Representing Languages by Learnable Rewriting Systems*

Rémi Eyraud, Colin de la Higuera, Jean-Christophe Janodet

EURISE, Université Jean Monnet de Saint-Etienne,
23 rue Paul Michelon, 42023 Saint-Etienne, France.
{remi.eyraud,cdlh,janodet}@univ-st-etienne.fr

Abstract. Powerful methods and algorithms are known to learn regular languages. Aiming at extending them to more complex grammars, we choose to change the way we represent these languages. Among the formalisms that allow to define classes of languages, the one of string-rewriting systems (SRS) has outstanding properties. Indeed, SRS are expressive enough to define, in a uniform way, a noteworthy and non trivial class of languages that contains all the regular languages, $\{a^n b^n : n \geq 0\}$, $\{w \in \{a,b\}^* : |w|_a = |w|_b\}$, the parenthesis languages of Dyck, the language of Lukasewitz, and many others. Moreover, SRS constitute an efficient (often linear) parsing device for strings, and are thus promising and challenging candidates in forthcoming applications of Grammatical Inference. In this paper, we pioneer the problem of their learnability. We propose a novel and sound algorithm which allows to identify them in polynomial time. We illustrate the execution of our algorithm throughout a large amount of examples and finally raise some open questions and research directions.

Keywords. Learning Context-Free Languages, Rewriting Systems.

1 Introduction

Whereas for the case of learning regular languages there are now a number of positive results and algorithm, things tend to get harder when the entire class of context-free languages is considered [10, 17]. Typical approaches have consisted in learning special sorts of grammars [20], by using genetic algorithms or artificial intelligence ideas [16], and by compression techniques [13]. Yet more and more attention has been drawn to the problem: One example is the Omphalos context-free language learning competition [19].

An attractive alternative when blocked by negative results is to change the representation mode. In this line, little work has been done for the context-free case: One exception is pure context-free grammars which are grammars where both the non-terminals and the terminals come from a same alphabet [8].

* This work was supported in part by the IST Programme of the European Community, under the Pascal Network of Excellence, IST-2002-506778. This publication only reflects the authors' views.

In this paper, we investigate string-rewriting systems (SRS). Invented in 1914 by Axel Thue, the theory of SRS (also called semi-Thue systems) and its extension to trees and to graphs was paid a lot of attention all along the 20^{th} century (see [1, 3]). Rewriting a string consists in replacing substrings by others, as far as possible, following laws called rewrite rules. For instance, consider strings made of a and b, and the single rewrite rule $ab \rightarrow \lambda$. Using this rule consists in replacing a substring ab by the empty string, thus in erasing ab. It allows to rewrite $abaabbab$ as follows:

$$abaabb\underline{ab} \rightarrow aba\underline{ab}b \rightarrow \underline{ab}ab \rightarrow \underline{ab} \rightarrow \lambda$$

Other rewriting derivations may be considered but they all lead to λ. Actually, it is rather clear on this example that a string will rewrite to λ *iff* it is a "parenthetic" string, *i.e.*, a string of the Dyck language. More precisely, the Dyck language is completely characterized by this single rewrite rule and the string λ, which is reached by rewriting all other strings of the language. This property was first noticed in a seminal paper by Nivat [14] which was the starting point of a large amount of work during the three last decades.

We use this property, and others to introduce a class of rewriting systems which is powerful enough to represent in an economical way all regular languages and some typical context-free languages: $\{a^n b^n : n \geq 0\}$, $\{w \in \{a, b\}^* : |w|_a = |w|_b\}$, the parenthesis languages of Dyck, the language of Lukasewitz, and many others. We also provide a learning algorithm called LARS (Learning Algorithm for Rewriting Systems) which can learn systems representing these languages from string examples and counter-examples of the language.

In section 2 we give the general notations relative to the languages we consider and discuss the notion of learning. We introduce our rewriting systems and their expressiveness in section 3 and develop the properties they must fulfill to be learnable in section 4. The general learning algorithm is presented and justified in section 5. We report in section 6 some experimental results and conclude.

2 Learning Languages

An *alphabet* Σ is a finite nonempty set of symbols called *letters*. A *string* w over Σ is a finite sequence $w = a_1 a_2 \dots a_n$ of letters. Let $|w|$ denote the length of w. In the following, letters will be indicated by $a, b, c, \dots$, strings by $u, v, \dots, z$, and the empty string by λ. Let Σ^* be the set of all strings. We assume a fixed but arbitrary total order $\leq$ on the letters of Σ. As usual, we extend $\leq$ to Σ^* by defining the *hierarchical order* [15], denoted $\trianglelefteq$, as follows:

$$\forall w_1, w_2 \in \Sigma^*, w_1 \trianglelefteq w_2 \textit{ iff } \begin{cases} |w_1| < |w_2| \text{ or} \\ |w_1| = |w_2| \text{ and } \exists u, v_1, v_2 \in \Sigma^*, \exists x_1, x_2 \in \Sigma \\ \text{ such that } w_1 = u x_1 v_1, w_2 = u x_2 v_2 \text{ and } x_1 < x_2. \end{cases}$$

By a language we mean any subset $L \subseteq \Sigma^*$. Many classes of languages were investigated in the literature. In general, the definition of a class $\mathbb{L}$ relies on a class $\mathbb{R}$ of abstract machines, here called *representations*, that characterize all and only the languages of $\mathbb{L}$: (1) $\forall R \in \mathbb{R}, \mathcal{L}(R) \in \mathbb{L}$ and (2) $\forall L \in$

$\mathbb{L}, \exists R \in \mathbb{R}$ such that $\mathcal{L}(R) = L$. Two representations R_1 and R_2 are *equivalent iff* $\mathcal{L}(R_1) = \mathcal{L}(R_2)$. In this paper, we will investigate the class REG of *regular* languages characterized by the class DFA of *deterministic finite automata* (*dfa*), and the class CFL of *context-free languages* represented by the class CFG of *context-free grammars* (*cfg*).

We now turn to our learning problem. The *size* of a representation R, denoted by $\|R\|$, is polynomially related to the size of its encoding.

Definition 1. *Let $\mathbb{L}$ be a class of languages represented by some class $\mathbb{R}$.*

1. *A sample S for a language $L \in \mathbb{L}$ is a finite set of ordered pairs $\langle w, label(w)\rangle \in \Sigma^* \times \{+, -\}$ such that if $label(w) = +$ then $w \in L$ and if $label(w) = -$ then $w \notin L$. The size of S is the sum of the lengths of all strings in S.*
2. *An $(\mathbb{L}, \mathbb{R})$-learning algorithm is a program that takes as input a sample of labeled strings and outputs a representation from $\mathbb{R}$.*

Finally, let us recall what "learning" means. We choose to base ourselves on the paradigm of polynomial identification, as defined in [6, 2], since many authors showed that it was both relevant and tractable. Other paradigms are known (*e.g.* PAC-learnability), but they are often either similar to this one or inconvenient for Grammatical Inference problems.

In this paradigm we first demand that the learning algorithm has a running time polynomial in the size of the data from which it is learning from. Next we want the algorithm to converge in some way to a chosen target. Ideally the convergence point should be met very quickly, after having seen a polynomial number of examples. As this constraint is usually too hard, we want the convergence to take place in the limit, *i.e.*, after having seen a finite number of examples. The polynomial aspects then correspond to the size of a minimal *learning* or *characteristic* sample, whose presence should ensure identification. For more details on these models we refer the reader to [6, 2].

3 Defining Languages with String-Rewriting Systems

String-rewriting systems are usually defined as sets of rewrite rules. These rules allow to replace factors by others in strings. However, as we feel that this mechanism is not flexible enough, we would like to extend it. Indeed, a rule that one would like to use at the beginning (prefix) or at the end of a string could also be used in the middle of this string and then have undesirable side effects.

Therefore, we introduce two new symbols \$ and £ that do not belong to the alphabet Σ and will respectively mark the beginning and the end of each string. In other words, we are going to consider strings from the set $\$\Sigma^*£$. As for the rewrite rules, they will be *partially* marked (and thus belong to $\overline{\Sigma^*} = (\lambda + \$)\Sigma^*(\lambda + £)$). Their forms will constrain their uses either to the beginning, or to the end, or to the middle, or to the string taken as a whole. Notice that this solution is an intermediate approach between the usual one and string-rewriting systems with variables introduced in [11].

Definition 2 (Delimited SRS).

- *A delimited rewrite rule is an ordered pair of strings (l, r), generally written $l \to r$, such that l and r satisfy one of the four following constraints:*
  1. *$l, r \in \$\Sigma^*$ (used to rewrite prefixes) or*
  2. *$l, r \in \$\Sigma^*\pounds$ (used to rewrite whole strings) or*
  3. *$l, r \in \Sigma^*$ (used to rewrite factors) or*
  4. *$l, r \in \Sigma^*\pounds$ (used to rewrite suffixes).*

  Rules of type 1 and 2 will be called \$-rules and rules of type 3 and 4 will be called non-\$-rules.
- *By a delimited string-rewriting system (DSRS), we mean any finite set $\mathcal{R}$ of delimited rewrite rules.*

Let $|\mathcal{R}|$ be the number of rules of $\mathcal{R}$ and $\|\mathcal{R}\|$ the sum of the lengths of the strings $\mathcal{R}$ is made of: $\|\mathcal{R}\| = \sum_{(l\to r)\in\mathcal{R}} |lr|$.

Given a DSRS $\mathcal{R}$ and two strings $w_1, w_2 \in \overline{\Sigma^*}$, we say that w_1 *rewrites in one step into* w_2, written $w_1 \to_{\mathcal{R}} w_2$ or simply $w_1 \to w_2$, *iff* there exists a rule $(l \to r) \in \mathcal{R}$ and two strings $u, v \in \overline{\Sigma^*}$ such that $w_1 = ulv$ and $w_2 = urv$. A string w is *reducible iff* there exists w' such that $w \to w'$, and *irreducible* otherwise. *E.g.*, the string $\$aabb\pounds$ rewrites to $\$aaa\pounds$ with rule $bb\pounds \to a\pounds$ and $\$aaa\pounds$ is irreducible. We get immediately the following property:

Proposition 1. *The set $\$\Sigma^*\pounds$ is stable w.r.t. $\to_{\mathcal{R}}$.*

In other words, \$ and £ cannot disappear or move in a string by rewriting.

Let $\to^*_{\mathcal{R}}$ (or simply $\to^*$) denote the reflexive and transitive closure of $\to_{\mathcal{R}}$. We say that w_1 *reduces to* w_2 or that w_2 *is derivable from* w_1 *iff* $w_1 \to^*_{\mathcal{R}} w_2$.

Definition 3 (Language Induced by a DSRS). *Given a DSRS $\mathcal{R}$ and an irreducible string $e \in \Sigma^*$, we define the language $\mathcal{L}(\mathcal{R}, e)$ as the set of strings that reduce to e using the rules of $\mathcal{R}$:*

$$\mathcal{L}(\mathcal{R}, e) = \{w \in \Sigma^* : \$w\pounds \to^*_{\mathcal{R}} \$e\pounds\}.$$

Deciding whether a string w belongs to a language $\mathcal{L}(\mathcal{R}, e)$ or not consists in trying to obtain e from w by a rewriting derivation. However, w may be the starting point of numerous derivations and so, such a task may be really hard. (Nevertheless, remember that we introduced \$ and £ to allow some control...) We will tackle these problems in next section but present some examples first.

Example 1. Let $\Sigma = \{a, b\}$.

- $\mathcal{L}(\{ab \to \lambda\}, \lambda)$ is the Dyck language. Indeed, this single rule erases factors ab, so we get the following example of derivation:

$$\$aabb\underline{ab}\pounds \to \$a\underline{ab}b\pounds \to \$\underline{ab}\pounds \to \$\pounds$$

- $\mathcal{L}(\{ab \to \lambda; ba \to \lambda\}, \lambda)$ is the language $\{w \in \Sigma^* : |w|_a = |w|_b\}$, since every rewriting step erases one a and one b.

– $\mathcal{L}(\{aabb \to ab; \$ab£ \to \$£\}, \lambda) = \{a^n b^n : n \in \mathbb{N}\}$. For instance,

$$\$aa\underline{aabb}bb£ \to \$a\underline{aabb}b£ \to \$\underline{aabb}£ \to \$\underline{ab}£ \to \$£$$

Notice that the rule $\$ab£ \to \$£$ is necessary for λ to belong to the language.
– $\mathcal{L}(\{\$ab \to \$\}, \lambda)$ is the regular language $(ab)^*$. Indeed,

$$\underline{\$ab}abab£ \to \underline{\$ab}ab£ \to \underline{\$ab}£ \to \$£$$

Actually, all regular languages can be induced by a DSRS:

Theorem 1. *For each regular language L, there exist a DSRS $\mathcal{R}$ and a string e such that $L = \mathcal{L}(\mathcal{R}, e)$.*

Proof (Hint). A DSRS that is only made of \$-rules defines a *prefix grammar* [5]. It has been shown that this kind of grammars generates exactly the regular languages.

4 Shaping Learnable DSRS

As already mentioned, a string w belongs to a language $\mathcal{L}(\mathcal{R}, e)$ *iff* one can build a derivation from w to e. However this raises many difficulties. Firstly, one can imagine a DSRS such that a string can be rewritten indefinitely[1]. In other words, an algorithm that would try to answer the problem may loop. Secondly, even if all the derivations induced by a DSRS are finite, they could be of exponential lengths and thus computationally intractable[2].

We first extend the hierarchical order $\trianglelefteq$ to the strings of $\overline{\Sigma^*}$, by defining the *extended hierarchical order*, denoted $\preceq$, as follows: $\forall w_1, w_2 \in \Sigma^*$, if $w_1 \lhd w_2$ then $w_1 \prec \$w_1 \prec w_1£ \prec \$w_1£ \prec w_2$. Therefore, if $a < b$, then $\lambda \lhd a \lhd b \lhd aa \lhd ab \lhd ba \lhd bb \lhd aaa \lhd \ldots$, so $\lambda \prec \$ \prec £ \prec \$£ \prec a \prec \$a \prec a£ \prec \$a£ \prec b \prec \ldots$ The following technical definition ensures that all the rewriting derivations are finite and tractable in polynomial time.

Definition 4 (Hybrid DSRS). *We say that a rule $l \to r$ is*
(i) length-reducing iff $|l| > |r|$ and (ii) length-lexicographic iff $l \succ r$.
A DSRS $\mathcal{R}$ is hybrid iff (i) all \$-rules (whose left hand sides are in $\$\Sigma^*(\lambda + £)$) are length-lexicographic and (ii) all non-\$-rules (whose left hand sides are in $\Sigma^*(\lambda + £)$) are length-reducing.

Theorem 2. *All the derivations induced by a hybrid DSRS $\mathcal{R}$ are finite. Moreover, every derivation starting from a string w has a length that is $\leq |w| \cdot |\mathcal{R}|$.*

[1] Consider the derivations induced by $\{a \to b; b \to a; c \to cc\}\ldots$

[2] Consider the DSRS $\{1£ \to 0£; 0£ \to c1d£; 0c \to c1; 1c \to 0d; d0 \to 0d; d1 \to 1d; dd \to \lambda\}$. All the derivations it induces are finite; Indeed, assuming that $d > 1 > 0 > c$, the left hand side l is lexicographically greater than the right hand side r for all rules $l \to r$, so this DSRS is strongly normalizing [3]. However, it induces the derivation $\$1111£ \to \$1110£ \to^* \$1101£ \to \$1100£ \to^* \$1011£ \to^* \ldots \to^* \$0000£$

Proof. Let $w_1 \rightarrow w_2$ be a single rewriting step. There exists a rule $l \rightarrow r$ and two strings $u, v \in \overline{\Sigma^*}$ such that $w_1 = ulv$ and $w_2 = urv$. Notice that if $|l| > |r|$ then $l \succ r$. Moreover, if $l \succ r$, then we deduce that $w_1 \succ w_2$. So if one has a derivation $w \rightarrow u_1 \rightarrow u_2 \rightarrow \ldots$, then $w \succ u_1 \succ u_2 \succ \ldots$. As $\preceq$ is a good order, there is no infinite and strictly decreasing chain of the form $w \succ u_1 \succ u_2 \succ \ldots$. So every derivation induced by $\mathcal{R}$ is finite. Now let $n \in \mathbb{N}$. Assume that for all strings w' such that $|w'| < n$, the lengths of the derivations starting from w' are at most $|w'| \cdot |\mathcal{R}|$. Let w be a string of length n. We claim that the maximum length of a derivation that would preserve the length of w cannot exceed $|\mathcal{R}|$ rewriting steps. Indeed, all rules that can be used along such a derivation are of the form $\$l \rightarrow \$r$, with $|l| = |r|$ and $l \succ r$; When such a rule is used once, then it cannot be used a second time in the same derivation. Otherwise, there would exists a derivation $\$lu\pounds \rightarrow \$ru\pounds \rightarrow \ldots \rightarrow \$lv\pounds$ with $|u| = |v|$ (since the length is preserved). As $\$ru\pounds \rightarrow^* \$lv\pounds$ and $|l| = |r|$ and $|u| = |v|$, we deduce that $r \succeq l$ which is impossible since $r \prec l$. So there are at most $|\mathcal{R}|$ rewriting steps that preserve the length of w, and then the application of a rule produces a string w' whose length is $< n$. So by induction hypothesis, the length of a derivation starting from w is no more than $|\mathcal{R}| + |w'| \cdot |\mathcal{R}| \leq |w| \cdot |\mathcal{R}|$. □

We saw that a hybrid DSRS induces finite and tractable derivations. Nevertheless, many different irreducible strings may be reached from one given string by rewriting. Therefore, answering the problem "$w \in \mathcal{L}(\mathcal{R}, e)$?" will require to compute *all* the derivations that start with w and check if one of them ends with e. In other words, such a DSRS is a kind of "undeterministic" (thus inefficient) parsing device. An usual way to circumvent this difficulty is to impose our hybrid DSRS to be also Church-Rosser [3].

Definition 5 (Church-Rosser DSRS). *We say that a DSRS $\mathcal{R}$ is Church-Rosser iff for all strings $w, u_1, u_2 \in \overline{\Sigma^*}$ such that $w \rightarrow^* u_1$ and $w \rightarrow^* u_2$, there exists $w' \in \overline{\Sigma^*}$ such that $u_1 \rightarrow^* w'$ and $u_2 \rightarrow^* w'$.*

In the definition above, if $w \rightarrow^* u_1$ and $w \rightarrow^* u_2$ and u_1 and u_2 are irreducible strings, then $u_1 = u_2 (= w')$. So given a string w, there is no more than *one* irreducible string that can be reached by a derivation starting with w, whatever the derivation is considered. However, the Church-Rosser property is undecidable in general [3], so we constrain our DSRS to fulfill a restrictive condition:

Definition 6 (ANo DSRS). *A DSRS $\mathcal{R}$ is almost nonoverlapping (ANo) iff for all rules $R_1 = l_1 \rightarrow r_1$ and $R_2 = l_2 \rightarrow r_2$ of $\mathcal{R}$:*

i. if $l_1 = l_2$ then $r_1 = r_2$;
ii. if l_1 is strictly included in l_2: $\exists u, v \in \Sigma^*, ul_1v = l_2, uv \neq \lambda$, then $ur_1v = r_2$;
iii. if a strict suffix of l_1 is a strict prefix of l_2:
$\exists u, v \in \Sigma^*, l_1u = vl_2, 0 < |v| < |l_1|$, then $r_1u = vr_2$.

Notice that if R_1 does not overlap R_2, then R_2 may still overlap R_1.

Theorem 3. *Every ANo DSRS is Church-Rosser. Moreover, every subsystem of an ANo DSRS is an ANo DSRS, and thus Church-Rosser.*

Proof. Let us show that an ANo DSRS $\mathcal{R}$ induces a rewriting relation $\rightarrow_{\mathcal{R}}$ that is *subcommutative* [7]. Let us write $w_1 \rightarrow_\varepsilon w_2$ *iff* $w_1 \rightarrow_{\mathcal{R}} w_2$ or $w_1 = w_2$. We claim that for all w, u_1, u_2, if $w \rightarrow_{\mathcal{R}} u_1$ and $w \rightarrow_{\mathcal{R}} u_2$, then there exists a string w' such that $u_1 \rightarrow_\varepsilon w'$ and $u_2 \rightarrow_\varepsilon w'$. Indeed, assume that $w \rightarrow_{\mathcal{R}} u_1$ uses a rule $R_1 = l_1 \rightarrow r_1$ and $w \rightarrow_{\mathcal{R}} u_2$ uses a rule $R_2 = l_2 \rightarrow r_2$. If both rewriting steps are independent, *i.e.*, $w = xl_1yl_2z$ for some strings x, y, z, then $u_1 = xr_1yl_2z$ and $u_2 = xl_1yr_2z$; Obviously, $u_1 \rightarrow_{\mathcal{R}} w'$ and $u_2 \rightarrow_{\mathcal{R}} w'$ with $w' = xr_1yr_2z$. Otherwise, R_1 overlaps R_2 (or vice-versa), and so $u_1 = u_2$, since $\mathcal{R}$ is ANo. An easy induction allows to generalize this property to derivations: If $w \rightarrow^*_{\mathcal{R}} u_1$ and $w \rightarrow^*_{\mathcal{R}} u_2$ then there exists w' such that $u_1 \rightarrow^*_\varepsilon w'$ and $u_2 \rightarrow^*_\varepsilon w'$, where $\rightarrow^*_\varepsilon$ is the reflexive and transitive closure of $\rightarrow_\varepsilon$. Finally, as $u_1 \rightarrow^*_\varepsilon w'$ and $u_2 \rightarrow^*_\varepsilon w'$, we deduce that $u_1 \rightarrow^*_{\mathcal{R}} w'$ and $u_2 \rightarrow^*_{\mathcal{R}} w'$. □

Finally, we get the following properties with our DSRS: (1) For all strings w, there is no more than *one* irreducible string that can be reached by a derivation which starts with w, whatever the derivation is considered. This irreducible string will be called the *normal form* of w and denoted $w\downarrow$. (2) No derivation can be prolonged indefinitively, so every string w has at least one normal form. And whatever the way a string w is reduced, the rewriting steps produce strings that are ineluctably closer and closer to $w\downarrow$. An important consequence is that one has an immediate algorithm to check whether $w \in \mathcal{L}(\mathcal{R}, e)$ or not: One only needs to (i) compute the normal form $w\downarrow$ of w and (ii) check if $w\downarrow$ and e are *syntactically* equal. As all the derivations have polynomial lengths, this algorithm is polynomial in time.

5 Learning Languages Induced by DSRS

In this section we present our learning algorithm and its properties. The idea is to enumerate the rules following the order $\preceq$. We discard those that are useless or inconsistent *w.r.t.* the data, and those that break the ANo condition.

The first thing LARS does is to compute all the factors of S_+ and to sort them *w.r.t.* $\preceq$. Left and right hand sides of the rules will be chosen in this set since it is reasonable to think that the positive examples contain all information that is needed to learn the target language. This assumption reduces dramatically the search space. LARS then enumerates the elements of this set thanks to two "for" loops, which allows to build the candidate rules.

Function `is_useful` discards the rules that cannot be used to rewrite at least one string of the current set I_+ (and are thus useless). Function `type` returns an integer in $\{1, 2, 3, 4\}$ and allows to check if the candidate rule is syntactically correct according to Def.2. Function `is_ANo` avoids the rules that would produce non ANo DSRS. Notice that a candidate rule that passes all these tests with success ensures that the DSRS will be syntactically correct, hybrid and ANo. The last thing to check is that the rule is consistent with the data, *i.e.*, that it does not produce a string belonging to both I_+ and I_-. This is easily performed by computing the normal forms of the strings of I_+ and I_-, which is the aim of function `normalize`.

Algorithm 1: LARS (Learning Algorithm for Rewriting Systems)

```
Data   : a sample ⟨S+, S−⟩
Result : ⟨R, e⟩ where R is a hybrid ANo DSRS and e is an irreducible string
begin
    R ⟵ ∅; I+ ⟵ S+; I− ⟵ S−;
    F ⟵ sort≼ {v : ∃u, w ∈ Σ̄*, uvw ∈ I+};
    for i = 1 to |F| do
        if is_useful(F[i],I+) then
            for j = 0 to i − 1 do
                if type(F[i]) = type(F[j]) then
                    S ⟵ R ∪ {F[i] → F[j]};
                    if is_ANo(S) then
                        E+ ⟵ normalize(I+,S); E− ⟵ normalize(I−,S);
                        if E+ ∩ E− = ∅ then
                            R ⟵ S; I+ ⟵ E+; I− ⟵ E−;
    e ⟵ min≼ I+;
    foreach w ∈ I+ do
        if w ≠ e then R ⟵ R ∪ {w → e};
    return ⟨R, e⟩;
end
```

Theorem 4. *Given a sample $\langle S_+, S_-\rangle$ of size m, algorithm* LARS *returns a hybrid ANo DSRS $\mathcal{R}$ and an irreducible string e such that $S_+ \subseteq \mathcal{L}(\mathcal{R}, e)$ and $S_- \cap \mathcal{L}(\mathcal{R}, e) = \emptyset$. Moreover, its execution time is a polynomial of m.*

Proof (Hint). The termination and polynomiality of LARS is straightforward. Moreover, the following four invariant properties are maintained all along the double "for" loops: (1) $\mathcal{R}$ is a hybrid ANo DSRS, (2) I_+ contains all and only the normal forms of the strings of S_+ *w.r.t.* $\mathcal{R}$, (3) I_- contains all and only the normal forms of the strings of S_- *w.r.t.* $\mathcal{R}$ and (4) $I_+ \cap I_- = \emptyset$. Clearly, these properties remain true before the "foreach" loop. Now at the end of the last "foreach" loop, it is clear that: (1) $\mathcal{R}$ is a hybrid ANo DSRS, (2) e is the normal form of all the strings of S_+, so $S_+ \subseteq \mathcal{L}(\mathcal{R}, e)$ and (3) the normal forms of the strings of S_- are all in I_- and $e \notin I_-$, so $S_- \cap \mathcal{L}(\mathcal{R}, e) = \emptyset$. □

We now establish an identification theorem for LARS. This theorem focuses on languages that may be defined thanks to special DSRS that we define now. We begin with the notion of consistent rule that characterizes the rules that LARS will have to find.

Definition 7 (Consistent Rule). *We say that a rule $R = l \to r$ is consistent w.r.t. a language $L \subseteq \Sigma^*$ iff $\forall u, v \in \overline{\Sigma^*}$, if $ulv \notin \$L\pounds$, then $urv \notin \$L\pounds$.*

Definition 8 (Closed DSRS). *Let $L = \mathcal{L}(\mathcal{R}, e)$ be a language and R_{max} the greatest*[3] *rule of $\mathcal{R}$ w.r.t. $\preceq$. We say that $\mathcal{R}$ is closed iff: (i) $\mathcal{R}$ is hybrid and ANo, and (ii) for all length-lexicographic \$-rules and all length-decreasing non-\$-rules S, if $S \preceq R_{max}$ and $S \notin \mathcal{R}$, then S is not consistent with L.*

We do not know whether this property is decidable or not. This is a work in progress. Nevertheless, this notion allows to get the following result:

Theorem 5. *Given a language $L = \mathcal{L}(\mathcal{R}, e)$ such that $\mathcal{R}$ is closed, there exists a finite characteristic sample $\langle CS_+, CS_- \rangle$ such that, on $\langle S_+, S_- \rangle$ with $CS_+ \subseteq S_+$ and $CS_- \subseteq S_-$, algorithm* LARS *finds e and returns a hybrid ANo DSRS $\mathcal{R}'$ such that $\mathcal{L}(\mathcal{R}', e) = \mathcal{L}(\mathcal{R}, e)$.*

Notice that the polynomiality of the characteristic sets is not established.

Proof (Hint). Let $L = \mathcal{L}(\mathcal{T}, e)$ be the target language. $\mathcal{T}$ is assumed closed. Let us first define CS_+ and CS_-:

1. For all $\mathcal{R} \subseteq \mathcal{T}$ and all $R \in \mathcal{T}$ such that $\mathcal{L}(\mathcal{R}, e) \neq L$ but $\mathcal{L}(\mathcal{R} \cup \{R\}, e) = L$, there exists $w = ulv \in \$L\pounds \setminus \$\mathcal{L}(\mathcal{R}, e)\pounds$ such that $w \in CS_+$, where $R = l \to r$. (Notice that if $\mathcal{L}(\mathcal{R}, e) \neq L$, then $\mathcal{L}(\mathcal{R}, e) \subset L$ since $\mathcal{R} \subseteq \mathcal{T}$.)
2. For all rules $l \to r \in \mathcal{T}$, there exists $u, v \in \overline{\Sigma^*}$ such that $ulv \in \$L\pounds \cap CS_+$ and $urv \in \$L\pounds \cap CS_+$.
3. For all length-lexicographic \$-rules and all length-decreasing non-\$-rules $R = l \to r \notin \mathcal{T}$, if $\mathcal{T} \cup \{R\}$ is ANo, then there exists $u, v \in \overline{\Sigma^*}$ such that $ulv \in (\overline{\Sigma^*} \setminus \$L\pounds) \cap CS_-$ and $urv \in \$L\pounds \cap CS_+$.

We now prove that if $S_+ \supseteq CS_+$ and $S_- \supseteq CS_-$ LARS returns a correct system. By construction of the characteristic set, F contains all the left and right hand sides of the rules of the target. Assume now that LARS has been run during a certain number of steps; Let $\mathcal{R}$ be the current hybrid ANo DSRS. As I_+ is not empty, let $m = \min_{\preceq} I_+$ and $\widehat{\mathcal{R}} = \mathcal{R} \cup \{w \to m : w \in I_+, w \neq m\}$. Notice that $\widehat{\mathcal{R}}$ is also a hybrid ANo DSRS. Finally let $\widehat{L} = \mathcal{L}(\widehat{\mathcal{R}}, m)$.

Let $R = l \to r$ be the next rule to be checked, *i.e.*, $l = F[i]$ and $r = F[j]$. We assume that R is well-typed and $\mathcal{R} \cup \{R\}$ is ANo, otherwise R does not belong to $\mathcal{T}$ and LARS discards it. There are two cases:

1. If R is inconsistent, then there exists $m = ulv \in (\overline{\Sigma^*} \setminus \$L\pounds) \cap CS_-$ and $m' = urv \in \$L\pounds \cap CS_+$. So $m \downarrow_{\widehat{\mathcal{R}}} \in I_-$, $m' \downarrow_{\widehat{\mathcal{R}}} \in I_+$ and LARS discards R.
2. If R is consistent, then consider system $\mathcal{S} = \widehat{\mathcal{R}} \cup \{T \in \mathcal{T} : R \prec T\}$. Either $\mathcal{L}(\mathcal{S}, e) = L$ and then rule R is not needed (but can be added with no harm). Or $\mathcal{L}(\mathcal{S}, e) \neq L$ and then there is a string $w = ulv$ in CS_+ (where $R = l \to r$). As $w \downarrow_{\widehat{\mathcal{R}}} \in I_+$ and $w \downarrow_{\widehat{\mathcal{R}}} = u'lv'$ (because $\widehat{\mathcal{R}} \cup \{R\}$ is Church-Rosser), this means that l is a factor of a string I_+, which is consistent, so LARS adds R to $\mathcal{R}$. □

[3] $\preceq$ is basically extended to ordered pairs of strings, thus to rules, as follows: $\forall u_1, u_2, v_1, v_2 \in \overline{\Sigma^*}, (u_1, u_2) \preceq (v_1, v_2)$ *iff* $u_1 \prec v_1$ or ($u_1 = v_1$ and $u_2 \preceq v_2$).

6 Experimental results

We present in this section some specific languages for which rewriting systems exist, and on which the algorithm LARS has been tested. In each case we describe the task, the learning set from which the algorithm has worked. We do not report any runtimes here as all computations took less than one second: Both the systems and the learning sets were small.

Dyck Languages. The language of all bracketed strings or balanced parentheses is classical in formal language theory. It is usually defined by the rewriting system $\langle\{ab \rightarrow \lambda\}, \lambda\rangle$. The language is context-free and can be generated by the grammar $\langle\{a, b\}, \{S\}, P, S\rangle$ with $P = \{S \Rightarrow aSbS; S \Rightarrow \lambda\}$. The language is learned in [18] from all positive strings of length up to 10 and all negative strings of length up to 20. In [12] the authors learn it from all positive and negative strings within a certain length, typically from five to seven. Algorithm LARS learns the correct grammar from both types of learning sets but also from much smaller sets of about 20 strings. Alternatively [16] have tested their GRIDS system on this language, but when learning from positive strings only. They do not identify the language. It should also be noted that the language can be modified to deal with more than one pair of brackets and remains learnable.

Language $\{a^n b^n : n \in \mathbb{N}\}$. Language $\{a^n b^n : n \in \mathbb{N}\}$ is a language often used as a context-free language that is not regular. The corresponding system is $\langle\{aabb \rightarrow ab; \$ab£ \rightarrow \$£\}, \lambda\rangle$. Variants of this language are $\{a^n b^n c^m : m, n \in \mathbb{N}\}$ which is studied in [18], or $\{a^m b^n : 1 \leq m \leq n\}$ from [12]. In all cases algorithm LARS has learned the intended system from as few as 20 examples, which is much less than for previous methods.

Regular languages. We have run algorithm LARS on benchmarks for regular language learning tasks. There are several such benchmarks. Those related to the ABBADINGO [9] tasks were considered too hard for LARS: As we have constructed a deterministic algorithm (in the line for instance of RPNI [15]) results when the required strings are not present are bad. We turned to smaller benchmarks, as used in earlier regular inference tasks [4]. These correspond to small automata, and thus to from 1 to 6 rewriting rules. In most cases LARS found a correct system, but when it did not the error was important.

Other languages and properties. Languages $\{w \in \{a, b\}^* : |w|_a = |w|_b\}$ and $\{w \in \{a, b\}^* : 2|w|_a = |w|_b\}$ are used in [12]. In both cases the languages can be learned by LARS from less than 30 examples.

Language of Lukasewitz is generated by grammar $\langle\{a, b\}, \{S\}, P, S\rangle$ with $P = \{S \Rightarrow aSS + b\}$. The intended system is $\langle\{abb \rightarrow b\}, b\rangle$ but what LARS returned was $\langle\{\$ab \rightarrow \lambda; aab \rightarrow a\}, b\rangle$, which is correct.

Language $\{a^m b^m c^n d^n : m, n \in \mathbb{N}\}$ is not linear (but then Dyck isn't either) and is recognized by system $\langle\{aabb \to ab; ccdd \to cd\}, abcd\rangle$.

On the other hand the language of palindromes ($\{w : w = w^R\}$) does not admit a DSRS, unless the centre is some special character. [12] learn this language whereas LARS cannot.

System $\langle\{ab^k \to b\}, b\rangle$ requires an exponential characteristic sample so learning this language with LARS is a hard task.

The system has also been tested on the Omphalos competition training sets without positive results. There are two explanations to this: On one hand LARS being a deterministic algorithm needs a restrictive learning set to converge (data or evidence driven methods would be more robust and still need to be investigated), and on the other hand, there is no means to know if the target languages admit rewriting systems with the desired properties.

7 Conclusion and Future Work

In this paper, we have investigated the problem of learning languages that can be defined with string-rewriting systems (SRS). We have first tailored a definition of "hybrid almost nonoverlapping delimited SRS", proved that they were efficient (often linear) parsing devices and showed that they allowed to define all regular languages as well as famous context-free languages (Dyck, Lukasewitz, $\{a^n b^n : n \geq 0\}$, $\{w \in \{a, b\}^* : |w|_a = |w|_b\}$, ...). Then we have provided an algorithm to learn them, LARS, and proved that it could identify, in polynomial time (but not data), the languages whose SRS had some "closedness" property. Finally, we have shown that LARS was capable of learning several languages, both regular and not.

However, much remains to be done on this topic. On the one hand, LARS suffers from its simplicity, as it failed in solving the (hard) problems of the Omphalos competition. We think that we could improve our algorithm either by pruning our exploration of the search space, or by studying more restrictive SRS (*e.g.*, special or monadic SRS [1]), or by investigating more sophisticated properties (such as *basicity*). On the other hand, other kind of SRS can be used to define languages, such as the CR-languages of McNaugthon [11], or the DL0 systems (that can generate deterministic *context-sensitive* languages). All these SRS may be the source of new attractive learning results in Grammatical Inference.

References

1. R. Book and F. Otto. *String-Rewriting Systems.* Springer-Verlag, 1993.
2. C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning Journal*, 27:125–138, 1997.
3. N. Dershowitz and J. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science : Formal Methods and Semantics*, volume B, chapter 6, pages 243–320. North Holland, Amsterdam, 1990.

4. P. Dupont. Regular grammatical inference from positive and negative samples by genetic search: the GIG method. In R. C. Carrasco and J. Oncina, editors, *Grammatical Inference and Applications, Proceedings of ICGI '94*, number 862 in LNAI, pages 236–245, Berlin, Heidelberg, 1994. Springer-Verlag.
5. M. Frazier and C.D. Page Jr. Prefix grammars: An alternative characterisation of the regular languages. *Information Processing Letters*, 51(2):67–71, 1994.
6. E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.
7. J. W. Klop. Term rewriting systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 1–112. Oxford University Press, 1992.
8. T. Koshiba, E. Mäkinen, and Y. Takada. Inferring pure context-free languages from positive data. *Acta Cybernetica*, 14(3):469–477, 2000.
9. K. Lang, B. A. Pearlmutter, and R. A. Price. The Abbadingo one DFA learning competition. In *Proceedings of ICGI'98*, pages 1–12, 1998.
10. S. Lee. Learning of context-free languages: A survey of the literature. Technical Report TR-12-96, Center for Research in Computing Technology, Harvard University, Cambridge, Massachusetts, 1996.
11. R. McNaughton, P. Narendran, and F. Otto. Church-Rosser Thue systems and formal languages. *Journal of the Association for Computing Machinery*, 35(2):324–344, 1988.
12. K. Nakamura and M. Matsumoto. Incremental learning of context-free grammars. In P. Adriaans, H. Fernau, and M. van Zaannen, editors, *Grammatical Inference: Algorithms and Applications, Proceedings of ICGI '02*, volume 2484 of *LNAI*, pages 174–184, Berlin, Heidelberg, 2002. Springer-Verlag.
13. C. Nevill-Manning and I. Witten. Identifying hierarchical structure in sequences: a linear-time algorithm. *Journal of Artificial Intelligence Research*, 7:67–82, 1997.
14. M. Nivat. On some families of languages related to the dyck language. In *Proc. 2nd Annual Symposium on Theory of Computing*, 1970.
15. J. Oncina and P. García. Identifying regular languages in polynomial time. In H. Bunke, editor, *Advances in Structural and Syntactic Pattern Recognition*, volume 5 of *Series in Machine Perception and Artificial Intelligence*, pages 99–108. World Scientific, 1992.
16. G. Petasis, G. Paliouras, V. Karkaletsis, C. Halatsis, and C. Spyropoulos. E-grids: Computationally efficient grammatical inference from positive examples. *to appear in Grammars*, 2004.
17. Y. Sakakibara. Recent advances of grammatical inference. *Theoretical Computer Science*, 185:15–45, 1997.
18. Y. Sakakibara and M. Kondo. Ga-based learning of context-free grammars using tabular representations. In *Proceedings of 16th International Conference on Machine Learning (ICML-99)*, pages 354–360, 1999.
19. B. Starkie, F. Coste, and M. van Zaanen. Omphalos context-free language learning competition. `http://www.irisa.fr/Omphalos`, 2004.
20. T. Yokomori. Polynomial-time identification of very simple grammars from positive data. *Theor. Comput. Sci.*, 1(298):179–206, 2003.