Fundamenta Informaticae XX (2008) 1–15
IOS Press

Observation of String-Rewriting Systems

Matteo Cavaliere*
Department of Computer Science and Artificial Intelligence
University of Sevilla
eMail: martew@inwind.it

Peter Leupold*
Research Group on Mathematical Linguistics
Rovira i Virgili University, Tarragona
eMail: klauspeter.leupold@estudiants.urv.es

Abstract. In most models of computation, a device performs some type of process, and only some final output is regarded as the result. In adding an observer to such a device, one can obtain a protocol of the entire process and then use this as the computation's result. In a series of articles this approach has proved to often exceed greatly the power of the system observed.

Here we apply this architecture to string-rewriting systems. They receive a string as input, and a combination of observer and decider then determines whether this string is accepted. This decision is based only on the rewriting process performed. First we determine the powers of painter, context-free, and inverse context-free rewriting systems in terms of McNaughton languages. Then these are investigated as components of rewriting/observer systems, and we obtain characterizations of the context-sensitive and the recursively enumerable languages. Finally we investigate some limitations, i.e. characterize some systems, where observation does not increase power.

Keywords: String-Rewriting, McNaughton Languages, Observation.

Address for correspondence: Research Group on Mathematical Linguistics, Pça. Imperial Tàrraco, 43005 Tarragona, Catalunya, España.
*This work was done, while both authors were funded by the Spanish Ministry of Culture, Education and Sport under the Programa Nacional de Formación de Profesorado Universitario (FPU).

1. Introduction

In a series of recent articles, a new approach to look at computations has been explored. The original motivation came from the emerging field of biologically inspired computing. In most formal models there, a soup of molecules is left to itself to react and rearrange, or is manipulated according to some routine, cf. for example the survey of several such models by Păun et al. [12].

Common to all these approaches is the proceeding of letting the system work according to its rules and then, after it halts – or by taking its action in discrete steps to infinite iteration – all or some of the resulting molecules are taken to constitute the result. This is what one is used to from classical models of computation like the Turing Machine, and also from practical devices like electrical computers.

However, when looking at the manner in which many chemical or biological experiments are conducted and evaluated, the final state is often rather irrelevant, and rather the entire process constitutes the result; take for example the monitoring of the size of a cell colony or the relation between hunter and prey. In both cases a momentary picture is worthless, because it does not reflect any of the dynamics that one is really interested in.

Inspired by this, a new architecture to look at biologically inspired models of computation was first introduced for membrane systems: membrane/observer systems [4]. Here an observer maps every configuration of a membrane system during its computation to one single letter; the configurations in this case are multisets. The sequence of all the letters of a halting computation then constitutes a word generated by the entire system.

Of course, for the mechanism as such, the original motivation of the underlying observed system is irrelevant. Thus it was a natural next step to explore also the power of observing more classical devices like generative grammars [5]. In both cases it was found that an essentially context-free underlying system with a regular observer suffices to construct an universal *generative device*. So the combination of components in the architecture proposed really results in a significant increase in computational power as compared to the one of the individual components.

In a first attempt to apply this approach also to *accepting languages*, conventional grammars were used and then stripped of several of their components [6]. However, they are not completely appropriate as underlying systems, because for example, they always start from the same input symbol rather than from an arbitrary input word. Essentially, just some rules are needed according to which this input is manipulated. Such systems of rules have been already studied under the name of string-rewriting systems.

In the constellation investigated here, we let such a string-rewriting system work on an input string. After every rewrite step a finite state machine, the observer, maps the entire string to one single letter over a different alphabet. In this way every possible configuration is in some sense categorized into one of a finite set of classes represented by the observer's output alphabet's letters. The concatenation of these observations in chronological order constitutes the entire observation. Finally, a finite automaton, the decider, reads the entire observation and, based only on this, judges whether the original input word is accepted or not. We schematically depict the architecture of this accepting device, called *rewriting/observer system* in Figure 1.

Figure 1. The architecture of a rewriting/observer system.

A somewhat comparable mechanism was described by Ilie and Salomaa [9] in a characterization of recursively enumerable languages. There two systems of rules –just like the painters we will use below– alternate according to a regular control mechanism, and in this way computational universality is achieved. So they use one rule system more, and on the other hand one level of control less to construct an equally powerful mechanism.

Using string-rewriting systems instead of grammars, it then seems more appropriate to compare the power of the rewriting/observer architecture directly to that of the underlying string-rewriting systems instead of similar generative grammars. While the main focus of research on string-rewriting has been on topics like confluence, decidability issues etc. [3], also their accepting power in terms of formal languages has been investigated under the name of McNaughton languages. They were first introduced in the special case of Church-Rosser languages [11], later investigated in a more general setting by Beaudry et al. [1].

In this article we first recall some basic facts about string-rewriting systems and the McNaughton languages associated with them. In Section 3 we introduce painter rewriting systems and determine their power in terms of McNaughton languages; the same is done for context-free and inverse context-free systems. The central notion of this article, rewriting/observer systems as sketched above, are introduced in Section 4, and a variant equivalent to Turing Machines is introduced, as well as two others, which are equivalent to Linear Bounded Automata. Then we investigate in Section 5 some variants with rewriting systems, which do not gain as much power through the combination with decider and observer like those of Section 4. This enables us to identify some key features, which lead to an especially big increase in power, when a string-rewriting system is used in the architecture with an observer and a decider.

Some of the results of Section 4 of this article translate results presented earlier for recognizing grammar/observer systems [6] at MCU 2004, St. Petersburg, to the framework presented here; in the cited article the reader can also find further motivations, which apply in the current context as well.

2. String-Rewriting and McNaughton Languages

Concerning string-rewriting systems, we follow notations and terminology as exposed by Book and Otto [3]. We only recall briefly the most basic notions needed here. By Σ we usually denote a finite alphabet, and then Σ^* is the set of all strings over this alphabet, including the empty string λ. For a string w we denote by $|w|$ its length.

Definition 2.1. A *string-rewriting system* R on an alphabet Σ is a subset of $\Sigma^* \times \Sigma^*$. Its elements are called *rewrite rules*, and are written either as ordered pairs (ℓ, r) or as $\ell \to r$ for $\ell, r \in \Sigma^*$. By $\mathsf{Dom}(R) := \{\ell : \exists r((\ell, r) \in R)\}$ and $\mathsf{Range}(R) := \{r : \exists \ell((\ell, r) \in R)\}$ we denote the set of all left-hand respectively right-hand sides of rules in R.

The *single-step reduction relation* induced by R is defined for any $u, v \in \Sigma^*$ as $u \Rightarrow_R v$ iff there exists an $(\ell, r) \in R$ and words $w_1, w_2 \in \Sigma^*$ such that $u = w_1 \ell w_2$ and $v = w_1 r w_2$. The *reduction relation* $\overset{*}{\Rightarrow}_R$ is the reflexive, transitive closure of $\Rightarrow_R$. When we want to emphasize that a certain number k of rewrite steps is necessary, we write $\overset{k}{\Rightarrow}_R$, and if the rewriting system used is clear, we will write simply $\Rightarrow$, omitting its name.

A string w is called *irreducible* with respect to R, if no rewrite rule from R can be applied to it, i.e. it does not contain any factor from $\mathsf{Dom}(R)$. The set of all such strings is denoted by $IRR(R)$.

By imposing restrictions on the set of rewriting rules, many special classes of rewriting systems can be defined. Following Hofbauer and Waldmann [8] we will call a rule (ℓ, r) *context-free* (*inverse context-free*), if $|\ell| \leq 1$ ($|r| \leq 1$). The class of rewriting-systems with only (inverse) context-free rules we denote by `CF` (`InvCF`). A system is *monadic*, if it is inverse context-free and for all its rewrite rules (ℓ, r) we have $|\ell| > |r|$. The class of monadic systems is denoted by `mon`.

A final notion from classical rewriting-theory is confluence. A system R is *confluent*, if for all words w, u, v we have that $w \Rightarrow_R^* u$ and $w \Rightarrow_R^* v$ imply the existence of a word w' such that $u \Rightarrow_R^* w'$ and $v \Rightarrow_R^* w'$ hold. To the names of classes of rewriting systems of this type we will add the prefix `con`.

For a string-rewriting system R we introduce the notation of $R(w)$ for denoting the set of sequences of configurations it can go through when starting from an input word w. These configurations are simple strings here. For example, for $R = \{(a, b), (a, c), (b, c)\}$ over the alphabet $\{a, b, c\}$ we have

$$R(ab) = \{\langle ab, cb, cc\rangle, \langle ab, ac, cc\rangle, \langle ab, bb, cb, cc\rangle, \langle ab, bb, bc, cc\rangle, \langle ab, ac, bc, cc\rangle\}.$$

Now we need to turn our attention to a manner of obtaining formal languages from string-rewriting systems. This was defined by McNaughton et al. [11], later investigated in more detail by Beaudry et al. [1]. Finally, Woinowski formalized this in so-called *Church-Rosser language systems* [14]. We do not need to use the entire formalism of these systems here and therefore simply say that a language $L \subseteq \Sigma^*$ is a McNaughton language of a string-rewriting system R, iff there exist an alphabet Γ containing Σ, strings $t_1, t_2 \in (\Gamma \setminus \Sigma)^* \cap IRR(R)$ and a letter $Y \in \Gamma$ such that for every word $w \in \Sigma^*$ we have $w \in L$ if and only if $t_1 w t_2 \stackrel{*}{\Rightarrow}_R Y$. This is denoted by $L \in R$-`McNL`.

A class of string-rewriting systems $\mathcal{S}$ defines its corresponding *McNaughton family of languages* $\mathcal{S}$-`McNL` in the canonical way that $\mathcal{S}$-`McNL` consists of all languages accepted by at least one rewriting system from the class $\mathcal{S}$. Without restrictions, string-rewriting systems are computationally universal in this sense.

Theorem 2.1. [1] The family of all McNaughton languages coincides with the class of recursively enumerable sets.

Other language classes from the theory of generative grammars that we will use are the finite, regular, context-free, context-sensitive, and recursively enumerable languages, which we will denote by FIN, REG, CFL, CS and RE respectively. For details about these we refer to the books of Harrison [7] or Salomaa [13].

3. Simple String-Rewriting Systems: Painters

A first, very simple class of string-rewriting systems that we will investigate are so-called *painters*. These are simple in the sense that they allow only two very elementary types of rules: ones renaming a letter to another, and ones deleting single letters. From the first type of rules stems their name, because the replacement of a letter can be interpreted as a repainting of the respective position. More formally, a rewriting system (Σ, R) is called a painter system, iff for all $(\ell, r) \in R$ we have $|\ell| = 1$ and $|r| \leq 1$. The class of painter systems we denote by `PA`, for confluent systems by `con-PA`.

As might be expected, both types of systems are not very powerful as acceptors of McNaughton languages.

Theorem 3.1. The McNaughton languages over an alphabet Σ accepted by painter systems are characterized as follows:

1. A language is in con-PA-McNL, iff it can be represented as A^*BA^* where A and B are disjoint subsets of Σ.
2. A language is in PA-McNL, iff it can be represented as A^*BA^* where A and B are arbitrary subsets of Σ.

Proof:
A first observation is that the strings t_1 and t_2 from the definition of McNaughton languages remain unchanged here. Since they are by definition irreducible, and since all rules have left sides of length one, they cannot be reduced together with any input letter, and thus they do not influence the reduction of the start word w in any way.

Now to case (i): due to confluence, a letter that can be derived to Y, will always be derived to Y. Therefore at most one of these letters can be contained in an accepted word. All other letters must be reduced to λ, any letter reduced to any other irreducible letter will cause non-acceptance. These three cases are formally represented by the following three sets for a given a painter system R over the alphabet Σ:

$$A := \{x \in \Sigma : x \overset{*}{\Rightarrow}_R \lambda\}$$
$$B := \{x \in \Sigma : x \overset{*}{\Rightarrow}_R Y\}$$
$$C := \{x \in \Sigma : \exists z(z \in IRR(R) \wedge x \overset{*}{\Rightarrow}_R z)\}.$$

By the explanation it is clear that the sets A and B correspond to the equally named ones in the theorem's statement and therefore provide the claimed description of $L(R)$. Note that B might also be the empty set, then either $t_1 = Y$ or $t_2 = Y$.

In case (ii), a letter reducible to Y might at the same time be reducible to λ, therefore it can occur more than once. The acceptance condition is that there is one occurrence of a letter that can be reduced to Y, and that all others can be reduced to λ. Thus the same sets as in case (i) provide the characterization, only now A and B might not be disjoint. Letters reducible to both Y and λ are contained in both of them.

The inverse inclusions in both cases should now be obvious. □

As explained in the proof, for string-rewriting systems R, where all elements of $\mathsf{Dom}(R)$ are not longer than one, there can be no interaction between the strings t_1 and t_2 and the input string w from the definition of McNaughton languages. Thus for the sake of simplicity, we will also use a slightly modified definition leaving away these two strings t_1 and t_2 in contexts, where this does not affect the validity of the results presented.

So we see that the McNaughton languages of painter systems are a rather simple subclass of the regular languages. Even many finite languages cannot be accepted.

Example 3.1. The language $\{ab\}$ cannot be accepted by any painter system. To reduce the word ab (possibly with additional strings t_1 and t_2) to the special letter Y, at least one of the two letters a and b would have to be reduced to λ. Without restriction of generality we assume $b \overset{*}{\Rightarrow} \lambda$. Then we have $abb \overset{*}{\Rightarrow} ab \overset{*}{\Rightarrow} Y$, and also the word abb is accepted, indeed the entire language b^*ab^+.

An example for the difference between the classes con-PA-McNL and PA-McNL is the language $b^+ \cup b^*ab^*$. It is accepted by the non-confluent string-rewriting system $\{(b, Y), (b, \lambda), (a, Y)\}$. However, it

cannot be accepted by any confluent system: the words a and bb are both in the language, this implies $a \stackrel{*}{\Rightarrow} Y$ and $bb \stackrel{*}{\Rightarrow} Y$, and these two imply $abb \stackrel{*}{\Rightarrow} YY$. This word is irreducible just as Y, since we deal onlywith painter rules. Because of confluence, YY is the only irreducible word derivable from abb, and therefore abb cannot be accepted although it is also in the language.

The following inclusions are quite clear, their properness has just been illustrated by the examples given.

Corollary 3.1. $\texttt{con-PA-McNL} \subsetneq \texttt{PA-McNL} \subsetneq REG$

For grammars as generating devices, adding context-free rules to a regular grammar can greatly increase its power. As stated before, we will call *context-free* all rules with left sides of length one. Adding such rules to a painter system does not – in contrast to the situation for grammars – increase its power.

Theorem 3.2. $\texttt{con-CF-McNL} = \texttt{con-PA-McNL}$ and $\texttt{CF-McNL} = \texttt{PA-McNL}$.

Proof:
Essentially, the same partition of the input alphabet as in the proof of Proposition 3.1 applies. Due to the context-freeness of the rules, we can still look at each letter separately, even if it might be expanded to a string longer than one during the reduction. Still the only important factors are, whether λ and Y are deriveable; and due to the context-freeness of the rules this depends exclusively on every letter by itself without regard of the neighboring ones. □

Another class that will be of interest are the inverse context-free string-rewriting systems; their McNaughton languages can be characterized nicely with a class of languages from the classical Chomsky hierarchy.

Theorem 3.3. $\texttt{InvCF-McNL} = CFL$.

Proof:
The inclusion $CFL \subseteq \texttt{InvCF-McNL}$ is clear, because by inverting all rules of a context-free grammar, we obtain an inverse context-free rewriting system, which reduces a word to the original grammar's start symbol, iff the word can be generated by that grammar.

For the converse inclusion, let $L \subseteq \Sigma^*$ be a language accepted by a string-rewriting system R from the class $\texttt{InvCF-McNL}$ with strings t_1 and t_2 and special symbol Y as in the definition of McNaughton languages. For all rewrite rules from the system R, their inversions are context-free (admitting also deleting rules $A \to \lambda$), and by setting Y as the start symbol we obtain a context-free language L'. The intersection $L' \cap t_1\Sigma^*t_2$ is then exactly L and is, of course, also context-free. □

This is especially interesting in the light of an earlier result by Beaudry et al., [1]

Theorem 3.4. [1] $\texttt{Mon-McNL} = CFL$.

Thus adding inverse context-free, but not monadic rules to a monadic string-rewriting system does not increase its power in terms of McNaughton languages, and despite $\texttt{Mon} \subsetneq \texttt{InvCF}$ the equality $\texttt{Mon-McNL} = \texttt{InvCF-McNL}$ holds. Recall that the rules, which are inverse context-free but not monadic, are those preserving length.

4. Observing String-Rewriting Systems

Having determined the accepting power of the relevant classes of string-rewriting systems in terms of McNaughton languages, we now investigate, to what extent this power can be increased by adding an observer and decider in the manner described in the introduction.

For the observer we need a device mapping arbitrarily long strings into just one singular symbol. As in earlier work [5] we use a special variant of finite automata with some feature known from Moore machines or also from subsequential transducers [2]: the set of states is labelled with the symbols of an output alphabet Ψ. Any computation of the automaton produces as output the label of the state it halts in (we are not interested in accepting / not accepting computations and therefore also not interested in the presence of final states); because we find it preferable that the observation of a certain string always lead to a fixed result, we consider here only deterministic and complete automata.

Formalizing this, a *monadic transducer* is a tuple $O = (Z, V, \Psi, z_0, \delta, \sigma)$ with state set Z, input alphabet V, initial state $z_0 \in Z$, and a complete transition function δ as known from conventional finite automata; further there is the output alphabet Ψ and a labelling function $\sigma : Z \mapsto \Psi$. The output of the monadic transducer is the label of the state it stops in. For a string $w \in V^*$ and a monadic transducer O we then write $O(w)$ for this output; for a sequence $\langle w_1, \ldots, w_n \rangle$ of $n \geq 1$ strings over V^* we write $O(w_1, \ldots, w_n)$ for the string $O(w_1) \cdots O(w_n)$. The class of all deterministic monadic transducers will be denoted by $\mathcal{MT}$. For simplicity, in what follows we present only the mappings that the observers define, without giving detailed implementations for them.

As *deciders* we require devices accepting a certain language over the output alphabet Ψ of the corresponding observer as just introduced. For this we do not need any new type of automaton but can rely on conventional finite automata with input alphabet Ψ. The output of a decider D, for a word $w \in \Psi^*$ in input, is denoted by $D(w)$. It consists in a simple `yes` or `no`. The class of all deciders will be denoted by $\mathcal{FA}$, like for standard finite state automata.

Now all the necessary components informally described in the introduction are defined, and we can give the central definition of this article.

Definition 4.1. A rewriting/observer system (in short RO) is a quadruple $\Omega = (\Delta, R, O, D)$; here Δ is the finite input alphabet, R is a string-rewriting system over an alphabet Σ, where $\Delta \subseteq \Sigma$; O, the observer, is a monadic transducer $(Z, \Sigma, \Psi, z_0, \delta, \sigma)$, and D is a decider with input alphabet Ψ.

The language accepted by such a system is the set of all words $w \in \Delta^*$ such that there exists a sequence $s \in R(w)$ such that $D(O(s)) =$ `yes`; formally

$$L(\Omega) := \{w : \exists s[s \in R(w) \wedge D(O(s)) = \texttt{yes}]\}.$$

For a class $\mathcal{R}$ of string-rewriting systems, $\mathcal{O}$ of observers and $\mathcal{D}$ of deciders, we will denote by $\mathcal{RO}(\mathcal{R}, \mathcal{O}, \mathcal{D})$ the class of all languages accepted by ROs consisting of components from the respective classes.

With an introductory example we first want to illustrate the functioning of an RO system. Also this will already demonstrate some techniques that can be used to obtain a general behaviour beyond the power of the individual parts.

Example 4.1. The non semilinear language $\{a^{2^n} : n \geq 1\}$ can be accepted by an RO system $\Omega = (\Delta, R, O, D)$ with $\Delta = \{a\}$ and the string-rewriting system R over the alphabet $\Sigma = \Delta \cup \{A, B\}$ with the inverse context-free rules $\{(aa, A), (AA, a), (A, B), (a, B)\}$.

The observer O is defined by the following mapping.

$$O(w) = \begin{cases} 1 & \text{if } w \in a^+A^+, \\ 2 & \text{if } w \in A^+a^+, \\ 3 & \text{if } w \in a^+, \\ 4 & \text{if } w \in A^+, \\ 5 & \text{if } w = B, \\ 6 & \text{else.} \end{cases}$$

The decider D accepts the language $3((1^*3) \cup (2^*4))^*5$.

This way only evolutions of the system are accepted, where –starting from the left– all a are pairwise replaced by one A, then the other way round etc. In every iteration the length of the string is cut in half. If we arrive in this manner at a single letter, then the original string had a power of 2 as length. The final reduction to B ensures that this is the case.

Already this gives a first hint of the fact that observation can result in a significant increase in power: as stated in Theorem 3.3, the class InvCF-McNL coincides with the context-free languages, while on the other hand $\{a^{2^n} : n \geq 1\}$ is not even semi-linear. Now we provide a first result that states in a more general and precise way the power that is gained by the cooperation of the different components of a RO system. With a rewriting system of sub-regular accepting power as characterized in Theorem 3.2, and with regular observer and decider already computational universality is achieved.

Theorem 4.1. $\mathcal{RO}(\mathtt{CF}, \mathcal{MT}, \mathcal{FA}) = RE$.

Proof:
It should be obvious that any RO system of the specified type can be simulated by a Turing machine. Thus we show only the inclusion from right to left.

For this we construct for any Turing machine M an equivalent RO system Ω with a string-rewriting-system from CF, an observer from $\mathcal{MT}$, and a decider from $\mathcal{FA}$. For this proof we will use off-line Turing Machines with only one combined input/working tape. The set δ of transitions is composed of elements of the form $Q \times \Phi \to Q \times \Phi \times \{+, -\}$, where Q is the set of states, Φ the tape alphabet, and $+$ or $-$ denote a move to the right or left respectively. An input word is accepted, if the Turing machine stops in a state that belongs to the set $F \subset Q$ of final states. Its initial state is $q_0 \in Q$. The special letter $\Box \in \Phi$ denotes an empty tape cell, the letters $\vdash$ and $\dashv$ also from Φ will be used to mark the left and right border respectively.

Now we build up a RO $\Omega = (\Phi, R, O, D)$ simulating M. Here R is a string-rewriting system over the alphabet $\Sigma = \Phi \cup (\Phi \times Q) \cup T$; the set T will be defined further down.

Before starting the simulation, we need to create the necessary working space, which in general exceeds the one occupied by the input word. For this we use two special symbols $\vdash$ and $\dashv$ for space creation on the left and right respectively. For all $x \in \Phi \setminus \{\Box\}$ now P contains the rules $x \to \vdash x$ and $x \to x \dashv$. Then there are the rules $\vdash \to \vdash \Box$ and $\dashv \to \Box \dashv$ to actually create the space.

Thus the initialization of a simulation on the input word aba looks as follows (the amount of space introduced might vary):

$$\begin{array}{rl} aba & \stackrel{1}{\Rightarrow} \\ \vdash aba & \stackrel{1}{\Rightarrow} \\ \vdash aba \dashv & \stackrel{2}{\Rightarrow} \\ \vdash \square\square aba \dashv & \stackrel{3}{\Rightarrow} \\ \vdash \square\square aba\square\square\square \dashv & \stackrel{1}{\Rightarrow} \\ \vdash \square\square \frac{a}{q_0} ba\square\square\square \dashv & \end{array}$$

It is the task of observer and decider to guarantee that things go in this order, and e.g. no $\dashv$ is created in the middle of the word. For this we set

$$O(w) = \begin{cases} A & \text{if } w \in (\Phi \setminus \{\square\})^*, \\ B & \text{if } w \in \vdash (\Phi \setminus \{\square\})^*, \\ C & \text{if } w \in \vdash (\Phi \setminus \{\square\})^* \dashv . \\ D & \text{if } w \in \vdash \square\Phi^* \dashv . \end{cases}$$

Thus a initialization as depicted above results in the observation $ABCD^+$.

All transitions in δ have associated to them an unique label t, and T is the set of all these labels. A letter from the set $\Phi \times Q$ shall indicate that M's head is in the indicated position and the current state is represented in the component from Q. To facilitate any potential configurations the rule set P contains all possible rules of the form $x \to y$, where $x \in \Phi$ and $y \in \Phi \times Q$.

Further, for all transitions $t : (q_1, x) \to (q_2, y, \mu)$ the rules $(x, q_1) \to t$ and $t \to y$ are added to P. We give now an example of how such a transition's simulation works for $\mu = +$. The letters from $\Phi \times Q$ with two components are depicted in the way $\frac{\text{component 1}}{\text{component 2}}$:

$$\frac{x}{q_1} z \stackrel{(x,q_1)\to t}{\Rightarrow} tz \stackrel{z\to(z,q_2)}{\Rightarrow} t\frac{z}{q_2} \stackrel{t\to y}{\Rightarrow} y\frac{z}{q_2}$$

The rest of the word should consist exclusively of letters from Φ with the two border markers. For the simulation of each transition $t : (q_1, x) \to (q_2, y, +)$, the mapping realized by the observer with output alphabet $\{A, B, C, D, I, T_1, T_2, T_f, T_n\}$ does the following:

$$O(w) = \begin{cases} T_1 & \text{if } w \in \vdash \Phi^* \frac{x}{q_1} \Phi^* \dashv \\ & \text{and not } w \in \vdash \Phi^* \frac{x}{q_0} \Phi^* \dashv, \\ T_2 & \text{if } w \in \vdash \Phi^* t \Phi^* \dashv, \\ T_f \text{ or } T_n & \text{if } w \in \vdash \Phi^* t \frac{y}{q_2} \Phi^* \dashv . \end{cases}$$

The configuration with $y\frac{z}{q_2}$ is again of the first type. Transitions moving to the left ($\mu = -$) are treated analogously. In the last clause, T_f is output if q_2 is an accepting state, T_n otherwise. Thus the correct simulation of a transition results in the observation $T_1T_2T_f$ or $T_1T_2T_n$.

Finally, the configurations of type $\vdash \square^+ \frac{x}{q_0} \Phi^* \dashv$ are mapped to the special output symbol I for all $x \in \Phi \setminus \{\square\}$; these cases must be excepted and not mapped to A in the simulation of transitions from

state q_0. This way we ensure that we start the simulation on the first tape symbol by setting the decider to accept the following language:

$$ABCD^+I(T_2(T_f \cup T_n)(T_1 \cup I))^*T_2T_f.$$

It is worth noting that this way no additional workspace can be created during the simulation of the computation. While this would not be a problem in principle, it makes the simulation easier to control.

In all configurations not mentioned here, the output of the observer is E. Since this letter does not appear in the language accepted by the decider, it leads to immediate rejection. Due to the very specialized nature of the observer mapping and the uniqueness of transitions, only correct simulations of accepting computations of M can lead to observations accepted by the decider. □

As can be seen from the proof, the rules with right sides longer than one are needed only in the initalisation phase to create the two border markers and the workspace exceeding that of the input word. Machines not needing such space are called Linear Bounded Automata (LBA) and accept exactly the context-sensitive languages. Thus we see immediately that all context-sensitive languages are in the class $\mathcal{RO}(\mathtt{PA}, \mathcal{MT}, \mathcal{FA})$. On the other hand, simple renaming and deleting can certainly be done by an LBA. Together with the fact that the simulation of observer and decider needs basically no space, this serves us to prove the following statement.

Corollary 4.1. $\mathcal{RO}(\mathtt{PA}, \mathcal{MT}, \mathcal{FA}) = CS$.

Proof:
The only thing that remains to be shown is that the simulation of observer and decider can be done by an LBA in its memory without writing anything on the worktape. For this we simply do not let the decider wait with its work, until the entire observation is made. Rather as soon as the observed letter of the step of the simulated RO is determined, this letter is not written anywhere, but the state of the decider is changed accordingly on-line. The finite number of states of a decider from $\mathcal{FA}$ can, of course be remembered in the LBA's control, as well as the process of observation by a monadic transducer can be simulated there, while moving along the tape without writing anything. □

It has been shown in Theorems 3.1 and 3.2 that both painters and context-free systems are less powerful in terms of McNaughton languages, when they are required to be confluent. As components of ROs, however, their being confluent or not does not affect their power. We will see in the proof of this statement that we can make confluent any such string-rewriting system by adding rules that are never used in computations of the corresponding RO, and then, of course, the language accepted remains the same.

Corollary 4.2. $\mathcal{RO}(\mathtt{con\text{-}PA}, \mathcal{MT}, \mathcal{FA}) = CS$.

Proof:
Let us suppose that we have an arbitrary $\Omega = (\Delta, R, O, D)$ from $\mathcal{RO}(\mathtt{PA}, \mathcal{MT}, \mathcal{FA})$. We will now sketch the construction of an equivalent one using a confluent string-rewriting system. Let R's alphabet be Σ. The new rewriting system will work over $\Sigma \cup \{Z, \square\}$, where Z and $\square$ are new symbols not in Σ. Our new rewriting system will be

$$R' := (R \setminus \{(\ell, \lambda) : \ell \in \Sigma\}) \cup \{(\ell, \square) : (\ell, \lambda) \in R\} \cup \{a \to Z : a \in \Sigma\} \cup \{(\square, Z)\}.$$

In case of painters, strings derived from the same initial string can have at most this string's length, since we eliminate all deleting rules, any word w is reduced to $Z^{|w|}$. This is unique for every w and also for every string reachable from w, and thus the system R' is confluent.

The observer is modified as follows: clearly, $IRR(R)$ is a regular set and can therefore be recognized by a monadic transducer. So we let O work as before, only when a string from $IRR(R)$ is detected, its output letter is marked in some way. Further, for all strings containing Z, also here a special new letter z is output, and the space symbol $\Box$ is ignored, i.e. it is read witthout changing the state. Now every computation of the original Ω can still be carried out, the observation is the same, only the last symbol is marked. Then we can reduce all remaining letters to Z, the observation receives a tail of zs. Clearly, every word from $L(\Omega)$ can still be recognized, and if the decider accepts only observations of the kind described, no other words will be accepted. □

One might suspect now that the same is true for context-free string-rewriting systems and that $\mathcal{RO}(\texttt{con-CF}, \mathcal{MT}, \mathcal{FA}) = RE$ holds. However, the same proof technique can not be applied, since one string can grow to irreducible strings of different lengths. Letterwise reduction to a single symbol would then not lead to identical strings. So an exact characterization of the class $\mathcal{RO}(\texttt{con-CF}, \mathcal{MT}, \mathcal{FA})$ remains an open problem. We only know that it is situated somewhere between the context-sensitive and the recursively ennumerable languages.

From Theorems 3.1 and 3.3 we know that $\texttt{PA-McN} \subset \texttt{InvCF-McN}$, and that the latter class is much bigger. Under observation with monadic tranducers and finite automata, however, they accept exactly the same class of languages.

Corollary 4.3. $\mathcal{RO}(\texttt{InvCF}, \mathcal{MT}, \mathcal{FA}) = \mathcal{RO}(\texttt{PA}, \mathcal{MT}, \mathcal{FA}) = CS$.

Proof:
Since in the proof of Theorem 4.1 all rules are non-deleting, the painter systems of Corollary 4.1 do not need such rules either. Thus all systems employed are also inverse context-free, and consequently $\mathcal{RO}(\texttt{PA}, \mathcal{MT}, \mathcal{FA}) \subseteq \mathcal{RO}(\texttt{InvCF}, \mathcal{MT}, \mathcal{FA})$.

On the other hand, for a system from $\mathcal{RO}(\texttt{InvCF}, \mathcal{MT}, \mathcal{FA})$ we can construct an equivalent one from $\mathcal{RO}(\texttt{PA}, \mathcal{MT}, \mathcal{FA})$ as follows. For this, once again we introduce a new symbol $\Box$ to denote an empty space; in contrast to the proof of Theorem 4.1, however, here it will be interpreted as not having any extension. All rules with left sides longer than one are replaced by a set of rules. For the example $AB \to C$ these are $A \to A'$, $B \to B'$, $A \to \Box$, and $B' \to C$. The observer's task will be to monitor that at any given time there are such new, primed symbols only at one position in the string and only stemming from the samerule.

Then the derivation must be done in the sequence $AB \Rightarrow A'B \Rightarrow A'B' \Rightarrow \Box B' \Rightarrow \Box C$. By now it should be clear that this process can easily be controlled within our architecture. Of course, in a later stage of the derivation, two such letters A and B might be separated by arbitrarily many $\Box$; therefore the observer should treat this symbol as non-existent, i.e. read it without changing its state. Then the resulting system accepts the same language as the original one.

□

This sheds some light on the fact that higher computational power in terms of McNaughton languages does not always imply the same in the context of rewriting/observer systems. The next section will explore some systems, where observation is not so effective in this sense.

5. Less Powerful RO Systems

In the generating case of Grammar/Observer systems, context-free grammars sufficed to obtain all recursively enumerable languages [5]. However, for the only slightly less powerful linear grammars, observation results even in a reduction in power to that of regular languages [10]. In a similar way, one should expect that also different types of string-rewriting systems are differently suited for usage in a rewriting/observer system.

Before looking at any specific variant of RO systems, we can establish some quite general lower bounds for their accepting power. It is hard to formulate lower bounds for every thinkable, pathological string-rewriting system, but our bounds hold at least for almost any reasonable combination of components.

For most RO systems the power will be at least the same as the power in terms of McNaughton languages of the underlying string-rewriting system. For instance, whenever length-increasing rules are allowed, then the strings t_1 and t_2 from the definition of McNaughton languages can simply be introduced by new rules in the very first steps of a reduction of an input string.

The following example will illustrate, how the same can also be done in the slightly more intricate case, where only inverse context-free rules are allowed. The steps of introducing t_1 and t_2 and rewriting the first and last symbol must be compressed into just one step each. This is done at the cost of amplifying the alphabet.

Example 5.1. The system $R = \{(ca, c), (bd, d), (cabd, Y)\}$ with initial strings $t_1 = c$ and $t_2 = d$ accepts the language a^+b^+. We now extend the alphabet by the letters $[ca]$ and $[bd]$ and the new string-rewriting system will be $R' = \{(a, [ca]), (b, [bd]), ([ca]a, [ca]), (b[bd], [bd]), ([ca][bd], Y)\}$

$$O(w) = \begin{cases} 1 & \text{if } w \in a^+b^+, \\ 2 & \text{if } w \in [ca]a^*b^+, \\ 3 & \text{if } w \in [ca]a^*b^*[bd], \\ 4 & \text{if } w = Y, \\ 5 & \text{else.} \end{cases}$$

The decider accepts the language 123^+4, and the entire system accepts a^+b^+ just like the original string-rewriting system. In the way demonstrated here any finite strings t_1 and t_2 can be encoded in the outermost symbol of an input string.

Another lower bound for the power of RO systems stems from the power already present in the observer. Thus any regular language can always be accepted by simply ignoring the underlying system and letting only the observer make the decision of acceptance or not in the very first step, when it reads the unaltered input string.

Example 5.2. Almost regardless of the underlying string-rewriting system, for a regular language L over an appropriate alphabet we can construct a RO system accepting L by using the observer

$$O(w) = \begin{cases} 1 & \text{if } w \in L, \\ 2 & \text{if } w \notin L. \end{cases}$$

Now the decider simply has to accept the language $1\{1,2\}^*$. Thus the observer checks in the first step, whether the input word belongs to L, everything else that happens after this is irrelevant for the acceptance, and the language accepted will be exactly L, if only the rewriting process stops on every word from L in at least one reduction.

Now we come to investigate a type of system, where really no increase in accepting power is observed, when we move from McNaughton languages to rewriting/observer systems. These are the *non-increasing* string-rewriting systems; a rewrite rule (ℓ, r) is non-increasing, if $|\ell| \geq |r|$. The class of non-increasing string-rewriting systems we denote by non-in. We recall a result of Beaudry et al. concerning the McNaughton languages accepted by these systems.

Theorem 5.1. ([1])
non-in-McNL $= CSL$.

To characterize the class $\mathcal{RO}(\texttt{non-in}, \mathcal{MT}, \mathcal{FA})$, we set in in relation to $\mathcal{RO}(\texttt{PA}, \mathcal{MT}, \mathcal{FA})$, which turns out to be rather straight-forward by techniques as used already above in the proof of Corollary 4.3.

Theorem 5.2. $\mathcal{RO}(\texttt{non-in}, \mathcal{MT}, \mathcal{FA}) = \mathcal{RO}(\texttt{PA}, \mathcal{MT}, \mathcal{FA})$.

Proof:
Every painter rule does not increase a string's length. Therefore we have PA $\subseteq$ non-in, which suffices to prove one direction of the equality. It remains to show that any rewriting/observer system with a non-increasing rewriting system can be simulated by one with a painter.

We will only sketch the construction, because it is essentially the same as in the proof of Corollary 4.3. We take the example of a non-increasing rewrite rule $ABC \to DE$. It can be simulated by the sequence

$$ABC \Rightarrow A'BC \Rightarrow A'B'C \Rightarrow A'B'C' \Rightarrow \square B'C' \Rightarrow \square DC' \Rightarrow \square DE,$$

where $\square$ once again denotes a deleted space and is ignored by the observer. Only painter rules are used and observer and decider can check that the rewriting rules are applied following the correct order. Since all original rules are non-increasing, never two or more symbols need to be created from one, with painter rules the entire original non-increasing RO can be simulated. $\square$

Combining Theorems 5.1 and 5.2 we obtain the result promised above, demonstrating that in some cases observation does not increase the accepting power compared to the McNaughton case.

Corollary 5.1. non-in-McNL $= \mathcal{RO}(\texttt{non-in}, \mathcal{MT}, \mathcal{FA}) = CSL$.

This section has shown that context-sensitivity in the rules does not really help much; it can be simulated by simple renaming of individual symbols and the control provided by the observer/decider complex. Essential features in simulating machines were rather the possibility to loop infinitely and to expand space. The first takes us beyond time restrictions to obtain in general all context-sensitive languages, the second lets also the space restriction fall and results in universality.

6. Perspectives

As we have introduced a completely new architecture, many interesting questions remain open. For one thing, investigation of McNaughton languages of types of string-rewriting systems used as underlying systems should be continued to obtain a richer arsenal of candidates to compare. One question is obviously left open in Theorem 3.3, namely a characterization of the class `con-InvCF-McNL`. Special attention might be paid here to all types of noetherian systems — these admit only derivations of a bounded length and might be more realistic in some contexts, for example when simulating chemical reactions without added energy.

Still more interesting seems the exploration of the rewriting/observer architecture itself. The observers, though regular, might still be too powerful for any application. In simulating the observation of a bio-chemical process, a sensible restriction seems to be allowing only the check for the absence or presence of a certain letter; this can often be done rather quickly, while actually quantifying the amount of a substance in a solution is a major task.

Another restriction would be to bound the time available to the observer. After all, it must be ready to start again, when the rewriting system has performed its next step. To simulate this, observers could be restricted to reading only a window of a fixed length of the string in every step. Here either a prefix or an arbitrary factor could be considered.

On the other hand, still inspired from biological motivations, we should consider that an observer may be not able to catch every single change of the observed string. In this case one could consider only a scattered subword of the entire observed evolution. These are only a few ideas coming to mind for following to more depth and possibly to more realistic modelling the lines of research presented in this article.

References

[1] M. BEAUDRY, M. HOLZER, G. NIEMANN and F. OTTO: *McNaughton families of languages*. In: Theoretical Computer Science 290, 2003, pp. 1581–1628.

[2] J. BERSTEL: *Transductions and Context-Free Languages*. Teubner, Stuttgart, 1979.

[3] R. BOOK and F. OTTO: *String-Rewriting Systems*. Springer, Berlin, 1988.

[4] M. CAVALIERE and P. LEUPOLD: *Evolution and Observation – A new way to look at Membrane Systems.* In: Membrane Computing, International Workshop, WMC 2003. Lecture Notes in Computer Science 2933, Springer-Verlag, Berlin, 2004, pp. 70–87.

[5] M. CAVALIERE and P. LEUPOLD: *Evolution and Observation — A Non-Standard Way to Generate Formal Languages*. In: Theoretical Computer Science 321, 2004, pp. 233-248.

[6] M. CAVALIERE and P. LEUPOLD: *Evolution and Observation — A Non-Standard Way to Accept Formal Languages*. In: MCU 2004, Lecture Notes in Computer Science 3354, Springer-Verlag, Berlin, 2005, pp. 152–162.

[7] M.A. HARRISON: *Introduction to Formal Language Theory*. Reading, Mass., 1978.

[8] D. HOFBAUER and J. WALDMANN: *Deleting String-Rewriting Systems preserve regularity*. In: Theoretical Computer Science 327, 2004, pp. 301-317.

[9] L. ILIE and A. SALOMAA: *2-Testability and Relabelings Produce Everything*. In: Journal of Computer and System Sciences 56(3), 1998, pp. 253-262.

[10] P. LEUPOLD: *Non-Universal Grammar/Observer Systems*. Manuscript.

[11] R. MCNAUGHTON, P. NARENDRAN and F. OTTO: *Church-Rosser Thue systems and formal languages*. In: Journal of the ACM 35, 1988, pp. 324–344.

[12] GH. PĂUN, G. ROZENBERG and A. SALOMAA: *DNA Computing – New Computing Paradigms*. Springer Verlag, Berlin, 1998.

[13] A. SALOMAA : *Formal Languages*. Academic Press, Orlando, 1973.

[14] J.R. WOINOWSKI: *The Context-Splittable Normal Form For Church Rosser Language Systems*. In: Information and Control 183, 2003, pp. 245-274.