψ is satisfiable.

It remains to be shown that the task of finding a 0-cl-RA $M = (\Sigma, I)$ consistent with the given input set of positive and negative samples (S^+, S^-), such that the width of instructions of M is bounded from above by l, is in NP. In Černo and Mráz (2010) it was shown that every 1-cl-RA can be transformed in a polynomial time to an equivalent context-free grammar. Therefore, the membership problem for 0-cl-RA and 1-cl-RA is also decidable in a polynomial time. The next question is, how many instructions do we need? It turns out that the number of instructions can be bounded from above by $\mathsf{size}(S^+) = \sum_{w \in S^+} |w|$, because for every positive sample $w_+ \in S^+$ the accepting computation $w_+ \vdash^*_M \lambda$ uses at

most $|w_+|$ many instructions. Therefore, we can first nondeterministically guess the set of instructions, and then verify in a polynomial time the consistency with the given input set of positive and negative samples. ■

Theorem 9 can be generalized to cover all clearing restarting automata.

Theorem 10 *Let $k \geq 1$ and $l \geq 4k + 4$ be fixed integers. We are given finite sets of positive and negative samples: S^+, S^-, $S^+ \cap S^- = \emptyset$, $\lambda \in S^+$. Our task is to find a k-cl-RA $M = (\Sigma, I)$ consistent with S^+ and S^-, such that the width of instructions of M is bounded from above by l. This task is* NP*-complete for $k = 1$ and* NP*-hard for $k > 1$.*

Proof See the technical report Černo (2012). ■

It should be pointed out that if we do not impose any upper bound l on the maximal width of instructions, then the task is easily decidable in a polynomial time for any $k \geq 0$. Suppose that $S^+ = \{w_1, w_2, \ldots, w_n\}$. For $k = 0$ just take the instructions $I = \{(\lambda, w_1, \lambda), (\lambda, w_2, \lambda), \ldots, (\lambda, w_n, \lambda)\}$ and verify (in a polynomial time) the consistency with negative samples S^-. For $k \geq 1$ the task is even more trivial, just take the instructions $I = \{(¢, w_1, \$), (¢, w_2, \$), \ldots, (¢, w_n, \$)\}$ and verify that $S^+ \cap S^- = \emptyset$ and $\lambda \in S^+$.

5. Conclusion

We have shown that it is possible to identify any clearing and subword-clearing restarting automaton in the limit, which implies that we can learn a large class of languages in this way. On the other hand, the task of finding a clearing restarting automaton consistent with the given set of positive and negative samples is NP-hard, provided that we impose an upper bound on the width of instructions. This result resembles the famous result of Gold (1978) who showed that the question of whether there is a finite automaton with at most n states which agrees with a given list of input/output pairs is NP-complete. Indeed, for every n-state finite automaton there is an equivalent clearing restarting automaton that has the width of instructions bounded from above by $O(n)$ (Černo and Mráz, 2010). Without an upper bound the task becomes trivially solvable in a polynomial time both for finite automata and clearing restarting automata. However, it is an open problem, whether similar negative results hold also for other more powerful models, such as subword-clearing restarting automata. It would be also interesting to investigate the complexity of membership and equivalence queries for these models and to study grammatical inference for other related models.

Acknowledgments

I would like to thank Friedrich Otto and František Mráz for their support and careful proofreading of almost all revisions of this paper and for all suggestions and advices that significantly contributed to the quality of this paper.

References

M. Beaudry, M. Holzer, G. Niemann, and F. Otto. Mcnaughton families of languages. *Theoretical Computer Science*, 290(3):1581 – 1628, 2003.

Ronald V Book and Friedrich Otto. *String-rewriting systems*. Springer-Verlag, New York, NY, USA, 1993.

Peter Černo. Clearing restarting automata and grammatical inference. Technical Report 1/2012, Charles University, Faculty of Mathematics and Physics, Prague, 2012. URL http://popelka.ms.mff.cuni.cz/cerno/files/cerno_clra_and_gi.pdf.

Peter Černo and František Mráz. Clearing restarting automata. In Henning Bordinh, Rudolf Freund, Markus Holzer, Martin Kutrib, and Friedrich Otto, editors, *Workshop on Non-Classical Models for Automata and Applications (NCMA)*, volume 256 of *books@ocg.at*, pages 77–90. Österreichisches Computer Gesellschaft, 2009.

Peter Černo and František Mráz. Clearing restarting automata. *Fundamenta Informaticae*, 104(1):17–54, 2010.

Colin de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, New York, NY, USA, 2010.

Rémi Eyraud, Colin de la Higuera, and Jean-Christophe Janodet. Lars: A learning algorithm for rewriting systems. *Machine Learning*, 66:7–31, 2007.

E. Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.

John E. Hopcroft and J. D. Ullman. *Formal Languages and their Relation to Automata*. Addison-Wesley, Reading, 1969.

Petr Jančar, František Mráz, Martin Plátek, and Jörg Vogel. Restarting automata. In Horst Reichel, editor, *FCT'95*, volume 965 of *LNCS*, pages 283–292, Dresden, Germany, August 1995. Springer.

Steffen Lange, Thomas Zeugmann, and Sandra Zilles. Learning indexed families of recursive languages from positive data: A survey. *Theor. Comput. Sci.*, 397(1-3):194–232, May 2008.

Robert McNaughton. Algebraic decision procedures for local testability. *Theory of Computing Systems*, 8:60–76, 1974.

Friedrich Otto. Restarting automata. In Zoltán Ésik, Carlos Martín-Vide, and Victor Mitrana, editors, *Recent Advances in Formal Languages and Applications*, volume 25 of *Studies in Computational Intelligence*, pages 269–303. Springer, Berlin, 2006.

Yechezkel Zalcstein. Locally testable languages. *J. Comput. Syst. Sci*, 6(2):151–167, 1972.