CLEARING RESTARTING AUTOMATA AND GRAMMATICAL INFERENCE TECHNICAL REPORT

PETER ČERNO
PETERCERNO@GMAIL.COM

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 [13] 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 [16] 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. There were several attempts to learn their instructions by using genetic algorithms, but the results are far from being applicable.

Clearing restarting automata were introduced in [4, 5] 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 [5] it has been 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 also 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 [15, 17]. 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 ([11]) 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 $O(n)$ (see [5]).

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 [10], which are expressive enough to define a nontrivial class of languages containing all regular languages and some context-free languages. [10] 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 (see [9]) 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 deep 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 [10], the exact relationship between the corresponding classes of languages is not clear at all.

The paper has the following structure. In Section 2 we first introduce string-rewriting systems and then extend them to context rewriting systems. The definition of clearing and subword-clearing restarting automata can be easily obtained as a special kind of context rewriting systems. In Section 3 we give a description of the general setting for grammatical inference. Here we also introduce an important concept called identification in the limit. In Section 4 we propose a learning schema for clearing and subword-clearing restarting automata. We show that it is possible, by using this schema, to identify any clearing and subword-clearing restarting automaton in the limit. In Section 5 we prove that the task of finding a consistent clearing restarting automaton with the given set of positive and negative samples is, in general, NP-hard, provided that we impose an upper bound on the width of instructions. Without an upper bound the problem is trivially solvable in a polynomial time. Conclusions are presented in Section 6. An implementation of the presented algorithms can be found on the following website: `http://code.google.com/p/clearing-restarting-automata/`.

2. Theoretical Background

In this section we follow the approach used in the book [2] by first introducing the abstract reduction systems and then defining string-rewriting systems as their special case. For brevity we omit the proofs.

Definition 2.1 ([2]). *Let $\mathbf{B}$ be a set of objects and let $\rightarrow$ be a binary relation on $\mathbf{B}$. Let $\rightarrow^{-1}$ be the inverse of $\rightarrow$, and let $\circ$ denote composition of relations.*

(1) $\rightarrow^0$ *is the identity relation.*
(2) $\rightarrow^n = \rightarrow^{n-1} \circ \rightarrow$.
(3) $\rightarrow^* = \bigcup_{n\geq 0} \rightarrow^n$ *and* $\rightarrow^+ = \bigcup_{n>0} \rightarrow^n$.
(4) $\leftrightarrow = \rightarrow \cup \rightarrow^{-1}$.
(5) $\leftrightarrow^n = \leftrightarrow^{n-1} \circ \leftrightarrow$.
(6) $\leftrightarrow^* = \bigcup_{n\geq 0} \leftrightarrow^n$ *and* $\leftrightarrow^+ = \bigcup_{n>0} \leftrightarrow^n$.

The relation $\rightarrow^*$ is reflexive and transitive, and the relation $\leftrightarrow^*$ is an equivalence relation on $\mathbf{B}$. In fact, it is the smallest equivalence relation on $\mathbf{B}$ that contains $\rightarrow$.

Definition 2.2 (Reduction systems [2]). *Let $\mathbf{B}$ be a set of objects and let $\rightarrow$ be a binary relation on $\mathbf{B}$.*

(1) *The structure $S = (\mathbf{B}, \rightarrow)$ is a* reduction system *and the relation $\rightarrow$ is the* reduction relation.
(2) *If $x \in \mathbf{B}$ and there is no $y \in \mathbf{B}$ such that $x \rightarrow y$, then x is* irreducible*; otherwise, x is* reducible. *The set of all irreducible elements of $\mathbf{B}$ with respect to $\rightarrow$ is denoted* $\mathsf{IRR}(S)$.

Definition 2.3 ([2]). *Let $(\mathbf{B}, \rightarrow)$ be a reduction system. If $x, y \in \mathbf{B}$ and $x \rightarrow^* y$, then x is an* ancestor *to y and y is a* descendant *of x. If $x, y \in \mathbf{B}$ and $x \leftrightarrow^* y$, then x and y are* equivalent.

Definition 2.4 ([2]). *Let $(\mathbf{B}, \rightarrow)$ be a reduction system. For $x, y \in \mathbf{B}$, if $x \leftrightarrow^* y$ and y is irreducible, then y is a* normal form *for x.*

Suppose that for every object in $\mathbf{B}$ there is a unique normal form. Then for every $x, y \in \mathbf{B}$, $x \leftrightarrow^* y$ if and only if the normal form of x is identically equal to the normal form of y. Now we consider conditions that guarantee the existence of unique normal forms in the abstract setting.

Definition 2.5 ([2])**.** *Let $S = (\mathbf{B}, \to)$ be a reduction system.*

(1) *S is* confluent *if for all $w, x, y \in \mathbf{B}$, $w \to^* x$ and $w \to^* y$ imply that there exists a $z \in \mathbf{B}$, $x \to^* z$ and $y \to^* z$.*
(2) *S is* locally confluent *if for all $w, x, y \in \mathbf{B}$, $w \to x$ and $w \to y$ imply that there exists a $z \in \mathbf{B}$, $x \to^* z$ and $y \to^* z$.*
(3) *S has the* Church-Rosser property *if for all $x, y \in \mathbf{B}$, if $x \leftrightarrow^* y$, there exists a $z \in \mathbf{B}$, such that $x \to^* z$ and $y \to^* z$.*

If we say that a system "is Church-Rosser" we mean that it has the Church-Rosser property.

Lemma 2.1 (Relating confluence and Church-Rosser property [2])**.** *Let $S = (\mathbf{B}, \to)$ be a reduction system. Then S is Church-Rosser if and only if it is confluent.*

Corollary 2.1 ([2])**.** *Let $S = (\mathbf{B}, \to)$ be a reduction system that is confluent. Then for each $x \in \mathbf{B}$, $[x]$ has at most one normal form.*

Definition 2.6 ([2])**.** *Let $(\mathbf{B}, \to)$ be a reduction system. The relation $\to$ is* noetherian *if there is no infinite sequence $x_0, x_1, \ldots \in \mathbf{B}$ such that for all $i \geq 0$, $x_i \to x_{i+1}$.*

Lemma 2.2 ([2])**.** *Let $(\mathbf{B}, \to)$ be a reduction system. If $\to$ is noetherian, then for every $x \in \mathbf{B}$, $[x]$ has a normal form.*

In this paper the definition of reduction will always satisfy the properties (1) of being acyclic and (2) being such that for every x, the set of descendants of x is finite. Under these conditions the reduction is noetherian.

Definition 2.7 ([2])**.** *If $S = (\mathbf{B}, \to)$ is a reduction system such that S is confluent and $\to$ is noetherian, then S is* convergent.

Theorem 2.1 ([2])**.** *Let $S = (\mathbf{B}, \to)$ be a reduction system. Suppose that $\to$ is noetherian. Then S is confluent if and only if S is locally confluent.*

Now we move from the general setting of reduction systems to a more specific setting of string-rewriting systems, where the set $\mathbf{B}$ will usually be a set of words. We use the standard notation from the theory of automata and formal languages. As our reference concerning this field we use the monograph [12].

An *alphabet* is a finite nonempty set. The elements of an alphabet Σ are called *letters* or *symbols*. A *word* or *string* over an alphabet Σ is a finite sequence consisting of zero or more letters of Σ, whereby the same letter may occur several times. The sequence of zero letters is called the *empty word*, written λ. The set of all words (all nonempty words, respectively) over an alphabet Σ is denoted by Σ^* (Σ^+, respectively). If x and y are words over Σ, then so is their *catenation* (or *concatenation*) xy (or $x \cdot y$), obtained by juxtaposition, that is, writing x and y one after another. Catenation is an associative operation and the empty word λ acts as an identity: $w\lambda = \lambda w = w$ holds for all words w. Because of the associativity, we may use the notation w^i in the usual way. By definition, $w^0 = \lambda$.

Let u be a word in Σ^*, say $u = a_1 \ldots a_n$ with $a_i \in \Sigma$. We say that n is the *length* of u and we write $|u| = n$. The sets of all words over Σ of length k, or at most k, are denoted by Σ^k and $\Sigma^{\leq k}$, respectively. By $|u|_a$, for $a \in \Sigma$, we denote the total number of occurrences of the letter a in u. The *reversal* (*mirror image*) of u, denoted u^R, is the word $a_n \ldots a_1$. Finally a *factorization* of u is any sequence u_1, ..., u_t of words such that $u = u_1 \cdots u_t$.

For a pair u, v of words we define the following relations:

(1) u is a *prefix* of v, if there exists a word z such that $v = uz$,
(2) u is a *suffix* of v, if there exists a word z such that $v = zu$, and
(3) u is a *factor* (or *subword*) of v, if there exist words z and z' such that $v = zuz'$.

Observe that u itself and λ are subwords, prefixes and suffixes of u. Other subwords, prefixes and suffixes are called *proper*.

Subsets, finite or infinite, of Σ^* are referred to as (*formal*) *languages* over Σ.

In formal language theory in general, there are two major types of mechanisms for defining languages: *acceptors* and *generators*. Acceptors are usually defined in terms of *automata*, which work as follows: they are given an input word and after some processing they either accept or reject this input word. For instance, the so-called *finite automata* consist of a finite set of internal states and a set of rules that govern the change of the current state when reading a given input symbol. The finite automaton reads a given input word from left to right starting in a specific *starting state*. After reading the input word it accepts only if it ends in a so-called *accepting state*, otherwise it rejects. Finite automata recognize the family of *regular languages*, which plays a central role in the whole formal language theory.

Generators, on the other hand, usually generate the language using some finite set of rules. Typically they are defined in terms of grammars. One of the most famous is the classical *Chomsky hierarchy* of grammars (and corresponding languages), which consists of *phrase-structure*, *context-sensitive*, *context-free*, and *regular* grammars (they are also called type 0, type 1, type 2, and type 3 grammars, respectively).

In this paper we will be interested mainly in the so-called *string-rewriting systems*, which are somehow on the edge between acceptors and generators.

Definition 2.8 (String-rewriting systems [2]). *Let Σ be a finite alphabet.*

(1) *A* string-rewriting system *R on Σ is a subset of $\Sigma^* \times \Sigma^*$. Each element (l, r) of R is a* (rewrite) rule. *The set $\{l \in \Sigma^* \mid \exists r \in \Sigma^* : (l, r) \in R\}$ is called the* domain *of R and is denoted $\mathsf{dom}(R)$. The set $\{r \in \Sigma^* \mid \exists l \in \Sigma^* : (l, r) \in R\}$ is called the* range *of R and is denoted $\mathsf{rng}(R)$. If R is finite, then the* size *of R is defined to be $\sum_{(l,r)\in R}(|l| + |r|)$ and is denoted $\|R\|$. The* width *of rule $(l, r) \in R$ is $|l| + |r|$.*
(2) *If R is a string-rewriting system on Σ, then the* single-step reduction relation *on Σ^*, that is induced by R is defined as follows: for any $u, v \in \Sigma^*$, $u \to_R v$ if and only if there exists $(l, r) \in R$ such that for some $x, y \in \Sigma^*$, $u = xly$ and $v = xry$.*

The reduction relation *on Σ^* induced by R is the reflexive and transitive closure of $\to_R$ and is denoted by $\to_R^*$.*

If R is a string-rewriting system on Σ, then $(\Sigma^*, \to_R)$ is a reduction system. We will frequently use R, as opposed to $(\Sigma^*, \to_R)$, when a reduction system is considered.

(3) *The* Thue congruence *generated by R is the relation $\leftrightarrow_R^*$.*
(4) *Two strings $u, v \in \Sigma^*$ are congruent modulo R if $u \leftrightarrow_R^* v$. For each $w \in \Sigma^*$, $[w]_R$ is called the* congruence class *of w modulo R.*

Lemma 2.3 ([2])**.** *If R is a finite string-rewriting system on alphabet Σ, then the set $\mathsf{IRR}(R)$ of irreducible strings with respect to R is a regular set; furthermore, a finite automaton for $\mathsf{IRR}(R)$ can be constructed in a polynomial time from R.*

Definition 2.9. *Let R be a string-rewriting system on Σ. We say that R is:*

(1) length-reducing, *if for each rule $(l, r) \in R : |l| > |r|$.*
(2) non-increasing, *if for each rule $(l, r) \in R : |l| \geq |r|$.*
(3) weight-reducing, *if there exists a so-called* weight function $f : \Sigma \to \mathsf{N}$, *such that for each rule $(l, r) \in R : f^*(l) > f^*(r)$, where $f^* : \Sigma^* \to \mathsf{N}$ is defined inductively as: $f^*(\lambda) = 0$, $f^*(xa) = f^*(x) + f(a)$ for all $x \in \Sigma^*, a \in \Sigma$.*

Note that the length-reducing string-rewriting systems represent only a special case of the more general weight-reducing string-rewriting systems. To see this just consider the following weight function: $f(a) := 1$ for all $a \in \Sigma$.

Also note that every weight-reducing string-rewriting system is noetherian.

In the literature string-rewriting systems are also known as *semi-Thue systems*. A string-rewriting system R with the property that $(l, r) \in R$ implies $(r, l) \in R$ is also called a *Thue system*. For a Thue system R, the single-step reduction relation $\to_R$ is symmetric, so that the reduction relation $\to_R^*$ coincides with the Thue congruence $\leftrightarrow_R^*$.

The reason that the relation $\leftrightarrow_R^*$ is called a "congruence" relation is that it is an equivalence relation that is compatible with respect to concatenation of strings.

String-rewriting systems play a central part in the definition of the so called Church-Rosser languages. A language $L \subseteq \Sigma^*$ is called a Church-Rosser language, if it consists of those strings $w \in \Sigma^*$ that, placed in the context $t_1 w t_2$ of certain auxiliary strings t_1 and t_2, reduce to a certain "accepting" symbol with respect to a finite, length-reducing and confluent string-rewriting system. Apart from the final symbol and the symbols occurring in the contexts t_1 and t_2, also other non-terminal symbols are allowed in this definition. Although the rewriting process with respect to the string-rewriting system considered is inherently non-deterministic, the confluence of the system ensures that each reduction sequence will lead to the same result.

The natural generalization of the above definition led to the development of the broad concept of the so-called McNaughton families [1]. Let $\mathcal{S}$ be a class of string-rewriting systems. Then $\mathcal{S}$ yields a family of languages $\mathcal{L}(\mathcal{S})$, which we call the *McNaughton family of languages* specified by $\mathcal{S}$, and which we define as follows. A language $L \subseteq \Sigma^*$ belongs to $\mathcal{L}(\mathcal{S})$, if there exists a finite alphabet Γ strictly containing Σ, a finite string-rewriting system $R \in \mathcal{S}$ on Γ, strings $t_1, t_2 \in (\Gamma \setminus \Sigma)^* \cap \mathsf{IRR}(R)$ and a letter $Y \in (\Gamma \setminus \Sigma) \cap \mathsf{IRR}(R)$ such that, for all $w \in \Sigma^*$: $w \in L \Leftrightarrow t_1 w t_2 \to_R^* Y$. Here the symbols of Σ are *terminals*, while those of $\Gamma \backslash \Sigma$ can be seen as *nonterminals*. The language L is said to be *specified* by the four-tuple (R, t_1, t_2, Y) and this fact will be expressed as $L = L(R, t_1, t_2, Y)$. By placing various restrictions on the finite string-rewriting systems used we will obtain different families of languages. We refer the interested reader to the article [1] where these families are studied in detail.

In this paper we follow a similar approach, but instead of using general strings $t_1, t_2 \in (\Gamma \setminus \Sigma)^*$ we will use only the single letters $t_1 = ¢$, $t_2 = \$$, called the sentinels, and instead of the symbol Y we will use the empty word. In the following we introduce a concept called *context rewriting system* which will serve us as a framework for clearing and subword-clearing restarting automata and also other similar models.

Definition 2.10 ([5]). *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*, $x \in LC_k = \Gamma^k \cup \{¢\} \cdot \Gamma^{\le k-1}$, y is called the* right context*, $y \in RC_k = \Gamma^k \cup \Gamma^{\le k-1} \cdot \{\$\}$ 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 $v \cdot r$. Each definition that uses the rewriting relation $\vdash_M$ can be relativized to any context (l, 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 2.1. *We also include a special case $k = 0$ in Definition 2.10. In this case we define $LC_0 = RC_0 = \{\lambda\}$, and the rest of the definition remains the same.*

Remark 2.2. *We also extend Definition 2.10 with the following set notation: if $X \subseteq LC_k$ and $Y \subseteq RC_k$ are finite nonempty sets, and Z is a finite nonempty set of instruction-rules of the form $z \to t$, $z, t \in \Gamma^*$, then we define $(X, Z, Y) = \{(x, z \to t, y) \mid x \in X, (z \to t) \in Z, y \in Y\}$. However, if $X = \{x\}$, then instead of writing $(\{x\}, Z, Y)$ we write only (x, Z, Y) for short. The same holds for the sets Z and Y, too.*

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

Lemma 2.4 (Error Preserving Property, [16])**.** *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 $v \notin L(M)$.*

Naturally, if we increase the length of contexts used in instructions of a CRS, we do not decrease their expressiveness.

Theorem 2.2 (Context extension theorem [5])**.** *For each k-CRS $M = (\Sigma, \Gamma, I)$ there exists a $(k+1)$-CRS $M' = (\Sigma, \Gamma, I')$ such that, for each $w, w' \in \Gamma^*$, it holds $w \vdash_M w' \Leftrightarrow w \vdash_{M'} w'$. Moreover, both M and M' use the same set of instruction-rules:*

$$\{z \to t \mid (x, z \to t, y) \in I\} = \{z' \to t' \mid (x', z' \to t', y') \in I'\} \ .$$

Remark 2.3. *Based on the above result, in Definition 2.10 we can allow contexts of any length up to k, i.e. we can use:*
$LC_{\le k} = \Gamma^{\le k} \cup ¢ \cdot \Gamma^{\le k-1} = \bigcup_{i \le k} LC_i$ *instead of* LC_k *and*
$RC_{\le k} = \Gamma^{\le k} \cup \Gamma^{\le k-1} \cdot \$ = \bigcup_{i \le k} RC_i$ *instead of* RC_k.

It is clear from Definition 2.10 that context rewriting systems are very similar to string-rewriting systems. Suppose that $M = (\Sigma, \Gamma, I)$ is a k-CRS and $Y \notin \Gamma$. Let us define a string-rewriting system $R(M) = \{(xzy, xty) \mid (x, z \to t, y) \in I\} \cup \{(¢\$, Y)\}$. Apparently $L(M) = L(R(M), ¢, \$, Y)$. This basically allows us to extend most of the terminology concerning string-rewriting systems also to our context rewriting systems. We say that a k-CRS M is confluent (locally confluent, Church-Rosser, respectively) if the corresponding string-rewriting system $R(M)$ is confluent (locally confluent, Church-Rosser, respectively). Analogously, we say that a k-CRS M is length-reducing (non-increasing, weight-reducing, noetherian, respectively) if the reduction relation $\to_R$ of the string-rewriting system $R = R(M)$ is length-reducing (non-increasing, weight-reducing, noetherian, respectively).

It is easy to see that general k-CRS can simulate any type 0 grammar (according to the Chomsky hierarchy [12]). Hence we will not study k-CRS in their general form, since they are too powerful (they can represent all recursively enumerable languages). Instead, we will always put some restrictions on the instruction-rules and then study such restricted models. In particular, we will 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 2.11 ([5])**.** *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 2.4. *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 $L(M) = L \cup \{\lambda\}$.*

This implicit acceptance of the empty word can be avoided by a slight modification of the definition of clearing restarting automata, or even context rewriting systems, but in principle, we would not get a more powerful model.

Example 2.1. *Let $M = (\Sigma, I)$ be a* 1-cl-RA *with $\Sigma = \{a, b\}$ and I consisting of the following two instructions:*

$$\begin{array}{ll} (1) & (a, ab, b), \\ (2) & (¢, ab, \$). \end{array}$$

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 [5]. We only mention that they can recognize all regular languages, some context-free languages and even some non-context-free languages. However, there are some context-free languages that are outside the class of languages accepted by clearing restarting automata.

Theorem 2.3 ([5])**.** *The language $L = \{a^n c b^n \mid n \geq 0\}$ is not recognized by any k-*cl-RA*.*

The above limitation led to the development of the extended versions of clearing restarting automata. In [5] there were introduced two extended versions – the so-called Δ-clearing restarting automata and Δ^*-clearing restarting automata. Both of them can use a single auxiliary symbol Δ only. Δ-clearing restarting automata can leave a mark – a symbol Δ – at the place of deleting besides rewriting into the empty word λ. Δ^*-clearing restarting automata can rewrite a subword w into Δ^k where k is bounded from above by the length of w. It was shown in [5] that Δ^*-clearing restarting automata are powerful enough to recognize all context-free languages. This result was later extended in [6, 7] to hold also for the more restricted Δ-clearing restarting automata.

Here we propose yet another model, the so-called subword-clearing restarting automata, which will be useful later in some grammatical inference scenarios.

Definition 2.12. *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.*

It can be easily shown that $L = \{a^n c b^n \mid n \geq 0\}$ can be recognized by a subword-clearing restarting automaton. However, not all context-free languages can be recognized by these automata. Consider for instance the language $\{ww^R \mid w \in \Sigma^*\}$.

As we already stated, we will be mostly interested in grammatical inference for clearing restarting automata. Therefore, in the following section we introduce the general setting for learning such models.

3. General Setting

The main source for this section is [9], which provides a nice and comprehensive survey of the techniques and results concerning grammatical inference. In the following we fix the alphabet Σ.

There are some problems whose tractability is of great importance in grammatical inference. Let $\mathcal{L}$ be a language class, $\mathcal{G}$ be a class of representation of objects for $\mathcal{L}$ and $L : \mathcal{G} \to \mathcal{L}$ be the *naming function*, i.e. $L(G)$ is the language denoted, accepted, recognized or represented by $G \in \mathcal{G}$. We can imagine $\mathcal{G}$ to be, for instance, the class of all clearing restarting automata, or the class of all subword-clearing restarting automata.

The first problem concerns the fact whether the following *membership problem* is decidable: given $w \in \Sigma^*$ and $G \in \mathcal{G}$, is $w \in L(G)$? For instance, for context-free grammars the membership problem is decidable in a polynomial time. In [5] we have proved that every $\mathsf{1\text{-}cl\text{-}RA}$ can be transformed in a polynomial time to an equivalent context-free grammar. Therefore, the membership problem for $\mathsf{0\text{-}cl\text{-}RA}$ and $\mathsf{1\text{-}cl\text{-}RA}$ is also decidable in a polynomial time. For other models it is an open problem. Nevertheless, every length-reducing (weight-reducing, respectively) context rewriting system M can be easily transformed into an equivalent growing context-sensitive grammar. This follows easily from the fact that the corresponding string-rewriting system $R(M)$ is length-reducing (weight-reducing, respectively). It is a well-known fact that the membership problem for any given fixed growing context-sensitive grammar is decidable in a polynomial time [8]. However, it can be also shown that the membership problem is NP-complete if the growing context-sensitive grammar is a part of the input [3]. If we restrict ourselves only to confluent length-reducing context rewriting systems, then the membership problem becomes trivially decidable in a polynomial time, because it does not matter which particular instruction we use in any step of the computation. All reduction paths starting from the same given input word w eventually end (after at most $|w|$ many steps) at the same irreducible word.

The second problem is the *equivalence problem*: given $G, G' \in \mathcal{G}$, do we have $L(G) = L(G')$? For context-free grammars, the equivalence problem is undecidable. It is decidable for finite automata, but the complexity depends on whether the automata are deterministic or not. It is an open problem whether the equivalence problem is decidable for clearing restarting automata.

In machine learning, however, we encounter also more difficult problems with no clear or established notion. How do we know that the method or algorithm in use is able to infer a reasonable model for our target language? The trick we will use is to consider that the problem we are really interested in is not about discovering the model that would explain the data, but about identifying the *hidden target model*. An alternative formalism for convergence may be that there is no target: the idea is just to induce a grammar from the data in such a way as to minimize some statistical criterion. But again, whether explicitly or implicitly, there is somewhere, hidden, an ideal solution that we can call a target. This discussion leads us to an important notion of the so-called *identification in the limit*. We will not delve into the technical details of this notion, but only sketch informally the basic idea.

A *presentation* ϕ is an enumeration of elements, which represents a source of information about some specific language $L \in \mathcal{L}$. An example of a presentation can be, for instance, the enumeration of all positive and negative samples of L (in some order). A *learning algorithm* $\mathbf{A}$ is a program that takes the first n elements of a presentation (denoted as ϕ_n) and returns some object $G \in \mathcal{G}$. We say that $\mathcal{G}$ is identifiable in the limit if there exists a learning algorithm $\mathbf{A}$ such that for any target object $G \in \mathcal{G}$ and any presentation ϕ of $L(G)$ there exists a rank n such that for all $m \geq n$, $\mathbf{A}(\phi_m)$ does not change and $L(\mathbf{A}(\phi_m)) = L(G)$. Notice that the above definition does not force us to learn the target object G, but only to learn an object equivalent to the target. However, there are some complexity issues with the identification in limit, since it neither tells us how we know when we have found what we are looking for nor how long it is going to take.

Another methodology in machine learning is the so-called *active learning*. In the previous formalism the presentations were basically uncontrolled by the learner. The most we could hope for was a correctly labeled data and that no essential piece of data was missing in the limit. There are nevertheless cases where the ability to control the data we receive further is at least desirable.

4. Learning Schema

In this section we propose a learning schema for clearing (subword-clearing) restarting automata and other similar models and show that it is possible to identify any hidden target model in the limit in this way. It is rather a schema than a precise algorithm, because some details are not completely specified and are left to be adapted according to the specific situation or, for instance, to other different models. However, the proofs will be based only on the weak assumptions of the general schema, and thus will work also for all its possible instances. In the following, the term automaton refers primarily to the clearing or subword-clearing restarting automaton, but in general also to any other similar noetherian model obtained from context rewriting systems by restricting its instruction-rules.

The problem we are interested in can be best described as follows. Suppose that we have two finite sets of words over the 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 may assume that $S^+ \cap S^- = \emptyset$ and $\lambda \in S^+$.

If we have no other restrictions, then the task becomes trivial even for clearing restarting automata. Just consider the instructions $I = \{(¢, w, \$) \mid w \in S^+, w \neq \lambda\}$. It follows trivially, that in this case $L(M) = S^+$, where $M = (\Sigma, I)$. Therefore, it is reasonable to include some other restrictions. There are many options available: we can impose an upper bound on the number of instructions, or we can restrict the maximal width of instructions etc. In the following, we will consider only such requirements, that make the task somehow non-trivial and at the same time allow only finite many automata satisfying these requirements. From this point of view, it is not very reasonable to control only the maximal number of instructions, since there is an infinite number of different automata having just one single instruction. Moreover, it is useless to consider too long instructions, since they might not even be applicable to any given positive or negative sample.

Therefore, we impose the maximal allowed width $l \geq 1$ and also the 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. [14] 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: Learning schema $\mathsf{Infer}(S^+, S^-, l, k)$

```
Input  : The set of positive $S^+$ and negative $S^-$ samples over $\Sigma$, $S^+ \cap S^- = \emptyset$,
         $\lambda \in S^+$. 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 $\Phi \leftarrow \mathsf{Assumptions}(S^+, l, k)$;
2 while $\exists w_- \in S^-, w_+ \in S^+, \phi \in \Phi : w_- \vdash^{(\phi)} w_+$ do
3     $\Phi \leftarrow \Phi \setminus \{\phi\}$;
4 $\Phi \leftarrow \mathsf{Simplify}(\Phi)$;
5 if $\mathsf{Consistent}(\Phi, S^+, S^-)$ then
6     return Automaton with the set of instructions $\Phi$;
7 Fail;
```

Algorithm 1 deserves some explanation. First, the function $\mathsf{Assumptions}(S^+, l, k)$ 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 all instructions that allow reduction from some negative sample to some positive sample, i.e. they violate the error preserving property (Lemma 2.4). In Step 4 we remove redundant instructions and in Step 5 we check if the remaining set of instructions is consistent with the given input set of positive and negative samples. In other words, we check if (1) for all $w_+ \in S^+ : w_+ \vdash_\Phi^* \lambda$ and (2) for all $w_- \in S^- : w_- \not\vdash_\Phi^* \lambda$. The condition (1) always holds, if we assume that in Step 1 we already obtained all instructions of the hidden target automaton. However, the condition (2) may fail. It may happen, that for some $w_- \in S^-$ and $w_+ \in S^+$ we get $w_- \vdash_\Phi^* w_+$. In other words, there may exist a sequence of words and instructions from Φ such that: $w_- = w_1 \vdash^{(\phi_1)} w_2 \vdash^{(\phi_2)} w_3 \vdash^{(\phi_3)} \dots w_n \vdash^{(\phi_n)} w_{n+1} = w_+$, where $n \geq 2$. One of the instructions $\phi_1, \dots, \phi_n$ definitely does not belong to the hidden target automaton, but it is not clear which one. The success of the above algorithm, therefore, 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 $\mathsf{Assumptions}$, then there is always a set of positive samples S_0^+ and a set of negative samples S_0^- such that the above schema converges to a correct solution for all sets of positive samples $S^+ \supseteq S_0^+$ and negative samples $S^- \supseteq S_0^-$. This also implies that we can infer a correct solution in the limit from any reasonable presentation of labeled samples, where the term reasonable means that the presentation will at some point cover all the samples from S_0^+ and S_0^-.

The time complexity of Algorithm 1 depends both on the time complexity of the function $\mathsf{Assumptions}$ in Step 1 and on the time complexity of the simplification function and the consistency check in Steps 4 and 5. There are correct implementations of the function $\mathsf{Assumptions}$ (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 $\mathsf{Assumptions}$

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 $\mathsf{Simplify}$ 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.

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 4.1 we define precisely what we mean by the term correct implementation of the function $\mathsf{Assumptions}$.

Definition 4.1. *We call the function* $\mathsf{Assumptions}$ correct *with respect to the model of clearing restarting automata, if the following conditions hold:*

1. *For every set $S^+ \subseteq \Sigma^*$ the set of instructions $\Phi = \mathsf{Assumptions}(S^+, l, k)$ 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 $S_0^+ \subseteq L(N)$, such that for every $S^+ \supseteq S_0^+$ the following holds: $J \subseteq \mathsf{Assumptions}(S^+, l, k)$.*

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 reasonably expect from the function $\mathsf{Assumptions}$ to give us these useless instructions. Without loss of generality, we may assume that $N = (\Sigma, J)$ is a minimal k-cl-RA (with respect to $|J|$) equivalent with $M = (\Sigma, I)$, such that $J \subseteq I$.

It can easily be seen that similar definitions can be formulated also for other models, e.g. subword-clearing restarting automata etc. In the following we will show some examples of correct functions $\mathsf{Assumptions}$ for clearing restarting automata.

Example 4.1. *The most trivial implementation of the function* $\mathsf{Assumptions}$ *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 4.2. *Here we define several functions* $\mathsf{Assumptions}$ *that are all correct with respect to Definition 4.1.*

1. $\mathsf{Assumptions}_{cl1}(S^+, l, k) := \{(x, z, y) \mid x \in LC_k, y \in RC_k, |z| > 0, |xzy| \leq l$ *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) the number of such patterns we will *find. The contexts serve here as a safety cushion against the inference of incorrect instructions.*

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

This is the same situation as before, except that in addition we require that the patterns xzy and xy occur in different positive samples.

(3) $\mathsf{Assumptions}_{cl3}(S^+, l, k) := \{(x, z, y) \mid x \in LC_k, y \in RC_k, |z| > 0, |xzy| \le l$ *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.

All 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.

Algorithm 2: Implementation of $\mathsf{Assumptions}_{cl1}(S^+, l, k)$

```
  Input : The set of positive $S^+$ samples over $\Sigma$, $\lambda \in S^+$. The maximal width of
          instructions $l \ge 1$. The length of contexts of instructions $k \ge 0$.
  Output: The set of instructions $\Phi$.
1 $\Phi \leftarrow \emptyset$;
2 $SW^+ \leftarrow \emptyset$;
3 foreach $w_+ \in S^+$ do
4     Add all subwords $\alpha$ of $¢w_+\$$, such that $|\alpha| \le l$, into $SW^+$;
5 foreach $w_+ \in S^+$ and subword $\alpha$ of $¢w_+\$$, such that $|\alpha| \le l$ do
6     if $\alpha = xzy$, where $x \in LC_k$, $y \in RC_k$, $|z| > 0$, $xy \in SW^+$ then
7         $\Phi \leftarrow \Phi \cup \{(x, z, y)\}$;
8 return $\Phi$;
```

Algorithm 2 shows one possible implementation of the function $\mathsf{Assumptions}_{cl1}$. It can be easily verified that both Cycles 3 and 5 run in a linear time, provided that l and k are fixed constants. Apparently, the set SW^+ itself can be implemented in a linear time.

The function $\mathsf{Assumptions}_{cl2}$ can be implemented in a very similar way. We only need to somehow memorize the origin of all particular subwords. This can be done by using a map (instead of a set) that maps every subword α to a list of words L_α such that α is a subword of a delimited positive sample $¢w_+\$$ if and only if $w_+ \in L_\alpha$.

Algorithm 3 shows a possible implementation of the function $\mathsf{Assumptions}_{cl3}$. Apparently, Cycle 3 runs in a linear time, provided that l and k are fixed constants.

In the following we prove the correctness of all functions $\mathsf{Assumptions}$ from Example 4.2 with respect to Definition 4.1. It is easy to see, that $\mathsf{Assumptions}_{cl3}(S^+, k, l) \subseteq \mathsf{Assumptions}_{cl2}(S^+, k, l) \subseteq \mathsf{Assumptions}_{cl1}(S^+, k, l)$. Therefore, we only need to prove the

Algorithm 3: Implementation of $\mathsf{Assumptions}_{cl3}(S^+, l, k)$

```
  Input  : The set of positive $S^+$ samples over $\Sigma$, $\lambda \in S^+$. The maximal width of
           instructions $l \geq 1$. The length of contexts of instructions $k \geq 0$.
  Output: The set of instructions $\Phi$.
1 $\Phi \leftarrow \emptyset$;
2 $DS^+ \leftarrow ¢ \cdot S^+ \cdot \$ = \{¢w_+\$ \mid w_+ \in S^+\}$;
3 foreach $w_+ \in S^+$ and $\omega$, such that $¢w_+\$ = \alpha\omega\beta$, $|\omega| \leq l$ do
4     if $\omega = xzy$, where $x \in LC_k$, $y \in RC_k$, $|z| > 0$, and $\alpha xy\beta \in DS^+$ then
5         $\Phi \leftarrow \Phi \cup \{(x, z, y)\}$;
6 return $\Phi$;
```

correctness for the most restrictive function $\mathsf{Assumptions}_{cl3}$. The correctness of the other two functions will follow 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 word w_ϕ. (It does not mean that ϕ is the only applicable instruction. There may also be some other instructions applicable to w_ϕ, but they will definitely not lead to any accepting computation). Let us fix for every $\phi \in J$ some accepting computation $w_\phi \vdash^{(\phi)} w'_\phi \vdash_N^* \lambda$. 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. $J \subseteq \mathsf{Assumptions}_{cl3}(S_0^+, l, k)$. The correctness follows easily from the monotonicity of the function $\mathsf{Assumptions}_{cl3}$ with respect to S^+ and the set inclusion relation. In general, $\mathsf{size}(S_0^+)$ 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, $\mathsf{size}(N) = \sum_{i \in J} |i|$. (See Example 4.3).

The above examples could also 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.

Example 4.3. *In this example we will construct a sequence of 2-clearing restarting automata: $M_0 = (\Sigma_0, I_0), M_1 = (\Sigma_1, I_1), M_2 = (\Sigma_2, I_2), \ldots$, such that for all $i \in \{0, 1, 2, \ldots\}$: $\Sigma_i = \{a_0, a_1, \ldots, a_i\}$, and $I_i \subseteq I_{i+1}$. We will prove that for each $i \in \{0, 1, 2, \ldots\}$ the size of the automaton M_i is polynomial with respect to i, and that there exists an instruction $\phi_i \in I_i$ such that the smallest word $w_i \in L(M_i)$, for which the instruction ϕ_i is applicable, has an exponential length with respect to i, and thus also with respect to the size of the automaton M_i. In constructing these automata we will use a technique of sending signals from one sentinel to the other (and vice versa), which was widely applied in* [5].

The automaton $M_0 = (\Sigma_0, I_0)$ *accepts only one word:* $a_0a_0a_0a_0$*, where* $\Sigma_0 = \{a_0\}$ *and* $I_0 = \{(\text{¢}, a_0a_0a_0a_0, \$)\}$*.*

The best way, how to describe other automata in the sequence is to show how they work in the reverse direction. The automaton M_1 *just sends a signal* a_1 *from the left sentinel* ¢ *to the right sentinel* \$ *starting from the word* $a_0a_0a_0a_0$*, as follows:*

$$\text{¢}\lambda\$ \dashv \text{¢}\underline{a_0a_0a_0a_0}\$ \dashv \text{¢}\underline{\mathbf{a_1}}a_0a_0a_0a_0\$ \dashv \text{¢}\mathbf{a_1}a_0\underline{\mathbf{a_1}}a_0a_0a_0\$ \dashv \text{¢}\mathbf{a_1}a_0\mathbf{a_1}a_0\underline{\mathbf{a_1}}a_0a_0\$ \dashv$$
$$\text{¢}\mathbf{a_1}a_0\mathbf{a_1}a_0\mathbf{a_1}a_0\underline{\mathbf{a_1}}a_0\$ \dashv \text{¢}\mathbf{a_1}a_0\mathbf{a_1}a_0\mathbf{a_1}a_0\mathbf{a_1}a_0\underline{\mathbf{a_1}}\$.$$

This can be easily achieved by the following set of instructions: $I_1 = \{(\text{¢}, a_0a_0a_0a_0, \$),$ $(\text{¢}, \underline{a_1}, a_0a_0)$*,* $(a_1a_0, \underline{a_1}, a_0a_0)$*,* $(a_1a_0, \underline{a_1}, a_0\$)$*,* $(a_1a_0, \underline{a_1}, \$)\}$*. It can be easily verified that the only words accepted by the automaton* M_1 *are the words shown in the above accepting computation. It is because if you proceed in the reverse direction, starting from the empty word, then you basically cannot get anything else than what we have in the above computation. This property will also hold for all subsequent automata in the sequence.*

The automaton $M_2 = (\Sigma_2, I_2)$ *is similar to* M_1*, except that this time it will send a signal* a_2 *from the right sentinel* \$ *to the left sentinel* ¢*. The reason why we want to send a signal in a reverse direction is that we want to preserve the nice property of having only one possible accepting computation. The automaton* M_2 *works as follows (it starts exactly where the previous automaton* M_1 *has ended):*

$$\text{¢}a_1a_0a_1a_0a_1a_0a_1a_0a_1\$ \dashv \text{¢}a_1a_0a_1a_0a_1a_0a_1a_0a_1\underline{\mathbf{a_2}}\$ \dashv$$
$$\text{¢}a_1a_0a_1a_0a_1a_0a_1a_0\underline{\mathbf{a_2}}a_1\mathbf{a_2}\$ \dashv \text{¢}a_1a_0a_1a_0a_1a_0a_1\underline{\mathbf{a_2}}a_0\mathbf{a_2}a_1\mathbf{a_2}\$ \dashv$$
$$\text{¢}a_1a_0a_1a_0a_1a_0\underline{\mathbf{a_2}}a_1\mathbf{a_2}a_0\mathbf{a_2}a_1\mathbf{a_2}\$ \dashv \text{¢}a_1a_0a_1a_0a_1\underline{\mathbf{a_2}}a_0\mathbf{a_2}a_1\mathbf{a_2}a_0\mathbf{a_2}a_1\mathbf{a_2}\$ \dashv$$
$$\dots$$
$$\text{¢}\mathbf{a_2}a_1\mathbf{a_2}a_0\mathbf{a_2}a_1\mathbf{a_2}a_0\mathbf{a_2}a_1\mathbf{a_2}a_0\mathbf{a_2}a_1\mathbf{a_2}a_0\mathbf{a_2}a_1\mathbf{a_2}\$$$

To enable this kind of computation we only need to add the following instructions: $(\circ\circ, \underline{a_2}, \$)$*,* $(\circ\circ, \underline{a_2}, \circ a_2)$*,* $(\text{¢}\circ, \underline{a_2}, \circ a_2)$*, and* $(\text{¢}, \underline{a_2}, \circ a_2)$*, where* $\circ$ *is a placeholder for any of the symbols from* $\{a_0, a_1\}$ *(of course, different occurrences of the placeholder* $\circ$ *can substituted by different symbols). As in the previous case, we have only one possible computation.*

Now we can generalize the above construction also to other automata in the sequence. The automaton M_i*, for* $i > 0$*, is obtained from the automaton* M_{i-1} *as follows:*

(1) *If* i *is odd then the automaton* M_i *will send a signal from the left sentinel* ¢ *to the right sentinel* \$*, thus, in order to obtain* I_i*, we only need to add the following instructions to* I_{i-1}*:* $(\text{¢}, \underline{a_i}, \circ\circ)$*,* $(a_i\circ, \underline{a_i}, \circ\circ)$*,* $(a_i\circ, \underline{a_i}, \circ\$)$*, and* $(a_i\circ, \underline{a_i}, \$)$*, where* $\circ$ *is a placeholder for any of the symbols from* $\{a_0, a_1, \dots, a_{i-1}\}$*.*

(2) *If* i *is even then the automaton* M_i *will send a signal from the right sentinel* \$ *to the left sentinel* ¢*, thus, in order to obtain* I_i*, we only need to add the following instructions to* I_{i-1}*:* $(\circ\circ, \underline{a_i}, \$)$*,* $(\circ\circ, \underline{a_i}, \circ a_i)$*,* $(\text{¢}\circ, \underline{a_i}, \circ a_i)$*, and* $(\text{¢}, \underline{a_i}, \circ a_i)$*, where* $\circ$ *is a placeholder for any of the symbols from* $\{a_0, a_1, \dots, a_{i-1}\}$*.*

First observe that the size of the M_i is polynomial with respect to i. This can be easily proved inductively by using a simple observation that we only add $O(i^2)$ new instructions to I_{i-1} when constructing I_i.

Now consider any $i > 0$. If i is even then let us take the instruction $\phi_i = (¢, \underline{a_i}, a_{i-1}a_{i-2})$. This instruction can be applied only after the previous signal a_{i-1} has arrived to the left sentinel ¢. In other words, it can be applied only to the longest word in $L(M_{i-1})$. However, the length of the longest word in $L(M_j)$ is exponential with respect to j, for all $j \geq 0$, since every time the signal traverses from one end to the other, the length of the resulting word more than doubles.

In the case when i is odd we can take the instruction $\phi_i = (a_{i-2}a_{i-1}, \underline{a_i}, \$)$, which can be applied only after the previous signal a_{i-1} has arrived to the right sentinel \$.

The above example clearly shows that sometimes we need to consider an exponentially large set (i.e. having an exponentially large binary representation) of positive samples S^+ in order to obtain all instructions of the hidden target automaton. In the construction itself we have used 2-clearing restarting automata as a witness of this fact. This naturally rises an open question, whether this can also happen to 1-clearing restarting automata.

Finally, the following Algorithm 4 shows a possible implementation of the function Simplify both for clearing and subword-clearing restarting automata.

Algorithm 4: Implementation of Simplify(Φ)

```
Input  : The set of instructions $\Phi$.
Output: The simplified set of instructions $\Psi$.
1 $\Psi \leftarrow \emptyset$;
2 foreach $\phi = (x, z \to t, y) \in \Phi$ in some fixed order do
3     if $z \not\vdash^*_\Psi t$ in the context $(x, y)$ then
4         $\Psi \leftarrow \Psi \cup \{(x, z \to t, y)\}$;
5 return $\Psi$;
```

In Definition 2.10 we have defined the the rewriting relation $\vdash$ only with respect to the context $(¢, \$)$, but it can be easily extended to a general context (x, y), where $x \in LC_k$, $y \in RC_k$. The statement "$z \vdash^*_\Psi t$ in the context (x, y)" then basically says that the instruction ϕ can be simulated by using some other instructions from Ψ, and therefore ϕ is redundant. We enumerate the instructions from Φ in some fixed order that guarantees that the later instructions in this order cannot simulate any former instruction. If we fix the maximal width of instructions l and the length of contexts k then the condition in Step 3 can be verified in the $O(1)$ time, and the whole algorithm runs in $O(|\Phi|)$ time.

In the following Theorem 4.1 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 4.1. *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 Definition4.1, 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 5:

Algorithm 5: Extension of Samples (S_0^+, S_0^-)

```
Input  : The set of positive and negative samples $(S_0^+, S_0^-)$.
Output: The set of extended positive and negative samples $(S_0^+, S_0^-)$.
$S_0^- \leftarrow \emptyset$;
while $\exists$ bad instruction $\phi \in \Phi$ such that $\phi$ is not disabled by $(S_0^+, S_0^-)$ do
    Let $w_- \notin L(N), w_+ \in L(N) : w_- \vdash^{(\phi)} w_+$;
    $S_0^+ \leftarrow S_0^+ \cup \{w_+\}$;
    $S_0^- \leftarrow S_0^- \cup \{w_-\}$;
```

Every added pair w_+, w_- effectively disables at least one instruction from Φ, so the function 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. □

Note that we do not have to disable all bad instructions having the width bounded from above by l. We could restrict ourselves only to the possibly smaller finite set of instructions $\Phi := \mathsf{Assumptions}(L(M), l, k)$, because basically these are the only instructions we may eventually encounter inside the algorithm $\mathsf{Infer}(S^+, S^-, l, k)$, provided that $S^+ \subseteq L(M)$. However, this modification does not improve our proof in any way, since the set $\mathsf{Assumptions}(L(M), l, k)$ may still be exponentially large with respect to the size of the automaton M. The best we can hope for is that in specific problem instances some smaller sets S^+, S^- will be sufficient. For instance, the set $\mathsf{Assumptions}(L(M), l, k)$ might be large, but it may also contain only a small number of bad instructions.

Example 4.4. *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 2.1). 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 4.2 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:* $¢\underline{a}a \vdash ¢a$*,* $a\underline{a}b \vdash ab$*,* $a\underline{b}b \vdash ab$*,* $b\underline{b}\$ \vdash b\$$*,* $¢\underline{ab}\$ \vdash ¢\$$*,* $a\underline{ab}b \vdash ab$*,* $¢\underline{aabb}\$ \vdash ¢\$$*. 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* $N = (\{a, b\}, \{(¢, \underline{ab}, \$),\ (a, \underline{ab}, b)\})$ *(after the simplification), which is equivalent to the hidden target automaton* M*. The function* $\mathsf{Simplify}$ *removes 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 4.4 is not big enough to guarantee the properties expressed in Theorem 4.1. What we mean here is that 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 prohibited by the pair (S^+, S^-). Therefore, $\mathsf{Infer}(S^+, S^-, l, k)$ would output an incorrect solution in this case.

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 $S_0^- = \{aab,\ abb,\ aaab,\ abbb,\ aaaab,\ aaabb,\ aabbb,\ abbbb,\ aaaaab,\ aaaabb,\ aabbbb,\ abbbbb,\ aaaaabb,\ aabbbbb,\ aaaaaabb,\ aabbbbbb\}$ satisfy the properties of Theorem 4.1, for $k = 1$ and $l = 6$. This is because $\mathsf{Assumptions}_{cl1}(S_0^+, l, k) = \mathsf{Assumptions}_{cl1}(\{a^n b^n \mid n \geq 0\}, l, k) = \Phi_0$, and the negative samples in S_0^- can filter out all bad instructions in Φ_0. In this case, the function $\mathsf{Infer}(S^+, S^-, l, k)$ will give us the correct solution for every finite set of positive samples $S^+ \supseteq S_0^+$ and every finite set of negative samples $S^- \supseteq S_0^-$ consistent with the target language $\{a^n b^n \mid n \geq 0\}$.

Also note that if we used the function $\mathsf{Assumptions}_{cl3}$ in Example 4.4 instead of the function $\mathsf{Assumptions}_{cl1}$ (see Example 4.2 for definition) we would need no negative samples at all.

In the following Example 4.5 we show how the inference algorithm can be used also for a more general subword-clearing restarting automata. We will use the following function Assumptions: $\mathsf{Assumptions}_{scl1}(S^+, l, k) := \{(x, z \to t, y) \mid x \in LC_k, y \in RC_k, |z| > 0, |xzty| \leq l,\ t \text{ is a proper subword of } z \text{ and } \exists w_1, w_2 \in S^+ : xzy \text{ is a subword of } ¢w_1\$ \text{ and } xty \text{ is a subword of } ¢w_2\$\}$. It can be shown (similarly as in Example 4.2) that this assumptions function is correct with respect to the model of subword-clearing restarting automata.

Example 4.5. *Consider the* 1-scl-RA $M = (\{a,b,c\}, \{(a, acb \to c, b), (¢, acb, \$)\})$ *recognizing the language* $L(M) = \{a^n cb^n \mid n > 0\} \cup \{\lambda\}$. *(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. 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 4.4, 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 4.1 (modified for the model of subword-clearing restarting automata), for $k = 1$ and $l = 6$.

Note that if we used a stronger function $\mathsf{Assumptions}$: $\mathsf{Assumptions}_{scl3}(S^+, l, k) := \{(x, z \to t, y) \mid x \in LC_k, y \in RC_k, |z| > 0, |xzty| \le l,$ t is a proper subword of z, and $\exists w_1, w_2 \in S^+ :$ $w_1 = \alpha z \beta, w_2 = \alpha t \beta$, x is a suffix of $¢\alpha$ and y is a prefix of $\beta\$\}$, then we would need no negative samples at all.

In the following Example 4.6 we illustrate how to execute an active learning approach for the inference of a more realistic language of correct arithmetical expressions.

Example 4.6. *Our goal is to infer a model of a subword-clearing restarting automaton recognizing the language of correct arithmetical expressions. For convenience, we use only a simplified 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. Instead, we focus more on the big picture and mention only those artifacts that have some impact on the decisions made during the learning process. 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.*

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 4.5 for definition) gives us altogether* 144 *instruction candidates, where only two of them:* $(¢, [, a)$ *and* $(a,], \$)$ *are going to 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* 37 *instructions after simplification. Let us generate some expressions recognized by this automaton* M_1. *The following Table 2 lists the set of all recognized expressions up to length* 5.

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

Table 1. The initial set of positive and negative samples.

Positive Samples S_1^+			*Negative Samples* S_1^-			
a	$[[a+a]]$	$a+[a-a]$	$+$	$a[$	$--$	$[]$
$a+a$	$[[a-a]]$	$a-[a+a]$	$-$	$a]$	$-[$	$]a$
$a-a$	$a+a+a$	$a-[a-a]$	$[$	$++$	$-]$	$]+$
$[a]$	$a+a-a$	$[a+a]+a$	$]$	$+-$	$[a$	$]-$
$[a+a]$	$a-a+a$	$[a+a]-a$	aa	$+[$	$[+$	$][$
$[a-a]$	$a-a-a$	$[a-a]+a$	$a+$	$+]$	$[-$	$]]$
$[[a]]$	$a+[a+a]$	$[a-a]-a$	$a-$	$-+$	$[[$	

Table 2. The set of all expressions up to length 5 recognized by M_1.

Expressions Recognized by M_1					
λ	$a+[a$	$[a]]$	$\mathbf{a}-\mathbf{a}+\mathbf{a}$	$a]-[a$	$[\mathbf{a}]+\mathbf{a}$
$\mathbf{a}$	$a-a]$	$[[a]$	$\mathbf{a}-\mathbf{a}-\mathbf{a}$	$a]]+a$	$[\mathbf{a}]-\mathbf{a}$
$\mathbf{a}+\mathbf{a}$	$a-[a$	$[[[a$	$a-a]]$	$a]]-a$	$[a]]]$
$\mathbf{a}-\mathbf{a}$	$a]+a$	$\mathbf{a}+\mathbf{a}+\mathbf{a}$	$\mathbf{a}-[\mathbf{a}]$	$a]]]]$	$[[a+a$
$a]]$	$a]-a$	$\mathbf{a}+\mathbf{a}-\mathbf{a}$	$a-[[a$	$[\mathbf{a}+\mathbf{a}]$	$[[a-a$
$[\mathbf{a}]$	$a]]]$	$a+a]]$	$a]+a]$	$[a+[a$	$[[\mathbf{a}]]$
$[[a$	$[a+a$	$\mathbf{a}+[\mathbf{a}]$	$a]+[a$	$[\mathbf{a}-\mathbf{a}]$	$[[[a]$
$a+a]$	$[a-a$	$a+[[a$	$a]-a]$	$[a-[a$	$[[[[a$

correct arithmetical expressions that it had not seen before, e.g. $a+[a]$. *Our next step will be to add the above 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 shown in Table 2. The inference algorithm will now return on the input* (S_2^+, S_2^-, l, k) *a consistent automaton* M_2 *having only* 36 *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. expressions shown in Table 3.*

Table 3. The set of some incorrect arithmetical expressions recognized by M_2.

Expressions Recognized by M_2		
$a-a]-a$	$a-[a-a$	$[[a-a]$
$a-a]+a$	$a-[a+a$	$[[a+a]$
$a+a]-a$	$a+[a-a$	$[a-a]]$
$a+a]+a$	$a+[a+a$	$[a+a]]$

Again, let $S_3^+ := S_2^+$ *and let* S_3^- *be the set of all negative samples in* S_2^- *extended by the incorrect arithmetical expressions shown in Table 3. This time the inference algorithm will return on the input* (S_3^+, S_3^-, l, k) *a consistent automaton* M_3 *having* 24 *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 set of instructions of M_3.

Instructions of M_3			
$(¢, \underline{a}, \$)$	$(¢, \underline{a+}, a)$	$(a, \underline{-a}, -)$	$(a, \underline{+a}, +)$
$(¢, \underline{a-}, a)$	$(¢, \underline{a+}, [)$	$(a, \underline{-a}, +)$	$(a, \underline{+a},])$
$(¢, \underline{a-}, [)$	$([, \underline{a+}, a)$	$(a, \underline{-a},])$	$(], \underline{+a}, \$)$
$([, \underline{a-}, a)$	$(-, \underline{a+}, a)$	$(], \underline{-a}, \$)$	$(¢, \underline{[a]}, \$)$
$(-, \underline{a-}, a)$	$(+, \underline{a+}, a)$	$(a, \underline{+a}, \$)$	$(¢, \underline{[a]} \to a, \$)$
$(+, \underline{a-}, a)$	$(a, \underline{-a}, \$)$	$(a, \underline{+a}, -)$	$([, \underline{[a]} \to a,])$

Unfortunately, the automaton is not complete yet. It does not recognize, for instance, the following correct arithmetical expression: $a + [a + [a]]$. This time we 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. The inference algorithm would fail on the input (S_2^+, S_2^-, l, k) and no additional samples would ever change this situation. The intuitive reason behind this observation can be best explained as follows: if we want to recognize the language $\{[^n a]^n \mid n \geq 0\}$ we cannot do it without some rewriting of the following form: $\underline{[^k a]^k} \to [^l a]^l$, where $0 \leq l < k$. This kind of rewriting requires the use of a subword-clearing instruction having the width at least 6 (e.g. $([, \underline{[a]} \to a,])$).

Another important observation is the following: Note that the symbols $+$ and $-$ are used interchangeably. We could have used only one single symbol (e.g. $\square$) representing both symbols $+$ and $-$. Better approach would be to deduce this fact from the set of positive samples. We could, for instance, deduce that both symbols $+$ and $-$ have the same set of contexts in the set of positive samples. Some kind of categorization would be crucial in real world applications where we encounter much bigger alphabets.

5. 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 5.1. *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 will 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^+. And at last, for each $a \in \Omega$ add all words $w \in \{a, \overline{a}\}^{\le l}$, such that $|w|_a \ge 1$ and $|w|_{\overline{a}} \ge 1$, 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 $w \in \{a, \overline{a}\}^{\le l}$, such that $|w|_a \ge 1$ and $|w|_{\overline{a}} \ge 1$, 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 [5] 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 5.1 can be generalized to cover all clearing restarting automata.

Theorem 5.2. *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. 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 $i \in \{1, 2, \ldots, n\}$. Let $\Omega = \{a_1, a_2, \ldots, a_m\}$ be the set of all variables occurring in ψ. As in Theorem 5.1, we will specify a finite set of positive and negative samples S^+, S^-, $S^+ \cap S^- = \emptyset$, $\lambda \in S^+$, such that the formula ψ is satisfiable if and only if there exists a k-cl-RA $M = (\Sigma, I)$ consistent with S^+ and S^-, where the width of instructions of M is bounded from above by l. The difference here is that the alphabet Σ will contain also some other special symbols, except $\Omega \cup \overline{\Omega}$. (We use the same notation as in the proof of Theorem 5.1). 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 $w_{C_i}^- = \Box^k \overline{\ell_{i,1}} \Box^k \overline{\ell_{i,2}} \Box^k \overline{\ell_{i,3}} \Box^k$ to the set of negative samples S^-, where $\Box$ is a special dummy symbol that will later match the contexts of the instructions. For each variable $a \in \Omega$, add the following positive samples: $w_0^+ = \Box^k a \Box^t \overline{a} \Box^k$, $w_1^+ = \Box^k a \Box^{k+t}$, $w_2^+ = \Box^{k+t} \overline{a} \Box^k$, and $w_3^+ = \Box^{4k}$ to the set of positive samples S^+, and also add the negative sample $w_4^- = \Box^k a \Box^{2k} \overline{a} \Box^k$ to the set of negative samples S^-, where $t = l - 2k - 3$. Since $l \geq 4k + 4$, it follows that $t \geq 2k + 1$, and thus $w_0^+ \neq w_4^-$. Our next goal is to allow only the following types of instructions, where $a \in \Omega$:

(a) $(\Box^k, a, \Box^k)$.
(b) $(\Box^k, \overline{a}, \Box^k)$.
(c) $(¢, \Box^k a \Box^{k+t}, \$)$.

(d) $(¢, \square^{k+t}\overline{a}\square^k, \$)$.
(e) $(¢, \square^{4k}, \$)$.

All of these instructions are allowed since they have the width at most l. (The longest are the instructions of the type (c) and (d), having the width $2k + t + 3 = l$). On the other hand, it is not possible to store the whole word $w_0^+ = \square^k a \square^t \overline{a} \square^k$ in one instruction containing both sentinels (such us, for instance, $(¢, \square^k a \square^t \overline{a} \square^k, \$)$), since its width would be $2k + t + 4 = l + 1 > l$. Also note, that the instructions (c) and (d) will never interfere with any of the negative samples $w_{C_i}^- = \square^k \overline{\ell_{i,1}} \square^k \overline{\ell_{i,2}} \square^k \overline{\ell_{i,3}} \square^k$, because: $|\square^k \overline{\ell_{i,1}} \square^k \square^k \square^k| = |\square^k \square^k \square^k \overline{\ell_{i,3}} \square^k| = 4k + 1$, while $|\square^k a \square^{k+t}| = |\square^{k+t} \overline{a} \square^k| = 2k + t + 1 = l - 2 \geq 4k + 2$. The same holds for the negative sample $w_4^- = \square^k a \square^{2k} \overline{a} \square^k$, because $|\square^k a \square^{2k} \square^k| = |\square^k \square^{2k} \overline{a} \square^k| = 4k + 1$.

In the following we introduce a general technique, how to prohibit the inference of any specific undesirable instruction. Note that it is no longer plausible just to introduce some negative samples as we did in the proof of Theorem 5.1, since now the instructions also have contexts. Suppose that we want to prohibit the instruction $\phi = (x, z, y)$, where $|xzy| \leq l$. Let x' (y', respectively) be the largest possible subword of x (y, respectively) not containing the sentinels (¢, \$); thus either $x = x'$, or $x = ¢x'$ (either $y = y'$, or $y = y'\$$, respectively). There are only the following four possible cases:

(1) $x = ¢x'$ and $y = y'\$$
(2) $x = ¢x'$ and $y = y'$
(3) $x = x'$ and $y = y'\$$
(4) $x = x'$ and $y = y'$

In the first case we only need to add the word $x'zy'$ to the set of negative samples S^- and the word $x'y'$ to the set of positive samples S^+ in order to prohibit the instruction $\phi = (x, z, y) = (¢x', z, y'\$)$. In the other cases let us first introduce a new symbol $\square_\phi$, which we also add to our alphabet Σ. This is what we do in the particular cases:

(1) $S^- := S^- \cup \{x'zy'\}$, $S^+ := S^+ \cup \{x'y'\}$.
(2) $S^- := S^- \cup \{x'zy'\square_\phi\}$, $S^+ := S^+ \cup \{x'y'\square_\phi\}$.
(3) $S^- := S^- \cup \{\square_\phi x'zy'\}$, $S^+ := S^+ \cup \{\square_\phi x'y'\}$.
(4) $S^- := S^- \cup \{\square_\phi x'zy'\square_\phi\}$, $S^+ := S^+ \cup \{\square_\phi x'y'\square_\phi\}$.

For every prohibited instruction ϕ we add two new samples: a positive sample w_ϕ^+ to the set S^+ and a negative sample w_ϕ^- to the set S^-. It is easy to see, that in each case we have effectively prohibited the instruction $\phi = (x, z, y)$. Note that this is the only instruction not containing the symbol $\square_\phi$ (and having $x \in LC_k, y \in RC_k$, see Definition 2.10) that reduces w_ϕ^+ to w_ϕ^-. However, in the following we will also have to verify the consistency of the constructed k-cl-RA M with these new samples, i.e. that for every prohibited instruction ϕ the following holds: $w_\phi^+ \vdash_M^* \lambda$ and $w_\phi^- \not\vdash_M^* \lambda$. In all cases, the newly added positive sample w_ϕ^+ can always be reduced to the empty word in one step by using the instruction $(¢, w_\phi^+, \$)$. This is because the width of this instruction is $2 + |w_+| \leq 2 + 2 + |x'y'| \leq 4 + 2k \leq l$. If we use a new symbol $\square_\phi$ in w_ϕ^+, then the instruction $(¢, w_\phi^+, \$)$ will be applicable only to this specific sample $w_\phi^+ \in S^+$, and thus will not interfere with other samples. On the other hand, the verification of the second condition ($w_\phi^- \not\vdash_M^* \lambda$) is more difficult. Fortunately, we will always use the symbol $\Box_\phi$ in w_ϕ^-, i.e. the first case will never occur in our proof.

Now we have all necessary ingredients to finish the proof. For each $a \in \Omega$ consider the following positive sample: $w_0^+ = \Box^k a \Box^t \overline{a} \Box^k$. By using the above technique we disable all instructions applicable to this word having the width at most l, except for the instructions of the form (a) – (e). Observe, that we will never attempt to disable any instruction of the form $\phi = (¢x', z, y'\$)$. This is because the word $w_0^+ = \Box^k a \Box^t \overline{a} \Box^k$ (as we already mentioned above) is too long. Moreover, there is only polynomially many disabled instructions, since there is only polynomially many subwords of the above word. Now we have completely specified the sets of positive and negative samples S^+, S^-, $S^+ \cap S^- = \emptyset$, $\lambda \in S^+$, and thus we can proceed with the proof in exactly the same way as in Theorem 5.1.

($\Rightarrow$) 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 show that there exists 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. Consider the following k-cl-RA $M = (\Sigma, I)$: First, we add to I the following set of instructions: $I_1 = \{(\Box^k, a, \Box^k) \mid a \in \Omega : v(a) = 1\} \cup \{(\Box^k, \overline{a}, \Box^k) \mid a \in \Omega : v(a) = 0\}$. It can be easily observed that, for all literals $l \in \Omega \cup \overline{\Omega} : \Box^k l \Box^k \vdash_M \Box^k \Box^k \Leftrightarrow v(l) = 1$, or equivalently: $\Box^k \overline{l} \Box^k \vdash_M \Box^k \Box^k \Leftrightarrow v(l) = 0$. Therefore no negative sample $w_{C_i}^- = \Box^k \overline{\ell_{i,1}} \Box^k \overline{\ell_{i,2}} \Box^k \overline{\ell_{i,3}} \Box^k$, where $i \in \{1, 2, \ldots, n\}$, can be reduced to the positive sample $\Box^{4k}$ by using the instructions from I_1, because otherwise it would mean that $v(\ell_{i,1}) = v(\ell_{i,2}) = v(\ell_{i,3}) = 0$. Next we add to I the following set of instructions $I_2 = \{(¢, \Box^k a \Box^{k+t}, \$), (¢, \Box^{k+t} \overline{a} \Box^k, \$) \mid a \in \Omega\} \cup \{(¢, \Box^{4k}, \$)\}$. As we have already stated above, no instruction from I_2 will ever interfere with any negative sample $w_{C_i}^- = \Box^k \overline{\ell_{i,1}} \Box^k \overline{\ell_{i,2}} \Box^k \overline{\ell_{i,3}} \Box^k$, or $w_4^- = \Box^k a \Box^{2k} \overline{a} \Box^k$. Finally, we add to I all instructions $(¢, w_\phi^+, \$)$, where w_ϕ^+ was added to the set of positive samples S^+ during the process of disabling some undesirable instruction ϕ. These instructions are applicable only to words containing these special symbols $\Box_\phi$, so we do not have to care about them at all. Now we will show that the constructed k-cl-RA $M = (\Sigma, I)$ is consistent with the set of positive and negative samples S^+ and S^-. (The width of instructions of M is apparently bounded from above by l). First, it is easy to see, that for each variable $a \in \Omega$ the following positive samples: $w_0^+ = \Box^k a \Box^t \overline{a} \Box^k$, $w_1^+ = \Box^k a \Box^{k+t}$, $w_2^+ = \Box^{k+t} \overline{a} \Box^k$, $w_3^+ = \Box^{4k}$ are all reducible to the empty word λ. The positive sample w_0^+ is always reducible to either w_1^+, or w_2^+, depending on whether $(\Box^k, a, \Box^k) \in I_1$, or $(\Box^k, \overline{a}, \Box^k) \in I_1$. The other positive samples: w_1^+, w_2^+, w_3^+ can be reduced to the empty word λ in a single step by using the corresponding instruction from I_2. The negative sample $w_4^- = \Box^k a \Box^{2k} \overline{a} \Box^k$ clearly cannot be reduced to the empty word λ. We can clear either the letter a, or $\overline{a}$ by using the corresponding instruction from I_1, but then we get the irreducible word $\Box^k \Box^{2k} \overline{a} \Box^k$, or $\Box^k a \Box^{2k} \Box^k$. None of the instructions from I_2 can be applied to a such word, since the length $|\Box^k a \Box^{2k} \Box^k| = |\Box^k \Box^{2k} \overline{a} \Box^k| = 4k + 1$, while $|\Box^k a \Box^{k+t}| = |\Box^{k+t} \overline{a} \Box^k| = 2k + t + 1 = l - 2 \geq 4k + 2$. It remains to be shown, that no negative sample w_ϕ^-, which was added to the set of negative samples S^- during the process of disabling some instruction ϕ, can be reduced to the empty word λ. Both the negative sample w_ϕ^- and the corresponding positive sample w_ϕ^+ contain the special symbol $\Box_\phi$. The negative sample w_ϕ^- without this special symbol $\Box_\phi$ is basically a subword of some word $\Box^k a \Box^t \overline{a} \Box^k$. First observe, that the only instruction from I, that could be possibly applied to the negative sample w_ϕ^-, is some instruction from I_1. Without loss of generality assume that we can apply the instruction $\rho = (\Box^k, a, \Box^k) \in I_1$ to the word w_ϕ^-, i.e. $w_\phi^- \vdash^{(\rho)} w'$. This also implies that $(\Box^k, \overline{a}, \Box^k) \notin I_1$. It is easy to see, that no other instruction from I can be applied to the resulting word w', except possibly the instruction $(¢, w_\phi^+, \$)$. But if $(¢, w_\phi^+, \$)$ was applicable to w', it would have implied that we wanted to disable the instruction ρ itself, which is not possible, since ρ is of the form (a). Thus, we have shown that the constructed k-cl-RA $M = (\Sigma, I)$ is consistent with S^+ and S^-, and that the width of instructions of M is bounded from above by l. As in Theorem 5.1, the size of the constructed set of positive and negative samples is linear with respect to the size of the formula ψ.

($\Leftarrow$) Now suppose that there exists 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. 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 $(\Box^k, a, \Box^k) \in I$, or $(\Box^k, \overline{a}, \Box^k) \in I$. Consider the positive sample $w_0^+ = \Box^k a \Box^t \overline{a} \Box^k$. We know that $\Box^k a \Box^t \overline{a} \Box^k \vdash_M^* \lambda$. Let $\phi \in I$ be the first instruction used in any such accepting computation. The instruction ϕ is either of the form $(\Box^k, a, \Box^k)$, or $(\Box^k, \overline{a}, \Box^k)$, because all other instructions are disabled. Moreover, it cannot happen that both instructions $(\Box^k, a, \Box^k)$, $(\Box^k, \overline{a}, \Box^k)$ are in I, because it would mean that $\Box^k a \Box^{2k} \overline{a} \Box^k \vdash_M^* \Box^{4k}$, where $\Box^k a \Box^{2k} \overline{a} \Box^k \in S^-$ and $\Box^{4k} \in S^+$. Now let us define the assignment $v : \Omega \to \{0,1\}$ as follows: for each $a \in \Omega : v(a) = 1$ if $(\lambda, a, \lambda) \in I$, and $v(a) = 0$ 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 $w_{C_i}^- = \Box^k \overline{\ell_{i,1}} \Box^k \overline{\ell_{i,2}} \Box^k \overline{\ell_{i,3}} \Box^k \in S^-$. Therefore, $(\Box^k, \overline{\ell_{i,1}}, \Box^k) \notin I$ or $(\Box^k, \overline{\ell_{i,2}}, \Box^k) \notin I$ or $(\Box^k, \overline{\ell_{i,3}}, \Box^k) \notin I$, 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.

As in Theorem 5.1, it can easily be shown that the task is NP-complete for 1-cl-RA, since the membership problem for 1-cl-RA is decidable in a polynomial time. Whether the task is in NP also for $k > 1$ depends on the complexity of membership problem for k-cl-RA. $\Box$

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 $\lambda \in S^+$.

6. Conclusion

We have shown that it is possible, by using a simple learning algorithm, 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. This fact is contrasted with the negative result that the task of finding a consistent clearing restarting automaton with the given set of positive and negative samples is NP-hard under the constraint that we impose an upper bound on the width of instructions. This result resembles the famous result of Gold ([11]) who showed that the question of whether there is a finite automaton with at most n states which agrees with the given set of positive and negative samples 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 $O(n)$ (see [5]). 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, Δ-clearing or Δ^*-clearing restarting automata. It would be also interesting to further investigate the complexity of membership and equivalence problem for these models and to study grammatical inference for other related models.

Acknowledgements

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

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

[2] R. V. Book and F. Otto. *String-rewriting systems*. Springer-Verlag, New York, NY, USA, 1993.

[3] G. Buntrock and K. Lorys. On growing context-sensitive languages. In *Proc. 19th ICALP, Lecture Notes in Computer Science (W. Kuich,ed*, pages 77–88. Springer-Verlag, 1992.

[4] P. Černo and F. Mráz. Clearing restarting automata. In H. Bordinh, R. Freund, M. Holzer, M. Kutrib, and F. 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.

[5] P. Černo and F. Mráz. Clearing restarting automata. *Fundamenta Informaticae*, 104(1):17–54, 2010.

[6] P. Černo and F. Mráz. Delta-clearing restarting automata and cfl. In G. Mauri and A. Leporati, editors, *Developments in Language Theory*, volume 6795 of *Lecture Notes in Computer Science*, pages 153–164. Springer Berlin / Heidelberg, 2011.

[7] P. Černo and F. Mráz. Delta-clearing restarting automata and cfl. Technical Report 8/2011, Charles University, Faculty of Mathematics and Physics, Prague, 2011.

[8] E. Dahlhaus and M. Warmuth. Membership for growing context sensitive grammars is polynomial. In P. Franchi-Zannettacci, editor, *CAAP '86*, volume 214 of *Lecture Notes in Computer Science*, pages 85–99. Springer Berlin / Heidelberg, 1986. 10.1007/BFb0022661.

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

[10] R. Eyraud, C. de la Higuera, and J.-C. Janodet. Lars: A learning algorithm for rewriting systems. *Machine Learning*, 66:7–31, 2007. 10.1007/s10994-006-9593-8.

[11] E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37, 1978.

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

[13] P. Jančar, F. Mráz, M. Plátek, and J. Vogel. Restarting automata. In H. Reichel, editor, *FCT'95*, volume 965 of *LNCS*, pages 283–292, Dresden, Germany, August 1995. Springer.

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

[15] R. McNaughton. Algebraic decision procedures for local testability. *Theory of Computing Systems*, 8:60–76, 1974. 10.1007/BF01761708.

[16] F. Otto. Restarting automata. In Z. Ésik, C. Martín-Vide, and V. Mitrana, editors, *Recent Advances in Formal Languages and Applications*, volume 25 of *Studies in Computational Intelligence*, pages 269–303. Springer, Berlin, 2006.

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

Department of Computer Science, Charles University, Faculty of Mathematics and Physics, Malostranské nám. 25, 118 00 PRAHA 1, Czech Republic