Internal Report 2012-2013-12 July 2013

Universiteit Leiden

Opleiding Informatica

Guided Rewriting in Families of Languages

Jesper van Engelen

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Bachelor thesis by J. van Engelen
Supervised by dr. H.J. Hoogeboom July 2013

GUIDED REWRITING IN FAMILIES OF LANGUAGES

J. VAN ENGELEN

Abstract. Guided rewriting is an operation on formal languages, motivated by RNA editing, replacing substrings by certain guide strings, provided that the strings are letter by letter equivalent. De Vink et al (MeCBIC, 2012) show that regular sets are closed under guided rewriting with finite guide sets. We extend this to regular guide sets, answering an open problem from that paper. Our approach is more general, so that it also applies to "cones", families closed under regular transductions, including context-free languages.

1. Introduction

Within an RNA molecule, nucleotide sequences can be changed by operations called insertion and deletion which add and delete the nucleobase uracil, respectively. This operation can be abstracted to the more general concept of insertion and deletion within strings in formal languages, based on certain rules.

The article *Combining Insertion and Deletion in RNA-editing Preserves Regularity* [1] by De Vink et al covers an operation on regular languages called guided rewriting. The operation maps a string to another string based on a set of guides, which allow certain substrings to be replaced by a guide string of the same length as the substring being replaced. The operation of guided rewriting on a language yields a language containg all elements that can be obtained by executing a rewriting on a string from the original language.

It is proven in [1] that regular languages are closed under the operation of guided rewriting with a finite set of guides. The concluding remarks of this article pose the question whether context-free languages are closed under guided rewriting, as well as the question whether regular languages are closed under guided rewriting with a regular set of guides.

We combine these two questions to form our research question: *which families of languages are closed under guided rewriting with a regular set of guides?* The relatively simple approach taken in [1] to obtain a finite representation of a rewriting can not be applied to regular guides. We find a general solution to the problems posed by separating the operation from the input language and constructing a finite state transducer mapping a language to its rewritten language for a regular set of guides, thereby proving that families of languages that are closed under homomorphism, inverse homomorphism and intersection with regular language, so-called cones, are closed under guided rewriting with a regular set of guides.

Date: Wednesday 10th July, 2013.

2. Guided rewriting

Using guided rewriting, a string over Σ can be rewritten to a string of the same length, based on a set of guides. A substring of a string can be replaced by a guide, which is also a string over Σ, but only if all characters in the original substring are adjustable to the character in the guide at the corresponding position. Adjustability between characters is governed by equivalence: only equivalent characters can be adjusted to one another. This process of replacing substrings of a string can be repeated with different guides at different positions for an arbitrary number of times, as long as the substring being replaced is adjustable to the guide.

Equivalence between characters is defined in so-called equivalence classes. For any alphabet, we define equivalence classes as sets of characters that are equivalent to one another (and to themselves, the equivalence relation is reflexive). It follows that the equivalence relation is also both symmetric and transitive. For $\alpha \in \Sigma$, equivalent to $\beta \in \Sigma$, we write $\alpha \sim \beta$. Extending this notion to strings, we say a string is adjustable to another string if for every position in both strings, the character at that position is adjustable to the character at the same position in the other string. For $x \in \Sigma^*$, adjustable to $x' \in \Sigma^*$, we write $x \sim x'$.

Figure 1 shows a very simple rewriting for a string *acbdc*, which is rewritten to *dbeab* using five unique guides and equivalence classes such that $b \sim c \sim e$ and $d \sim a$. Using this image, we explain the notion of characters being adjustable to other characters, and the subsequent notion of strings being adjustable to other strings.

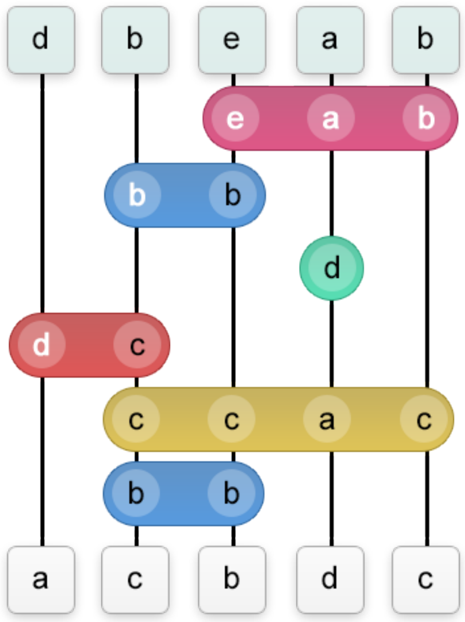

Figure 1. A rewriting of string *acbdc* to *dbeab*.

The first (lowest) blue guide bb in Figure 1 rewrites the input string $acbdc$ to $a\mathbf{bb}dc$, emphasizing the part of the string being rewritten. This is allowed as $b \sim c$ and $b \sim b$. Directly after that, the yellow guide $ccac$ is applied, rewriting $abbdc$ to $a\mathbf{ccac}$. This is allowed as $b \sim c$, $d \sim a$ and $c \sim c$. Extending this to strings, $cb \sim bb$ and $bbdc \sim ccac$.

For proving for some families of languages that they are closed under the operation of guided rewriting with a regular set of guides, we will need to represent rewritings of strings or parts of strings in some finite way. To do so, we will need to adapt some definitions from [1], ultimately yielding a way to prove these closure properties.

2.1. Guides.

We refer to guides in a similar way as [1]. A guide g from the set of guides G is a string over Σ. As our set of guides G is regular, it is accepted by some finite state automaton (FSA) $\mathcal{A}$. Using this, we can refer to a guide as a sequence of transition input characters in $\mathcal{A}$, representing a string. Thus, instead of using a finite set of guides $g_1, \ldots, g_n \in G$ as is done in [1], we use guides $G = L(\mathcal{A})$.

In following references to this finite state automaton $\mathcal{A}$ accepting our set of guides, we assume that $\mathcal{A}$ is in a normal form where it has an initial state with no incoming transitions, and a single accepting state with no outgoing transitions. By standard construction, any FSA $\mathcal{A}'$ that is not in this normal form can be converted to a nondeterministic finite automaton $\mathcal{A}$ that is in this normal form such that $L(\mathcal{A}) = L(\mathcal{A}')$. We represent $\mathcal{A}$ as a 5-tuple $(Q, \Sigma, \delta, s_0, F)$ with:

- Q a finite set of states
- Σ the input alphabet
- $\delta \subseteq (Q \times \Sigma \times Q)$ the transitions
- s_0 the initial state
- $F = \{s_f\}$ the set of accepting states

For any formal language L, we define the language L_G as the set of strings that can be derived from a string in L by guided rewriting with a regular set of guides. We say a string $x \in \Sigma^*$ is rewriteable to $x' \in \Sigma^*$, if there is some guide $g \in G$ such that $x = uvw$ with $u, v, w \in \Sigma^*$, $x' = ugw$ and $v \sim g$. We write $x \Rightarrow x'$.

Definition 1. *Let L be a language over Σ with a set of guides G. We define the rewritten language of L as*

$$L_G = \{v \in \Sigma^* \mid u \Rightarrow^* v \text{ for some } u \in L\}.$$

Relating this definition to Figure 1 with $acbdc \in L$ and $G = \{bb, ccac, dc, d, eab\}$ we find that the result of this rewriting, $dbeab$, is in L_G, by the rewriting $acbdc \Rightarrow a\mathbf{bb}dc \Rightarrow a\mathbf{ccac} \Rightarrow \mathbf{dc}cac \Rightarrow dcc\mathbf{d}c \Rightarrow d\mathbf{bb}dc \Rightarrow db\mathbf{eab}$.

We distinguish between guided rewriting with a finite set of guides, simply *guided rewriting*, and guided rewriting with a regular set of guides, from here on *regular guided rewriting*.

3. Closure properties of regular guided rewriting

In studying the closure properties of regular guided rewriting, we rely heavily on the use of finite state transducers. A finite state transducer (FST) is a finite state automaton which has two tapes: one input tape and one output tape. It generates strings with output symbols by mapping an input string to an output string, instead of simply accepting strings, but otherwise behaves a lot like a nondeterministic finite automaton. A finite state transducer is a 6-tuple

$$T = (Q, \Sigma, \Gamma, I, F, \delta)$$

Where Q is the set of states, Σ is the input alphabet, Γ is the output alphabet, I is the set of input states, F is the set of final (accepting) states and δ is the set of transitions. A transition $t \in \delta$ is defined as a 4-tuple (p, a, b, q) where $p, q \in Q$ and $a \in (\Sigma \cup \{\epsilon\}), b \in (\Gamma \cup \{\epsilon\})$ denoting a transition from state p to state q with input symbol a and output symbol b.

To prove that some families of languages are closed under regular guided rewriting, we will provide a method for constructing a finite state transducer mapping L to L_G for any $L \subseteq \Sigma^*$ with given Σ and a regular set of guides G, based on the finite state automaton $\mathcal{A}$ accepting G. If we succeed in doing this, we have proven that any family of languages closed under finite state transductions is also closed under regular guided rewriting.

3.1. Cones.

In the context of formal languages, a cone is a family of languages that is closed under homomorphism, inverse homomorphism and intersection with regular language [3, p. 201].

Definition 2. *A family of languages $\mathcal{L}$ is a cone if and only if the following properties hold for $\mathcal{L}$:*

- *$\mathcal{L}$ is closed under homomorphism (h)*
- *$\mathcal{L}$ is closed under inverse homomorphism (h^{-1})*
- *$\mathcal{L}$ is closed under intersection with regular language ($\cap Reg$)*

The families of regular (REG), context-free (CF) and recursively enumerable (RE) languages meet these criteria.

By Theorem 3.8 from [3], commonly known as *Nivat's Theorem*, we know that all cones are closed under finite state transductions. Furthermore, as all cone operations (homomorphism, inverse homomorphism and intersection with regular language) can be implemented using a finite state transducer, and finite state transductions are closed under composition, all families of languages that are closed under finite state transductions are cones.

Thus, if a family of languages is closed under the operation of finite state transductions, it is a cone, and vice versa.

We proceed to state our main theorem.

Theorem 1. *Every cone is closed under regular guided rewriting.*

In the following sections, we will work towards a finite representation of rewritings to be used in our finite state transducer, ultimately using this finite representation to prove Theorem 1.

4. Slices

To construct a finite state transducer that rewrites strings character-by-character, we will need to move away from the horizontal perspective (per guide) we have taken on rewriting so far, and look at rewritings from a vertical perspective (per position in the input string). Considering our previous example of a rewriting in Figure 1, we look at the sequences of characters along the vertical lines, instead of looking at the guides.

For this vertically-oriented representation, we introduce the notion of slices, closely related to the notion of slices in [1].

A slice denotes a sequence of characters, each originating from a certain guide, to which one character of the input string at a certain position is rewritten. As our set of guides G is not necessarily finite, we can no longer rely on the notion of guide-offset pairs as is done in [1]. Instead, we will express the current state in the entire set of guides (which a guide-offset pair essentially is as well) by a transition (from one state to another state under an input character) from the FSA $\mathcal{A}$ accepting G.

Definition 3. *Let $\beta \in \Sigma$. A sequence $s\ell$ of transitions (p, α, q) is called a* slice *for β and $\mathcal{A}$ if it holds for all $(p, \alpha, q) \in s\ell$ that:*

- $(p, \alpha, q) \in \delta$
- $\beta \sim \alpha$

A slice $s\ell$ is called a slice for a string $u \in \Sigma^*$ at position *n with $1 \leq n \leq \#u$ if it is a slice for $u[n]$.*

A graphical representation of such a slice is shown in Figure 2. As you can see, we have inset state 1 from the left and state 6 from the right. These inset states denote the initial state (state 1) and the accepting state (state 6) in $\mathcal{A}$, to emphasize the special role the initial state and the accepting state play in the notion of slices, which is clarified later.

As you may notice, we are not using a finite index set (which was done in [1]), but instead denoting a slice as a sequence, which is by definition ordered.

Following this change in the notion of slices, we introduce the notion of *matching slice sequences*, corresponding with the original definition of *slice sequences* from [1]. Using regular guides instead of a finite set of guides, we have to adapt our way of viewing adjacency between slices. Instead of testing whether every element of a slice has a corresponding element in the previous slice for the same guide at the previous position, we only check whether the source state of each element in a slice has a corresponding target state in the preceding slice and whether the target state of each element in a slice has a corresponding source state in the succeeding slice.

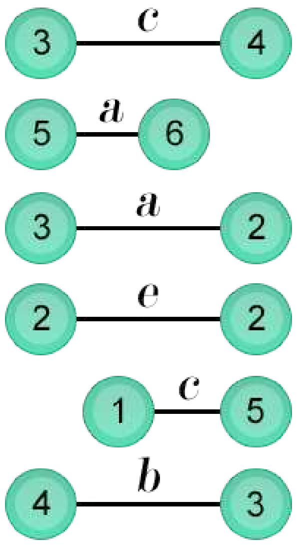

Figure 2. A slice in regular guided rewriting.

However, a source state in a slice does not need a corresponding target state in the previous slice if it is an initial state. Conversely, a target state in a slice does not need a corresponding source state in the next slice if it is an accepting state. Thus, we must check that all non-initial source states from a slice have a matching target state in the preceeding slice, and that all non-accepting target states from a slice have a corresponding source state in the succeeding slice. This is why it is important that $\mathcal{A}$ is in the normal form introduced in Section 2.1: we must be able to differentiate between the initial and accepting state and the other states, making sure that the initial state has no incoming transitions and the accepting state has no outgoing transitions.

We define the left and right parts of a slice, denoting the sequence of source states that are not initial states and the the sequence of target states that are not accepting states in all elements of the slice, respectively.

Definition 4. *Let $s\ell$ be a slice with elements $(p_i, \alpha_i, q_i)_{i=1}^{n}$. Then we define $\mathit{left}(s\ell)$ as the sequence of states p_i for $1 \leq i \leq n$ for which $p_i \neq s_0$ in the same order as the elements of $s\ell$. We define $\mathit{right}(s\ell)$ as the sequence of states q_i for $1 \leq i \leq n$ for which $q_i \neq s_f$ in the same order as the elements of $s\ell$.*

Extending this definition, for a sequence of slices $\sigma = (s\ell_i)_{i=1}^{n}$, we define $\mathit{left}(\sigma) = \mathit{left}(s\ell_1)$ and $\mathit{right}(\sigma) = \mathit{right}(s\ell_n)$.

A graphical representation of $\mathit{left}(s\ell)$ and $\mathit{right}(s\ell)$ is shown in Figure 3.

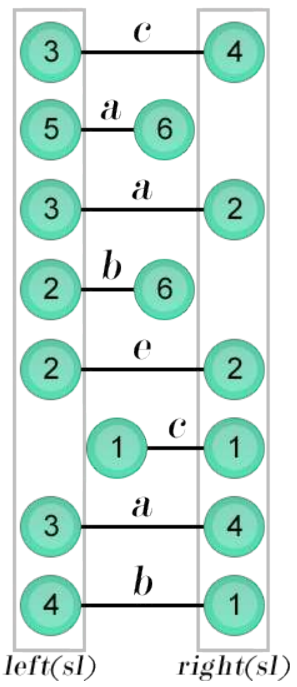

FIGURE 3. Graphical representation of the *left* and *right* parts of a slice.

Using these definitions, we can easily construct our definition for a matching slice sequence, which is a slice sequence in which all adjacent slices fit, but for which it is not necessarily true that it starts with only initial states and ends with only accepting states.

Definition 5. *A sequence $\sigma = (s\ell_n)_{n=1}^{\#u}$ of slices is called a matching slice sequence for a string u if the following holds:*

- *$s\ell_n$ is a slice for u at position n, for $n = 1, \ldots, \#u$*
- *For every $1 \leq n < \#u$, $right(s\ell_n) = left(s\ell_{n+1})$*

Following these definitions, we define a *successful matching slice sequence* σ to be a matching slice sequence for which it holds that $left(\sigma) = \epsilon$ and $right(\sigma) = \epsilon$, where ϵ denotes the empty sequence. It follows that any succesful matching slice sequence represents a rewriting on a string, so the horizontal and vertical perspectives coincide. In classical formal language theory, this concept of matching slice sequences coincides with the notion of crossing sequences in two-way devices (see, e.g., [2]).

The character at the topmost transition of the slice, $yield(s\ell)$, is defined in a similar way as in [1]. For a slice $s\ell$ for β with elements $(p_i, \alpha_i, q_i)_{i=1}^{n}$, we define $yield(s\ell) = \alpha_n$ for $s\ell \neq \emptyset$. In case $s\ell = \emptyset$, we define $yield(s\ell) = \beta$.

We define the yield of an entire slice sequence $\sigma = (s\ell_n)_{n=1}^{\#u}$ for a string u as $v = v_1 \dots v_{\#u}$ with $v_i = \mathit{yield}(s\ell_i)$ for every $1 \leq i \leq \#u$. Thus, $\mathit{yield}(\sigma)$ is the total rewritten string of u for σ.

5. Reduced state sequences

To construct a finite state transducer mapping L to L_G, we need to find a way to express slices inside our finite state transducer. In [1], as the set of guides is finite, it is possible to simply remove repeating guides at a certain position to obtain a finite set of slices. However, as our set of guides is regular, the number of possible slices without repeating guides is potentially infinite. This is the result of the absence of a bound on the length of guides (as our set of guides can be infinite), removing the guarantee that repeating guides must occur at some point. We show this by an example with guides $G = ba^*c$ and $a \sim b$ the only equivalence. Then for every string $a^n c$ from input language $L = a^*c$, we can generate $b^n c$ by rewriting. However, to do so, we need at least n unique guides ending at the last position in the string (the position of c), yielding a slice with at least n elements for the last character. Thus, removing repeating guides will never result in the last slice having fewer than n elements, showing that it can occur that we need an infinite set of slices to represent a rewriting with a slice sequence.

Knowing that a *transition* in our finite state transducer will have to represent the rewriting to a certain character and should thereby be derived from a certain sequence of transitions, we should obtain the *states* in our finite state transducer from sequences of states in our FSA. We argue that nonrepeating sequences of states with corresponding transitions capture the essence of a rewriting (Lemma 1 and Definition 7).

Note that we can obtain all possible state sequences for any slice sequence by using $\mathit{left}(s\ell)$ and $\mathit{right}(s\ell)$. However, as we want to construct a finite state transducer, our set of states needs to be finite, which our set of possible state sequences, in all but one case (when there are no rewrites possible, i.e. the only possible slice is the empty slice), is not. So, we need to find a way to obtain a finite set of state sequences from our set of all possible state sequences.

To do so, we introduce an operation on state sequences called *reduction*. By reducing a state sequence, we remove all lower occurences of states occuring more than once, leaving a sequence of states in which only the top occurence of a state is "visible". Thus, a reduced state sequence is a sequence of states with no repetitions.

Definition 6. *Let X be a sequence $(x_1, \dots, x_n)$. We use the notation $\mathit{del}_x(X)$ to remove all occurences of x from X. We define the operation $\mathit{red}(X)$ to be:*

- $\mathit{red}(X) = \mathit{red}(\mathit{del}_{x_n}(X)) \cdot x_n$ *for* $X \neq \emptyset$
- $\mathit{red}(X) = \epsilon$ *for* $X = \emptyset$

5.1. **Relations between reduced state sequences.** As stated before, the transitions in our finite state transducer need to represent the rewriting of a single character to another character, and should therefore be derived from a sequence of transitions rewriting a single character. We define an operation between reduced state sequences that indicates whether a slice for a certain character exists which has one of the state sequences as its left part, and the other as its right part.

Definition 7. *Let r and r' be two reduced state sequences. Then $r \mapsto_\alpha r'$ if a slice $s\ell$ for some β exists such that the following holds:*

- $\mathit{red}(\mathit{left}(s\ell)) = r$
- $\mathit{red}(\mathit{right}(s\ell)) = r'$
- $\mathit{yield}(s\ell) = \alpha$

We say $r \mapsto r'$ if any α exists such that $r \mapsto_\alpha r'$.

It follows from the definition above that any matching slice sequence can be expressed using state sequences and the $\mapsto_\alpha$-relation. However, it is not that obvious that any sequence of state sequences $R = \{r_0, \ldots, r_n\}$ for which $r_{i-1} \mapsto_{\alpha_i} r_i$ for all $1 \leq i \leq n$ has a corresponding slice sequence with the same yield (i.e. a yield equal to $\alpha_1 \ldots \alpha_n$). After explaining the methods, operations and relations we have introduced in an example in the following section, we will prove this property of these sequences of state sequences in Section 7.

6. Regular guided rewriting example

Before proceeding in proving Theorem 1, we will illustrate the concept of regular guided rewriting and its related operations and relations with an example of a rewriting.

Given alphabet $\Sigma = \{a, b, c, d, e\}$. Let $\mathcal{A}$ be the nondeterministic finite state automaton in the normal form described in Section 4 as depicted in Figure 4, accepting the regular set of guides G. For our equivalence classes, we use $a \sim c \sim e$ and $b \sim d$.

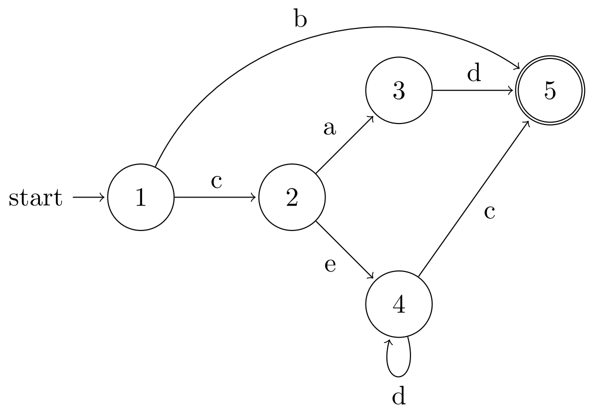

Figure 4. Finite state automaton $\mathcal{A}$ accepting G.

As is clear from Figure 4, $G = L(\mathcal{A})$ can be expressed using the regular expression $b \mid c(ad|ed^*c)$. We will continue to explain the notion of rewriting, slices, slice sequences, the *left*, *right* and *red* operators, and the $\mapsto_\alpha$ relation.

To do so, we will use a possible rewriting on the string $u = aaedbacd$. A graphical representation of this rewriting is given in Figure 5. In this graphical representation, the initial state and accepting state are grayed out and inset to emphasize their special role in the rewriting, as was done by using an inset in Figure 2.

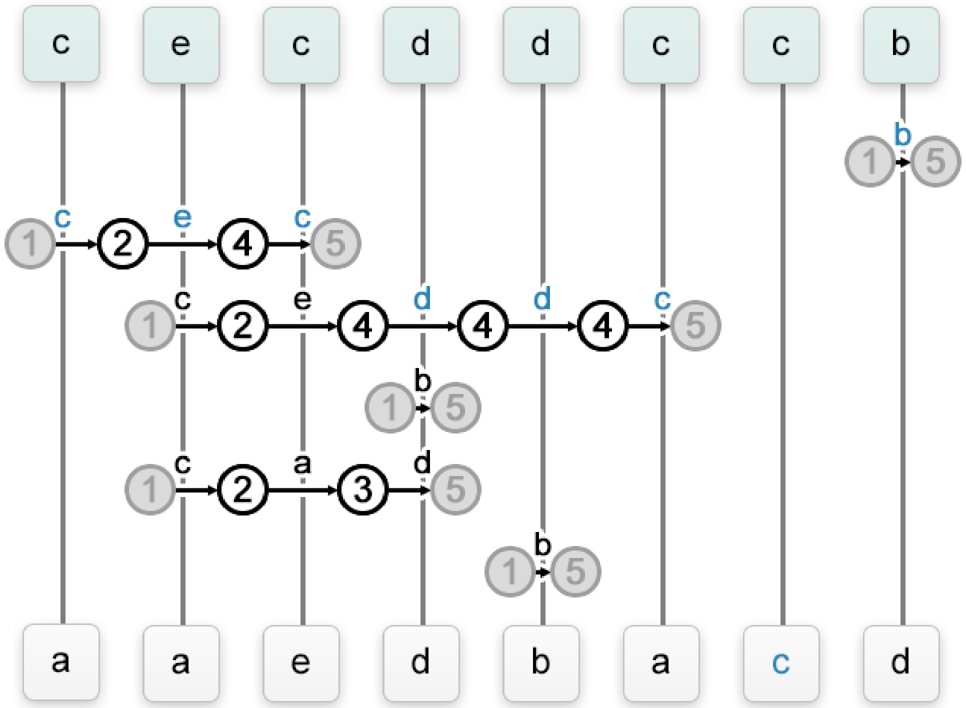

FIGURE 5. A possible rewriting for $u = aaedbacd$.

The bottom row of characters forms the input string u, while the top row of characters forms the result of the rewriting, i.e. the yield. Each horizontal sequence of states and transitions represents a rewriting of the string at that position in the string. As you may notice, each of these sequences is a path in $\mathcal{A}$ from 1 to 5. For example, for the second sequence, looking from the bottom up, is the path $1 \xrightarrow{c} 2 \xrightarrow{a} 3 \xrightarrow{d} 5$, rewriting the input string $a\mathbf{aed}bacd$ to $a\mathbf{cad}bacd$ (emphasizing the substring being rewritten).

Shifting to the notion of slices and slice sequences, we look at the diagram from a vertical perspective. Instead of approaching the rewriting by looking at paths in $\mathcal{A}$, we view a rewriting as a sequence of transition sequences (i.e. slices). As slices consist of transitions, every state in our diagram that is not the initial state (1) or the accepting state (5) is part of two slices. For example, state 2 is part of both the first slice (in the transition $1 \xrightarrow{c} 2$, where it is the target state) and the second slice (in the transition $2 \xrightarrow{e} 4$, where it is the source state).

As we did in our first representation of an example slice in Figure 2, we inset the initial state (1) and the accepting state (5). This is related to the *left* and *right* operations on a slice, in which the initial and accepting states are left out.

We view a more detailed representation of the third slice, which has e as its input character and yields c. This slice, $s\ell_3$, is depicted in Figure 6.

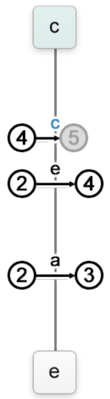

FIGURE 6. Slice $s\ell_3$ from the rewriting represented in 5.

As is clear from Figure 6, $s\ell_3$ consists of three transitions. More precisely, $s\ell_3 = ((2, a, 3), (2, e, 4), (4, c, 5))$. Using this slice, we can explain the notion of the operations *left* and *right* on a slice. By Definition 4, $\mathit{left}(s\ell_3)$ consists of all source states in $s\ell_3$ that are not the initial state. Correspondingly, $\mathit{right}(s\ell_3)$ consists of all target states in $s\ell_3$ that are not the accepting state. By this definition, both $\mathit{left}(s\ell_3)$ and $\mathit{right}(s\ell_3)$ must be in the same order as the transitions of $s\ell_3$.

As the source states of $s\ell_3$ do not contain the initial state, $\mathit{left}(s\ell)$ is simply $(2, 2, 4)$. The right part, on the other hand, contains the accepting state. Thus, $\mathit{right}(s\ell_3) = (3, 4)$.

We move on to the reduction operation, which removes all lower occurences of states in a state sequence. Considering the state sequence $\mathit{left}(s\ell) = (2, 2, 4)$, we notice that state 2 occurs twice. Thus, by Definition 6, $\mathit{left}(s\ell_3)$ does not contain the lowest (first) occurence of 2, keeping the other elements in the same order, yielding $\mathit{red}(\mathit{left}(s\ell_3)) = (2, 4)$. As the right part of $s\ell_3$ contains no duplicate elements, $\mathit{red}(\mathit{right}(s\ell_3)) = \mathit{right}(s\ell_3) = (3, 4)$.

Now that we have these two reduced state sequences, $r = \mathit{red}(\mathit{left}(s\ell_3))$ and $r' = \mathit{red}(\mathit{right}(s\ell_3))$, the $\mapsto_\alpha$-relation becomes clear. This relation, from Definition 7, simply says that there is a slice that has r and r' as its reduced left and right parts, respectively, yielding α. It is obvious that $(2, 4) \mapsto_c (3, 4)$, as we can find a slice for which these properties hold, namely $s\ell_3$. In this way, we can represent an entire slice sequence.

Now that we have explained our method of representing a matching slice sequence as a sequence of reduced state sequences with transitions, we move on to prove that any such sequence has a corresponding matching slice sequence.

7. Converting sequences of state sequences to slice sequences

To use reduced state sequences and the $\mapsto_\alpha$ relation in our finite state transducer, we need to prove that for any sequence of reduced state sequences for which a transition is possible between all adjacent reduced state sequences, there is a corresponding matching slice sequence that yields the same string.

Lemma 1. *Let R be a sequence of reduced state sequences $R = \{r_0, \ldots, r_n\}$ for which $r_{i-1} \mapsto_{\alpha_i} r_i$ for all $1 \leq i \leq n$. Then there exists a matching slice sequence σ for which it holds that*

$$\mathit{yield}(\sigma) = \alpha_1 \ldots \alpha_n$$

We prove this lemma by induction on the natural number n. Firstly, we show that two slice sequences σ and σ' can be combined to a single slice sequence with properties matching the properties of the two initial slice sequences, if $\mathit{red}(\mathit{right}(\sigma)) = \mathit{red}(\mathit{left}(\sigma'))$. We do so by providing a mechanism for constructing this new slice sequence, and showing that it has the same yield.

Lemma 2. *Let σ and σ' be two matching slice sequences on u and u', respectively, such that $\mathit{red}(\mathit{right}(\sigma)) = \mathit{red}(\mathit{left}(\sigma'))$. Then there exists a slice sequence ϕ for $u \cdot u'$ such that*

- $\mathit{red}(\mathit{left}(\phi)) = \mathit{red}(\mathit{left}(\sigma))$
- $\mathit{red}(\mathit{right}(\phi)) = \mathit{red}(\mathit{right}(\sigma'))$
- $\mathit{yield}(\phi) = \mathit{yield}(\sigma) \cdot \mathit{yield}(\sigma')$

To prove this lemma, we provide a method for constructing ϕ based on σ and σ'. In this method, we rely on a visual representation of a slice sequence to obtain an easy method of copying sequences of transitions. The partial diagrams for the slice sequences used are represented in Figure 7.

Given σ and σ', the two matching slice sequences from Lemma 2, we can check the states of $\mathit{right}(\sigma)$ against the states of $\mathit{left}(\sigma')$, one by one. As the reduced right part of the first slice sequence is equal to the reduced left part of the second slice sequence, we know that if the current state being checked for σ is not equal to the current state being checked for σ', there is a state that has already been checked in σ' which is equal to the current state in σ, or that there is a state that has already been checked in σ which is equal to the current state in σ'. Without changing the yield of any of the two partial slice sequences, we can copy the sequences of transitions corresponding to this state that occured previously, thereby "fixing" the unmatching pair of states.

We formalize this approach.

Proof for lemma 2. Given the matching slice sequences σ and σ', let $R = \mathit{right}(\sigma) = \{r_1, \ldots, r_m\}$ and $R' = \mathit{left}(\sigma') = \{r'_1, \ldots, r'_n\}$. Our approach is as follows: Let i and

j be the indexes of the states we are currently evaluating, starting with $i = m, j = n$. If $r_i = r'_j$, move to the next pair of states by decrementing i and j by 1, and starting evaluation again. If, however, $r_i \neq r'_j$, then either r_i occurs in $r'_{j+1} \dots r'_n$ or r'_j occurs in $r_{i+1} \dots r_m$, or both. Assuming $r_i = r'_k$ occurs in $r'_{j+1} \dots r'_n$, consider the sequence of transitions and states corresponding to r'_k in the diagram for σ', and copy this sequence above r'_j in the diagram. Otherwise, assuming $r'_j = r_k$ occurs in $r_{i+1} \dots r_m$, consider the sequence of transitions and states corresponding to r_k in the diagram for σ, and copy this sequence above r_i in the diagram.

If both $i = 0$ and $j = 0$ then $R = R'$, and we obtain ϕ by concatenating the two constructed matching slice sequences. If this is not the case, we proceed by decrementing both i and j by 1, unless a sequence from σ was copied, in which case we decrement only j, or a sequence from σ' was copied, in which case we decrement only i.

By performing the copying steps, we "fix" an unmatching pair by inserting a sequence of transitions in one of the diagrams. By doing so, we do not violate the properties from Lemma 2. Firstly, the yield remains the same. As we are copying a sequence that has already occured above the current index, the copied sequence will never be "visible" and will thus not affect the yield. Secondly, the reduced left and right parts of ϕ will match the reduced left and right parts of σ and σ', respectively, as the possibly copied transition sequences have an outer left (for σ) or right (for σ') state which has already occured at that position in the sequence it was copied from, at an index higher than the copied sequence. Thus, this outer left or right state is deleted with the *red*-operator. Thirdly, the copying is always permitted in the context of adjustability, as at all positions in the string, the transition input character is adjustable to the transition input character in the sequence it was copied from at the same position. This holds because the sequence is copied at the same position. Thus, all characters in the slices for the copied sequence are adjustable to the transition input character in the new slice, as the adjustability relation is an equivalence relation.

As the yield of both σ and σ' remains the same during construction, concatenating the slice sequences in the end will yield a slice sequence with the concatenated yield, thereby meeting the last requirement for ϕ in Lemma 2. $\square$

We illustrate this approach by using diagrams to show the process of checking the states. Consider Figure 7, which shows the initial configuration with σ and σ' from which we will construct ϕ. As is clear from this figure, $\mathit{right}(\sigma) = (8, 5, 3)$ and $\mathit{left}(\sigma) = (8, 3, 5, 3)$ (hence, $\mathit{red}(\mathit{right}(\sigma)) = \mathit{red}(\mathit{left}(\sigma')) = (8, 5, 3)$. Figure 8 shows the first two evaluation rounds, which both succeed, as for both $i = 3, j = 4$ and $i = 2, j = 3$ it holds that $r_i = r'_j$. After decrementing again, however, we get $i = 1, j = 2$ and thus $r_i \neq r'_j$ (as $r_1 = 8, r'_2 = 3$). We then take the approach outlined in the proof of Lemma 2. We find that for $k = 3$, it holds that $r_k = r'_j$. Thus, we copy the transition sequence corresponding with r_3 directly above the transition sequence for r_i, as it shown in Figure 9. After copying, the new r_{i+1} matches r'_j, and we can continue with the next pair, which consists of r_1 ($i = 1$, as i is not decremented due to the copying in σ) and r'_1. This pair matches, which yields $i = 0, j = 0$ after decrementing both, allowing us to obtain ϕ by concatenating the two constructed slice sequences.

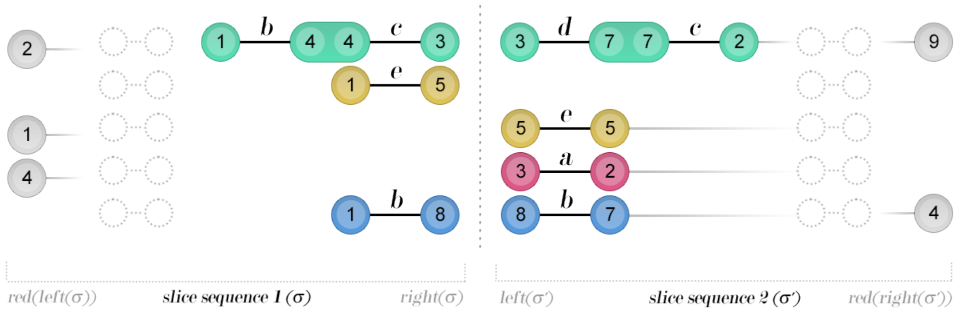

FIGURE 7. Initial configuration showing slice sequences σ and σ', to be combined to form ϕ.

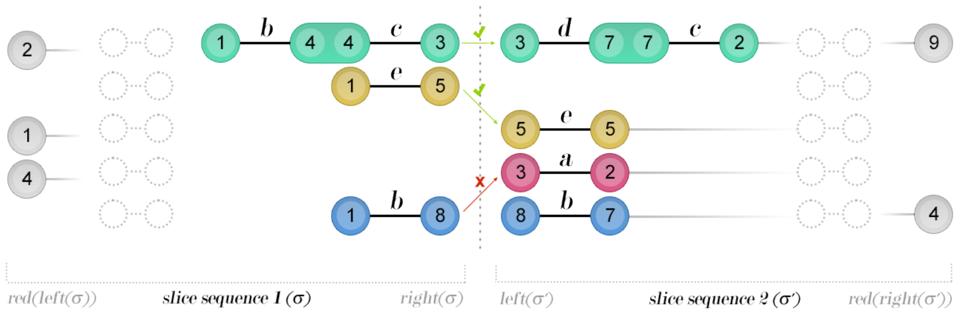

FIGURE 8. The first two states match, but the third one doesn't.

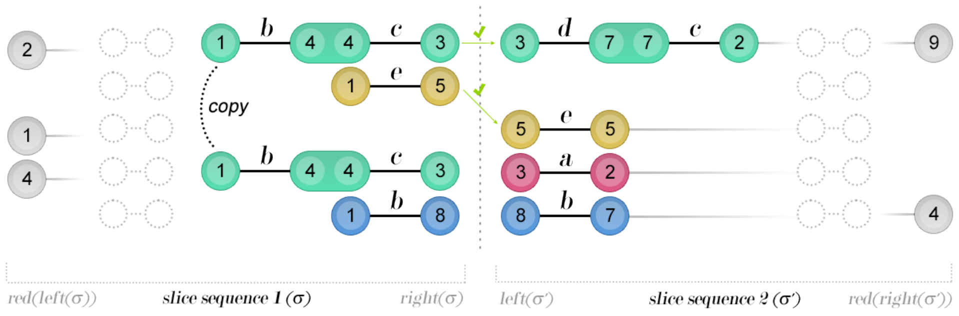

FIGURE 9. Copy the entire transition sequence for the matching state.

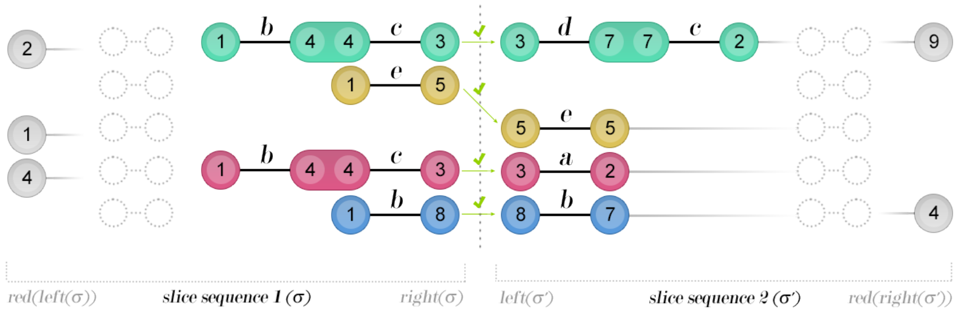

FIGURE 10. Check remaining states, which match, thus we have found ϕ

We are now in position to prove Lemma 1.

Proof for lemma 1. Let R be a sequence of reduced state sequences $R = \{r_0, \ldots, r_n\}$ for which $r_{i-1} \mapsto_{\alpha_i} r_i$ for all $1 \leq i \leq n$. We construct σ for which it holds that $\mathit{yield}(\sigma) = \alpha_1 \ldots \alpha_n$ by induction. For $n = 0$, the base case, we obtain the empty slice sequence, ϵ. For $n > 0$, the induction step, we construct the slice sequence by combining the slice sequence for $n-1$ with a slice sequence S. S is the slice sequence consisting of a single slice, namely the slice with source states r_{n-1}, target states r_n and transitions between them that form a slice $s\ell$ such that $\mathit{yield}(s\ell) = \alpha_n$. We know that such a slice always exists by Definition 7. Then, we combine the slice sequence from $n-1$ with S by Lemma 2, which has the yield $\alpha_1 \ldots \alpha_n$ as the slice sequence for $n-1$ has, by induction, the yield $\alpha_1 \ldots \alpha_{n-1}$ and $\mathit{yield}(S) = \alpha_n$. $\square$

8. PROOF

We are now in position to prove Theorem 1 by constructing a finite state transducer $T = (Q, \Sigma, \Gamma, I, F, \delta)$ mapping L to L_G.

Proof of Theorem 1. Let Σ be an alphabet, let $\mathcal{A}$ be the finite automaton in the normal form introduced in Section 4 accepting the set of guides G. We construct a finite state transducer $T = (Q, \Sigma, \Gamma, I, F, \delta)$ mapping any language $L \in \Sigma^*$ to L_G. Suppose $\mathcal{S}$ is the set of all possible reduced state sequences. The finite state transducer T is specified as follows:

- $Q = \mathcal{S}$
- $\Gamma = \Sigma$
- $I = \{\epsilon\}$
- $F = \{\epsilon\}$
- $\delta = \{(p, a, b, q) \mid p, q \in Q, a, b \in \Sigma, p \mapsto_b q, a \sim b\}$

For this proof to hold, it is important for it to maintain two properties. Firstly, it is important that any possible sequence of reduced state sequences with $\mapsto_\alpha$-relations can be obtained by following a path in our finite state transducer. This property holds as our finite state transducer is defined in this way: it consists of reduced state sequences with transitions directly based on the $\mapsto_\alpha$-relations.

Secondly, all strings generated by our finite state transducer from any input in L must produce an output in L_G (i.e. for an input $x \in L$ there can not be a path generating $x' \notin L_G$). As a path in our finite state transducer represents a sequence of reduced state sequences with corresponding transitions, this property follows from Lemma 1.

Thus, we have proven Theorem 1, concluding that cones are closed under the operation of guided rewriting. $\square$

9. Conluding remarks

We have studied an extension of the operation of guided rewriting as introduced in [1], using a regular set of guides instead of a finite set of guides. We have proven that families of languages that are closed under homomorphism, inverse homomorphism and intersection with regular language, so-called cones, are closed under this operation, thereby answering our research question.

Furthermore, looking closely at the finite state transducer constructed in Section 8, we can derive that not only are cones closed under regular guided rewriting, but so-called faithful cones as well. A faithful cone is a cone that is not necessarily closed under all homomorphisms, but only under homomorphisms called ϵ-free homomorphisms, which are homomorphisms that never map a letter to the empty word. As the finite state transducer we constructed in 8 never maps to the empty word, we can adapt Nivat's Theorem (mentioned in Section 3.1) to show that faithful cones, which include the families of recursive (REC) and context-sensitive (CS) languages, are indeed closed under regular guided rewriting as well.

References

[1] E. de Vink, H. Zantema, and D. Bosnacki. Combining insertion and deletion in rna-editing preserves regularity. In Gabriel Ciobanu, editor, *Membrane Computing and Biologically Inspired Process Calculi*, volume 100 of *EPTCS*, pages 48–62, 2012.

[2] F. Hennie. One-tape, off-line turing machine computations. *Information and Control*, 8(6):553–578, 1965.

[3] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages: Word, Language, Grammar*, volume 1 of *Handbook of Formal Languages*. Springer-Verlag GmbH, 1997.