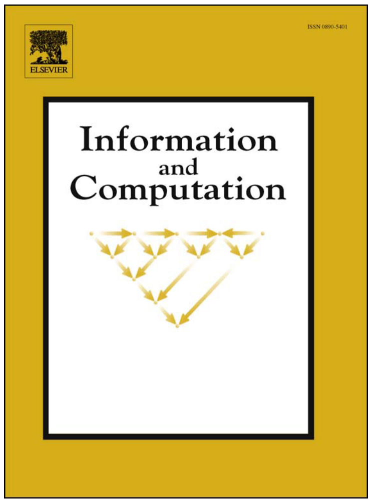
Accepted Manuscript

Characterising REGEX languages by regular languages equipped with factor-referencing

Markus L. Schmid



PII: S0890-5401(16)00010-9
DOI: http://dx.doi.org/10.1016/j.ic.2016.02.003
Reference: YINCO 4156

To appear in: *Information and Computation*

Received date: 27 November 2014
Revised date: 7 January 2016

Please cite this article in press as: M.L. Schmid, Characterising REGEX languages by regular languages equipped with factor-referencing, *Inf. Comput.* (2016), http://dx.doi.org/10.1016/j.ic.2016.02.003

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

ACCEPTED MANUSCRIPT

Characterising REGEX Languages by Regular Languages Equipped with Factor-Referencing[☆]

Markus L. Schmid

Fachbereich 4 – Abteilung Informatik, Universität Trier, D-54286 Trier, Germany

Abstract

A (factor-)reference in a word is a special symbol that refers to another factor in the same word; a reference is dereferenced by substituting it with the referenced factor. We introduce and investigate the class ref-REG of all languages that can be obtained by taking a regular language R and then dereferencing all possible references in the words of R. We show that ref-REG coincides with the class of languages defined by regular expressions as they exist in modern programming languages like Perl, Python, Java, etc. (often called REGEX languages).

Keywords: REGEX Languages, Regular Languages, Memory Automata

ACCEPTED MANUSCRIPT

1. Introduction

It is well known that most natural languages contain at least some structure that cannot be described by context-free grammars and also with respect to artificial languages, e. g., programming languages, it is often necessary to deal with structural properties that are inherently non-context-free (Floyd's proof (see [10]) that *Algol 60* is not context-free is an early example). Hence, as Dassow and Păun [8] put it, "the world seems to be non-context-free." On the other hand, the full class of context-sensitive languages, while powerful enough to model the structures appearing in natural languages and most formal languages, is often, in many regards, simply *too* much. Therefore, investigating those properties of languages that are inherently non-context-free is a classical research topic, which, in formal language theory is usually pursued in terms of *restricted* or *regulated rewriting* (see Dassow and Păun [8]), and in computational linguistics *mildly context-sensitive* languages are investigated (see, e. g., Kallmeyer [13]).

In [9], Dassow et al. summarise the three most commonly encountered non-context-free features in formal languages as *reduplication*, leading to languages of the form $\{ww \mid w \in \Sigma^*\}$, *multiple agreements*, modelled by languages of the

[☆]A preliminary version [17] of this paper was presented at the conference DLT 2014.
Email address: MSchmid@uni-trier.de (Markus L. Schmid)

ACCEPTED MANUSCRIPT

form $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 1\}$ and *crossed agreements*, as modeled by $\{\mathtt{a}^n\mathtt{b}^m\mathtt{c}^n\mathtt{d}^m \mid n, m \geq 1\}$. In this work, we solely focus on the first such feature: reduplication.

The concept of reduplication has been mainly investigated by designing language generators that are tailored to reduplications (e. g., L systems (see Kari et al. [14] for a survey), Angluin's pattern languages [2] or H-systems by Albert and Wegner [1]) or by extending known generators accordingly (e. g., Wijngaarden grammars, macro grammars, Indian parallel grammars or deterministic iteration grammars (cf. Albert and Wegner [1] and Bordihn et al. [3] and the references therein)). A more recent approach is to extend regular expressions with some kind of copy operation (e. g., pattern expressions by Câmpeanu and Yu [6], synchronized regular expressions by Della Penna et al. [15], EH-expressions by Bordihn et al. [3]). An interesting such variant are regular expressions with backreferences (REGEX for short), which play a central role in this work. REGEX are regular expressions that contain special symbols that refer to the word that has been matched to a specific subexpression. Unlike the other mentioned language descriptors, REGEX seem to have been invented entirely on the level of software implementation, without prior theoretical formalisation (see Friedl [12] for their practical relevance). An attempt to formalise and investigate REGEX and the class of languages they describe from a theoretical point of view has been started recently (see [4, 6, 16, 11]). This origin of REGEX from applications render their theoretical investigation difficult. As pointed out by Câmpeanu and Santean in [5], "we observe implementation inconsistencies, ambiguities and a lack of standard semantics." Unfortunately, to at least some extend, these conceptional problems inevitably creep into the theoretical literature as well.

Regular expressions often serve as an user interface for specifying regular languages, since finite automata are not easily defined by human users. On the other hand, due to their capability of representing regular languages in a concise way, regular expressions are deemed inappropriate for implementations and sometimes for proving theoretical results about regular languages (e. g., certain closure properties or decision problems). We encounter a similar situation with respect to REGEX (which, basically, are a variant of regular expressions), i. e., their widespread implementations suggest that they are considered practically useful for specifying languages, but the theoretical investigation of the language class they describe proves to be complicated. Hence, we consider it worthwhile to develop a characterisation of this language class, which is independent from actual REGEX.

To this end, we introduce the concept of *unresolved* reduplications on the word level. In a fixed word, such a reduplication is represented by a pointer or reference to a factor of the word and resolving or dereferencing such a reference is done by replacing the pointer by the value it refers to, e. g.,

$$w = \mathtt{a}\,\underbrace{\mathtt{b}\,\mathtt{a}\,\mathtt{c}}_{x}\hspace{-0.6em}\overbrace{\hphantom{\mathtt{c}}\,\mathtt{b}}^{y}\,\mathtt{c}\,\overbrace{x\,\mathtt{c}\,\mathtt{b}}^{z}\,z\,y\,\mathtt{a}\,,$$

where the symbols x, y and z are pointers to the factors marked by the brackets labelled with x, y and z, respectively. Resolving the references x and y yields

ACCEPTED MANUSCRIPT

$\mathtt{abacbcbaccb}z\mathtt{cba}$ and resolving reference z leads to $\mathtt{abacbcbaccbbaccbcba}$. Such words are called reference-words (or ref-words, for short) and sets of ref-words are ref-languages. For a ref-word w, $\mathcal{D}(w)$ denotes the word w with all references resolved and for a ref-language L, $\mathcal{D}(L) = \{\mathcal{D}(w) \mid w \in L\}$. We shall investigate the class of ref-regular languages, i. e., the class of languages $\mathcal{D}(L)$, where L is both regular and a ref-language, and, as our main result, we show that it coincides with the class of REGEX languages. Furthermore, by a natural extension of classical finite automata, we obtain a very simple automaton model, which precisely describes the class of ref-regular languages (= REGEX languages). This automaton model is used in order to introduce a subclass of REGEX languages, that, in contrast to other recently investigated such subclasses, has a polynomial time membership problem and we investigate the closure properties of this subclass. As a side product, we obtain a very simple alternative proof for the closure of REGEX languages under intersection with regular languages; a known result, which has first been shown by Câmpeanu and Santean [5] by much more elaborate techniques.

2. Definitions

Let $\mathbb{N} = \{1, 2, 3, \ldots\}$ and $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. For an alphabet B, the symbol B^+ denotes the set of all non-empty words over B and $B^* = B^+ \cup \{\varepsilon\}$, where ε is the empty word. For the *concatenation* of two words w_1, w_2 we write $w_1 \cdot w_2$ or simply $w_1 w_2$. We say that a word $v \in B^*$ is a *factor* of a word $w \in B^*$ if there are $u_1, u_2 \in B^*$ such that $w = u_1 v u_2$. For any word w over B, $|w|$ denotes the length of w, for any $b \in B$, by $|w|_b$ we denote the number of occurrences of b in w and for any $A \subseteq B$, we define $|w|_A = \sum_{b \in A} |w|_b$.

We use regular expressions as they are commonly defined (see, e. g., Yu [18]). By DFA and NFA, we refer to the set of deterministic and nondeterministic finite automata. Depending on the context, by DFA and NFA we also refer to an individual deterministic or nondeterministic automaton, respectively.

For any language descriptor D, by $L(D)$ we denote the language described by D and for any class $\mathfrak{D}$ of language descriptors, let $\mathcal{L}(\mathfrak{D}) = \{L(D) \mid D \in \mathfrak{D}\}$. In the whole paper, we assume Σ to be an arbitrary finite alphabet with $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\} \subseteq \Sigma$.

We next formally define the concept of reference-words that is intuitively described in the introduction.

2.1. References in Words

Let $\Gamma = \{[_{x_i},]_{x_i}, x_i \mid i \in \mathbb{N}\}$, where, for every $i \in \mathbb{N}$, the pairs of symbols $[_{x_i}$ and $]_{x_i}$ are *parentheses* and the symbols x_i are *variables*. For the sake of convenience, we shall sometimes also use the symbols x, y and z to denote arbitrary variables. A *reference-word over* Σ (or *ref-word*, for short) is a word over the alphabet $(\Sigma \cup \Gamma)$. For every $i \in \mathbb{N}$, let $h_i : (\Sigma \cup \Gamma)^* \to (\Sigma \cup \Gamma)^*$ be the morphism with $h_i(z) = z$ for all $z \in \{[_{x_i},]_{x_i}, x_i\}$, and $h_i(z) = \varepsilon$ for all

ACCEPTED MANUSCRIPT

$z \notin \{[_{x_i},]_{x_i}, x_i\}$. A reference word w is *valid* if, for every $i \in \mathbb{N}$,

$$h_i(w) = x_i^{\ell_1}\, [_{x_i}\,]_{x_i}\, x_i^{\ell_2}\, [_{x_i}\,]_{x_i}\, x_i^{\ell_3} \ldots x_i^{\ell_{k_i-1}}\, [_{x_i}\,]_{x_i}\, x_i^{\ell_{k_i}}\,, \qquad (1)$$

for some $k_i \in \mathbb{N}$ and $\ell_j \in \mathbb{N}_0$, $1 \leq j \leq k_i$. Intuitively, a ref-word w is valid if, for every $i \in \mathbb{N}$, there is a number of matching pairs of parentheses $[_{x_i}$ and $]_{x_i}$ that are not nested and, furthermore, no occurrence of x_i is enclosed by such a matching pair of parentheses. However, for $i, j \in \mathbb{N}$ with $i \neq j$, we do not impose any restriction with respect to the relative order of symbols $[_{x_i},]_{x_i}, x_i$ on the one hand and symbols $[_{x_j},]_{x_j}, x_j$ on the other. In particular, it is not required that w is a well-formed parenthesised expression with respect to *all* occurring parentheses, e. g., the structure $w_1[_{x_i} w_2 [_{x_j} w_3]_{x_i} w_4]_{x_j} w_5$ with $i \neq j$, $|w_2 w_3|_{]_{x_i}} = 0$ and $|w_3 w_4|_{]_{x_j}} = 0$ is possible.

The set of valid ref-words is denoted by $\Sigma^{[*]}$. A factor $[_x\, u\,]_x$ of a $w \in \Sigma^{[*]}$ where the occurrences of $[_x$ and $]_x$ are matching parentheses, i. e., $|u|_{]_x} = 0$, is called a *reference for variable* x, and u is the *value* of this reference. A reference is a *first order reference*, if its value does not contain another reference and it is called *pure*, if it is a first order reference and its value does not contain variables. Two references of some ref-word w are *overlapping* if one reference contains exactly one of the delimiting parentheses of the other reference, e. g., in $w_1[_x w_2 [_y w_3]_x w_4]_y w_5$ the references $[_x w_2 [_y w_3]_x$ and $[_y w_3]_x w_4]_y$ are overlapping. Let $w \in \Sigma^{[*]}$ and let x be a variable that occurs in w. An occurrence of a variable x in w that is not preceded by a reference for x is called *undefined*. Every occurrence of a variable x in w that is not undefined *refers* to the reference for x, which precedes this occurrence. This definition is illustrated by Equation 1, where all $k_i - 1$ references for variable x_i are shown and, for every j, $1 \leq j \leq k_i - 1$, the ℓ_{j+1} occurrences of x_i between the j^{th} and $(j+1)^{\text{th}}$ reference for x_i are exactly the occurrences of x_i that refer to the j^{th} reference for variable x_i. The first ℓ_1 occurrences of x_i are all undefined. In order to illustrate the definitions introduced above, we consider the following example:

Example 1.

$$w_1 = [_x \mathtt{ab}]_x [_y \mathtt{c}x\mathtt{b}]_y [_x \mathtt{bb}y]_x \mathtt{c}y\mathtt{b}y[_y x]_y \mathtt{cc}, \quad w_2 = [_x \mathtt{ba}x]_x \mathtt{a}x[_x \mathtt{bc}]_x [_x \mathtt{ba}[_x \mathtt{a}]_x \mathtt{a}]_x x\,,$$
$$w_3 = [_x [_y \mathtt{b}]_x \mathtt{c}x[_x \mathtt{b}]_y zy\mathtt{b}[_y \mathtt{c}z]_y z[_z \mathtt{cc}]_x]_z, \quad w_4 = [_x \mathtt{a}[_y \mathtt{b}[_z \mathtt{bba}]_z \mathtt{c}]_y \mathtt{b}y\mathtt{b}]_x xy\,.$$

The words w_1, w_3 and w_4 are valid ref-words, whereas w_2 is not valid, since there is a reference for x that contains an occurrence of x and also a reference for x that contains other references for the same variable x. All references of w_1 are first order references, whereas the reference for variable x in w_4 is an example for a reference that is not a first order reference, since it contains a reference for another variable y. In w_3, the first reference for x and the last reference for z are the only pure references. Furthermore, all occurrences of z in w_3 are undefined and the leftmost references for x and y are overlapping.

For the sake of convenience, from now on, we call valid ref-words simply ref-words. If a word over $(\Sigma \cup \Gamma)$ is not a ref-word, then we always explicitly mention this.

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

Next, we define how a ref-word over Σ can be dereferenced, i. e., how it can be transformed into a (normal) word over Σ. To this end, let $w \in \Sigma^{[*]}$. The *dereference* of w, denoted by $\mathcal{D}(w)$, is constructed by first deleting all undefined occurrences of variables in w and then substituting all pure references and all occurrences of variables that refer to them by its value (ignoring possible parentheses in the value), until there is no pure reference left. We illustrate this definition with an example:

Example 2.

$$\mathcal{D}(z\mathtt{a}[_z x[_x y\mathtt{b}[_y \mathtt{c}]_x \mathtt{b} x[_x \mathtt{c}]_y \mathtt{b}]_x y\mathtt{c}]_z x\mathtt{c}z) = \mathcal{D}(\mathtt{a}[_z [_x \mathtt{b}[_y \mathtt{c}]_x \mathtt{b} x[_x \mathtt{c}]_y \mathtt{b}]_x y\mathtt{c}]_z x\mathtt{c}z) =$$

$$\mathcal{D}(\mathtt{a}[_z \underbrace{\mathtt{b}[_y \mathtt{c}}_{x}\, \mathtt{b}\, \underbrace{\mathtt{bc}}_{x}[_x \mathtt{c}]_y \mathtt{b}]_x y\mathtt{c}]_z x\mathtt{c}z) = \mathcal{D}(\mathtt{a}[_z \mathtt{b}\, \underbrace{\mathtt{cbbc}[_x \mathtt{c}\, \mathtt{b}}_{y}]_x\, \underbrace{\mathtt{cbbcc}}_{y}\, \mathtt{c}]_z x\mathtt{c}z) =$$

$$\mathcal{D}(\mathtt{a}[_z \mathtt{bcbbc}\, \underbrace{\mathtt{cb}}_{x}\, \mathtt{cbbccc}]_z\, \underbrace{\mathtt{cb}}_{x}\, \mathtt{c}z) = \mathtt{a}\, \underbrace{\mathtt{bcbbccbcbbccc}}_{z}\, \mathtt{cbc}\, \underbrace{\mathtt{bcbbccbcbbccc}}_{z}\ .$$

If we delete all undefined variables in w and then substitute the variables that refer to pure references as described above, then it does not matter in which order this is done. This is due to the fact that, since pure references do not contain variables, their content does not change by substituting some variables by words. Furthermore, a reference that is not pure is turned into a pure one by substituting all its variables by words, no matter in which order this is done. These considerations show that $\mathcal{D}$ is well-defined; next we show that, for every ref-word w, $\mathcal{D}(w) \in \Sigma^*$.

Proposition 3. *For every* $w \in \Sigma^{[*]}$, $\mathcal{D}(w) \in \Sigma^*$.

Proof. Let w be a ref-word without any undefined occurrences of variables and with at least one reference, which obviously means that it contains at least one first order reference. Let π be the leftmost first order reference in w. If π contains an occurrence of a variable, then, since this variable is not undefined, it must refer to some reference to the left of π, contradicting the assumption that π is the leftmost first order reference. Consequently, if there are no undefined occurrences of variables, then the leftmost first order reference must be pure. Hence, in the construction of $\mathcal{D}(w)$, as long as the current word contains a reference, there is at least one pure reference. This directly implies the statement of the Proposition. □

Obviously, a ref-word $w \in \Sigma^{[*]}$ can be considered as some kind of Lempel-Ziv compression of the word $\mathcal{D}(w)$. However, in this work we are exclusively concerned with language theoretic aspects of ref-words and we wish to point out that for a successful application of ref-words in data-compression, one would need an inverse function of $\mathcal{D}$, i. e., an actual compression function.

Having defined ref-words, we are now ready to extend this concept to languages by considering sets of ref-words. The dereference function then automatically generalises from an operation on words to an operation on languages.

ACCEPTED MANUSCRIPT

2.2. References in Languages

For every $i \in \mathbb{N}$, let $\Gamma_i = \{[_{x_j},]_{x_j}, x_j \mid j \leq i\}$. A set of ref-words L is a *ref-language* if $L \subseteq (\Sigma \cup \Gamma_i)^*$, for some $i \in \mathbb{N}$.[1] For the sake of convenience, we simply write $L \subseteq \Sigma^{[*]}$ to denote that L is a ref-language. For every ref-language L, we define the *dereference of* L by $\mathcal{D}(L) = \{\mathcal{D}(w) \mid w \in L\}$ and, for any class $\mathfrak{L}$ of ref-languages, $\mathcal{D}(\mathfrak{L}) = \{\mathcal{D}(L) \mid L \in \mathfrak{L}\}$.

An $L \subseteq \Sigma^{[*]}$ is a *regular ref-language* if L is regular. A language L is called *ref-regular* if it is the dereference of a regular ref-language, i. e., $L = \mathcal{D}(L')$ for some regular ref-language L'. For example, the copy language $L_c = \{ww \mid w \in \Sigma^*\}$ is ref-regular, since $L_c = \mathcal{D}(L'_c)$, where L'_c is the regular ref-language $\{[_x\, w\,]_x\, x \mid w \in \Sigma^*\}$. The class of ref-regular languages is denoted by ref-REG.

By definition, REG $\subseteq$ ref-REG and it can be easily shown that ref-REG is contained in the class of context-sensitive languages (note that this also trivially follows from our main result that ref-REG equals the class of REGEX languages, which are known to be context-sensitive). On the other hand, the class of context-free languages is not included in ref-REG, e. g., $\{\mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N}\} \notin$ ref-REG (see Câmpeanu et al. [4]).

Other interesting examples of ref-regular languages are the set of imprimitive words: $\mathcal{D}(\{[_x\, w\,]_x\, x^n \mid w \in \Sigma^*, n \geq 1\})$, the set of words $\mathtt{a}^n$, where n is not prime: $\mathcal{D}(\{[_x\, \mathtt{a}^m\,]_x\, x^n \mid m, n \geq 2\})$, the set of bordered words: $\mathcal{D}(\{[_x\, u\,]_x\, v\, x \mid u, v \in \Sigma^*, |u| \geq 1\})$ and the set of words containing a square: $\mathcal{D}(\{u\, [_x\, v\,]_x\, x\, w \mid u, v, w \in \Sigma^*, |v| \geq 1\})$.

2.3. References in Expressions

Extended regular expressions with backreferences (or REGEX for short) are regular expressions in which subexpressions can be labelled by a number i and then the symbol $\backslash i$ constitutes a *backreference* to this subexpression. When we match a REGEX to a word, then the symbol $\backslash i$ is treated as exactly the word the subexpression labelled by i has been matched to. For example, $(_1(\mathtt{a}+\mathtt{b})^*)_1(\mathtt{c}^* + (_2\mathtt{a}^*\mathtt{b})_2)(\backslash 2 + \mathtt{b}^*)(\backslash 1)^*$ is a REGEX in which $\backslash 1$ has to be matched to whatever the subexpression $(_1(\mathtt{a}+\mathtt{b})^*)_1$ has been matched to and $\backslash 2$ has to be matched to whatever the subexpression $(_2\mathtt{a}^*\mathtt{b})_2$ has been matched to. For more detailed definitions and further information on REGEX, we refer to [4, 16, 11, 5].

We can also define the syntax and semantics of REGEX in a convenient way by using classical regular expressions that define ref-languages. More precisely, any REGEX over Σ with k backreferences can be transformed into a classical regular expression over $(\Sigma \cup \Gamma_k)$ by replacing all occurrences of parenthesis $(_i$ and $)_i$ by the symbols $[_{x_i}$ and $]_{x_i}$, respectively, and all backreferences $\backslash i$ by x_i, e. g., the REGEX $r = (_1(\mathtt{a}+\mathtt{b})^*)_1(\mathtt{c}^* + (_2\mathtt{a}^*\mathtt{b})_2)(\backslash 2 + \mathtt{b}^*)(\backslash 1)^*$ translates into the regular expression $r' = [_{x_1}(\mathtt{a}+\mathtt{b})^*]_{x_1}(\mathtt{c}^* + [_{x_2}\mathtt{a}^*\mathtt{b}]_{x_2})(x_2 + \mathtt{b}^*)(x_1)^*$. It can be easily verified (a formal proof is omitted and left to the reader) that $L(r')$

[1]By requiring the number of symbols from Γ to be finite, ref-languages are languages over finite alphabets, which makes them suitable for finite automata.

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

is a ref-language and $\mathcal{D}(L(r')) = L(r)$. This observation also shows that every REGEX language is a ref-regular language:

Proposition 4. $\mathcal{L}(\text{REGEX}) \subseteq \text{ref-REG}$.

On the other hand, a regular expression s with $L(s) \subseteq \Sigma^{[*]}$ does not translate into a REGEX in an obvious way, which is due to the fact that in s it is not necessarily the case that every occurrence of $[_x$ matches with an occurrence of $]_x$ and, furthermore, even matching pairs of parentheses do not necessarily enclose subexpressions. For example, the regular expression

$$s = [_{x_1}\texttt{((}[_{x_2}\texttt{b}^*]_{x_1}\texttt{a}^* x_1 [_{x_1}\texttt{)} + \texttt{(}[_{x_2}\texttt{a}^*\texttt{c}^*\texttt{))}\,\texttt{ba}]_{x_2}\texttt{(}x_2 + \texttt{d}\texttt{)}^*]_{x_1}\texttt{a}x_1$$

describes a ref-language, but replacing the symbols $[_{x_i},]_{x_i}$ and x_i by $(_i,)_i$ and $\backslash i$, respectively, does not yield a REGEX. In the following, we say that a regular expression r over the alphabet $(\Sigma \cup \Gamma_k)$ has the REGEX *property* if $L(r) \subseteq \Sigma^{[*]}$ and for each factor $[_{x_i} r']_{x_i}$ with $|r'|_{]_{x_i}} = 0$, r' is a subexpression of r. If a regular expression r has the REGEX *property*, then it can be easily seen that it can be transformed into a REGEX r' by replacing the symbols $[_{x_i},]_{x_i}$ and x_i by $(_i,)_i$ and $\backslash i$, respectively, and $L(r') = \mathcal{D}(L(r))$. This directly implies the following proposition:

Proposition 5. *Let r be a regular expression over $(\Sigma \cup \Gamma_k)$, $k \in \mathbb{N}$. If r has the* REGEX *property, then $\mathcal{D}(L(r)) \in \mathcal{L}(\text{REGEX})$.*

This means that in order to prove $\mathcal{L}(\text{REGEX}) = \text{ref-REG}$, we need to show that every regular expression r that describes a ref-language can be transformed into a regular expression r' with the REGEX property and that is equivalent to r with respect to the dereference of the described ref-languages, i. e., $\mathcal{D}(L(r)) = \mathcal{D}(L(r'))$ (but not necessarily $L(r) = L(r')$). In Sections 3 and 4, we solve this task by first characterising ref-REG by a simple extension of finite automata.

3. Memory Automata

By a natural extension of classical finite automata, we now define memory automata, which are the main technical tool for proving the results of this paper.

A memory automaton is a classical NFA that is equipped with a finite number of k memory cells, each capable of storing a word. Each memory is either *closed*, which means that it is not affected in a transition, or *open*, which means that the currently scanned input symbol is appended to its content. In a transition it is possible to *consult* a closed memory, which means that its content, if it is a prefix of the remaining input, is consumed in one step from the input and, furthermore, also stored in all the open memories. A closed memory can be opened again, but then it completely loses its previous content; thus, memories are not able to store subsequences, but only factors of the input. We shall now formally define the model of memory automata.

ACCEPTED MANUSCRIPT

Definition 1. For every $k \in \mathbb{N}$, a *k-memory automaton*, denoted by MFA(k), is a tuple $M = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of *states*, Σ is a finite *alphabet*, q_0 is the *initial state*, F is the set of *final states* and

$$\delta : Q \times (\Sigma \cup \{\varepsilon\} \cup \{1, 2, \ldots, k\}) \to \mathcal{P}(Q \times \{\mathtt{o}, \mathtt{c}, \diamond\}^k)$$

is the *transition function* (where $\mathcal{P}(A)$ denotes the power set of a set A). The elements $\mathtt{o}$, $\mathtt{c}$ and $\diamond$ are called *memory instructions*.

A configuration of M is a tuple $(q, w, (u_1, r_1), \ldots, (u_k, r_k))$, where $q \in Q$ is the *current state*, w is the *remaining input*, for every i, $1 \leq i \leq k$, $u_i \in \Sigma^*$ is the *content of memory i* and $r_i \in \{\mathtt{O}, \mathtt{C}\}$ is the *status of memory i* (i. e., $r_i = \mathtt{O}$ means that memory i is open and $r_i = \mathtt{C}$ means that it is closed). For a memory status $r \in \{\mathtt{O}, \mathtt{C}\}$ and a memory instruction $s \in \{\mathtt{o}, \mathtt{c}, \diamond\}$, we define $r \oplus s = \mathtt{O}$ if $s = \mathtt{o}$, $r \oplus s = \mathtt{C}$ if $s = \mathtt{c}$ and $r \oplus s = r$ if $s = \diamond$. Furthermore, for a tuple $(r_1, \ldots, r_k) \in \{\mathtt{O}, \mathtt{C}\}^k$ of memory statuses and a tuple $(s_1, \ldots, s_k) \in \{\mathtt{o}, \mathtt{c}, \diamond\}^k$ of memory instructions, we define $(r_1, \ldots, r_k) \oplus (s_1, \ldots, s_k) = (r_1 \oplus s_1, \ldots, r_k \oplus s_k)$.

M can change from a configuration $c = (q, w, (u_1, r_1), \ldots, (u_k, r_k))$ to a configuration $c' = (p, w', (u'_1, r'_1), \ldots, (u'_k, r'_k))$, denoted by $c \vdash_M c'$, if there exists a transition $\delta(q, b) \ni (p, s_1, \ldots, s_k)$ such that $w = v\,w'$, where $v = b$ if $b \in (\Sigma \cup \{\varepsilon\})$ and $v = u_b$ and $r_b = \mathtt{C}$ if $b \in \{1, 2, \ldots, k\}$. Furthermore, $(r'_1, \ldots, r'_k) = (r_1, \ldots, r_k) \oplus (s_1, \ldots, s_k)$ and, for every i, $1 \leq i \leq k$,

$$u'_i = \begin{cases} u_i v & \text{if } r'_i = r_i = \mathtt{O}, \\ v & \text{if } r'_i = \mathtt{O} \text{ and } r_i = \mathtt{C}, \\ u_i & \text{if } r'_i = \mathtt{C}. \end{cases}$$

The symbol $\vdash^*_M$ denotes the reflexive and transitive closure of $\vdash_M$. For any $w \in \Sigma^*$, a configuration $(p, v, (u_1, r_1), \ldots, (u_k, r_k))$ is *reachable* (*on input w*), if

$$(q_0, w, (\varepsilon, \mathtt{C}), \ldots, (\varepsilon, \mathtt{C})) \vdash^*_M (p, v, (u_1, r_1), \ldots, (u_k, r_k)).$$

A $w \in \Sigma^*$ is accepted by M if a configuration $(q_f, \varepsilon, (u_1, r_1), \ldots, (u_k, r_k))$ with $q_f \in F$ is reachable on input w and $L(M)$ is the set of words accepted by M.

For any $k \in \mathbb{N}$, MFA(k) is the class of k-memory automata and MFA $= \bigcup_{k \geq 1}$ MFA(k). We also use MFA(k) and MFA in order to denote an instance of a k-memory automaton or a memory automaton with some number of memories, respectively. The set $\mathcal{L}(\text{MFA}) = \{L(M) \mid M \in \text{MFA}\}$ is the *class of* MFA *languages*. Next, we illustrate how an MFA accepts a language with an example:

Example 6. *Let $L = \{v_1v_2v_3v_1v_2v_2v_3 \mid v_1, v_2, v_3 \in \{\mathtt{a}, \mathtt{b}\}^*\}$. An* MFA(3) *can accept L by reading a prefix $u = v_1v_2v_3$ of the input and storing v_1, v_2 and v_3 in the memories 1, 2 and 3, respectively. The length of u as well as the factorisation $u = v_1v_2v_3$ is nondeterministically guessed. Now we can check whether the remaining input equals $v_1v_2v_2v_3$ by first consulting memory 1, then memory 2 twice and finally memory 3. Alternatively, while reading the prefix $u = v_1v_2v_3$, we can also store v_1v_2 in memory 1 and v_2v_3 in memory 2 and then check*

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

whether the remaining input equals $v_1v_2v_2v_3$ by consulting first memory 1 and then memory 2. We point out that this alternative approach, which needs one memory less, requires the factor v_2 of the input to be simultaneously recorded by both memories or, in other words, the factors recorded by the memories overlap in the input word. This possible overlap of recorded factors shall play an important role for our further results.

An MFA is not necessarily complete, i. e., it may contain states q for which $\delta(q, b)$ is undefined for some $b \in \Sigma$. In a sense, we could make any MFA complete by adding a non-accepting trap-state q_t (i. e., $\delta(q_t, b) = q_t$, $b \in \Sigma$) and transitions $\delta(q, b) = q_t$ for all undefined $\delta(q, b)$. As can be easily verified, this modification does not change the accepted language. However, if for a state q a transition is defined that consults a memory, then, by adding transitions of the form $\delta(q_t, b) = q_t$, we create a nondeterministic choice, since we may choose now between consulting the memory or changing in the trap state. Consequently, as we shall see later, this classical idea of completing an automaton does not work for *deterministic* MFA.

In the remainder of this section, we shall establish some basic properties of memory automata. The purpose of these results is twofold. Some of them are simply necessary for proving our main result in Section 4, while others are general observations, which are motivated by the fact that MFA provide an automaton characterisation of REGEX languages and we hope that they will be a helpful tool in the further investigation of this language class and may serve as a foundation for algorithms.

3.1. Determinism in Memory Automata

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a k-memory automaton. M is *pseudo deterministic* if δ is a function $Q \times (\Sigma \cup \{\varepsilon\} \cup \{1, 2, \ldots, k\}) \to Q \times \{\mathtt{o}, \mathtt{c}, \diamond\}^k$, i. e., for every $q \in Q$ and $b \in (\Sigma \cup \{\varepsilon\} \cup \{1, 2, \ldots, k\})$, $|\delta(q, b)| \leq 1$. Thus, for pseudo deterministic MFA, we shall write transitions as $\delta(q, b) = (p, s_1, s_2, \ldots, s_k)$ instead of $\delta(q, b) \ni (p, s_1, s_2, \ldots, s_k)$. However, a pseudo deterministic memory automaton is not fully deterministic in the classical sense, since it is still possible that for a $q \in Q$ both $\delta(q, i)$ and $\delta(q, b)$ are defined for some $i \in \{1, 2, \ldots, k\}$, $b \in \Sigma$, or both $\delta(q, i)$ and $\delta(q, j)$ are defined for some $i, j \in \{1, 2, \ldots, k\}$, $i \neq j$, which constitutes a nondeterministic choice.

Any MFA can be transformed into an equivalent pseudo deterministic one by applying a variant of the subset construction that also takes the memories into account, i. e., instead of $\mathcal{P}(Q)$ as the set of new states, we use $\mathcal{P}(Q) \times \{\mathtt{O}, \mathtt{C}\}^k$.

Lemma 7. *Let $k \in \mathbb{N}$. For every $M \in \mathrm{MFA}(k)$, there exists a pseudo deterministic $M' \in \mathrm{MFA}(k)$ with $L(M) = L(M')$.*

Proof. Let $M = (Q, \Sigma, \delta, q_0, F)$ be an MFA(k) that is not pseudo deterministic. We construct a pseudo deterministic $M' = (Q', \Sigma, \delta', q'_0, F') \in \mathrm{MFA}(k)$ with $L(M') = L(M)$. The idea is to perform a subset construction on M that

ACCEPTED MANUSCRIPT

also takes the statuses of the memories into account. More formally, we define

$$\begin{aligned}
Q' &= \{[T, r_1, r_2, \ldots, r_k] \mid T \subseteq Q, r_i \in \{\mathtt{O}, \mathtt{C}\}, 1 \le i \le k\}\,,\\
F' &= \{[T, r_1, r_2, \ldots, r_k] \mid [T, r_1, r_2, \ldots, r_k] \in Q', T \cap F \neq \emptyset\} \text{ and}\\
q_0' &= [\{q_0\}, \mathtt{C}, \mathtt{C}, \ldots, \mathtt{C}]\,.
\end{aligned}$$

Next, we define the transition function δ'. For every $[T, r_1, r_2, \ldots, r_k] \in Q'$, $b \in (\Sigma \cup \{\varepsilon\} \cup \{1, 2, \ldots, k\})$, $r_1', r_2', \ldots, r_k' \in \{\mathtt{O}, \mathtt{C}\}$ and $s_1, s_2, \ldots, s_k \in \{\mathtt{o}, \mathtt{c}, \diamond\}$ with $(r_1', \ldots, r_k') = (r_1, \ldots, r_k) \oplus (s_1, \ldots, s_k)$, we define $\delta'([T, r_1, r_2, \ldots, r_k], b) = ([T', r_1', r_2', \ldots, r_k'], s_1, \ldots, s_k)$, where

$$T' = \bigcup_{t \in T} \{t' \mid \delta(t, b) \ni (t', s_1, \ldots, s_k)\}\,.$$

This concludes the definition of M' and it can be easily seen that M' is in fact pseudo deterministic. It remains to prove that $L(M') = L(M)$.

We assume that

$$(q, w, (u_1, r_1), \ldots, (u_k, r_k)) \vdash_M (p, w', (u_1', r_1'), \ldots, (u_k', r_k'))\,,$$

which implies that there exists a transition $\delta(q, b) \ni (p, s_1, \ldots, s_k)$ such that $w = v\,w'$, where $v = b$ if $b \in (\Sigma \cup \{\varepsilon\})$ and $v = u_b$ and $r_b = \mathtt{C}$ if $b \in \{1, 2, \ldots, k\}$. Furthermore, for every i, $1 \le i \le k$, $(r_1', \ldots, r_k') = (r_1, \ldots, r_k) \oplus (s_1, \ldots, s_k)$ and

$$u_i' = \begin{cases} u_i v & \text{if } r_i' = r_i = \mathtt{O},\\ v & \text{if } r_i' = \mathtt{O} \text{ and } r_i = \mathtt{C},\\ u_i & \text{if } r_i' = \mathtt{C}. \end{cases}$$

Hence, as defined above, for every $T \subseteq Q$ with $q \in T$, $\delta'([T, r_1, r_2, \ldots, r_k], b) = ([T', r_1', r_2', \ldots, r_k'], s_1, \ldots, s_k)$ holds, where $p \in T'$. In particular, this implies that

$$\begin{aligned}
&([T, r_1, \ldots, r_k], w, (u_1, r_1), \ldots, (u_k, r_k)) \vdash_{M'}\\
&([T', r_1', \ldots, r_k'], w', (u_1', r_1'), \ldots, (u_k', r_k'))\,.
\end{aligned}$$

On the other hand, if, for some $T, T' \subseteq Q$ with $q \in T$ and $p \in T'$,

$$\begin{aligned}
&([T, r_1, \ldots, r_k], w, (u_1, r_1), \ldots, (u_k, r_k)) \vdash_{M'}\\
&([T', r_1', \ldots, r_k'], w', (u_1', r_1'), \ldots, (u_k', r_k'))\,,
\end{aligned}$$

then there exists a transition $\delta'([T, r_1, \ldots, r_k], b) = ([T', r_1', \ldots, r_k'], s_1, \ldots, s_k)$ such that $w = v\,w'$, where $v = b$ if $b \in (\Sigma \cup \{\varepsilon\})$ and $v = u_b$ and $r_b = \mathtt{C}$ if $b \in \{1, 2, \ldots, k\}$. Furthermore, for every i, $1 \le i \le k$, $(r_1', \ldots, r_k') = (r_1, \ldots, r_k) \oplus (s_1, \ldots, s_k)$ and

$$u_i' = \begin{cases} u_i v & \text{if } r_i' = r_i = \mathtt{O},\\ v & \text{if } r_i' = \mathtt{O} \text{ and } r_i = \mathtt{C},\\ u_i & \text{if } r_i' = \mathtt{C}. \end{cases}$$

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

By definition of δ', this implies that $\delta(q,b) \ni (p, s_1, \ldots, s_k)$, which means that

$$(q, w, (u_1, r_1), \ldots, (u_k, r_k)) \vdash_M (p, w', (u'_1, r'_1), \ldots, (u'_k, r'_k))\,.$$

Consequently, by induction we can conclude that, for every $w \in \Sigma^*$ and $q_f \in F$,

$$(q_0, w, (\varepsilon, \mathtt{C}), \ldots, (\varepsilon, \mathtt{C})) \vdash_M^* (q_f, \varepsilon, (u_1, r_1), \ldots, (u_k, r_k))$$

if and only if, for some $T_f \subseteq Q$ with $q_f \in T_f$,

$$\begin{aligned}&([\{q_0\}, \mathtt{C}, \ldots, \mathtt{C}], w, (\varepsilon, \mathtt{C}), \ldots, (\varepsilon, \mathtt{C})) \vdash_{M'}^* \\ &([T_f, r_1, \ldots, r_k], \varepsilon, (u_1, r_1), \ldots, (u_k, r_k))\,.\end{aligned}$$

This directly implies that $L(M') = L(M)$. □

Analogously to the definition of determinism for classical finite automata, we can define determinism for memory automata as the situation that in every state there is at most one applicable transition. More precisely, a k-memory automaton $M = (Q, \Sigma, \delta, q_0, F)$ is *deterministic* if it is ε-free and, for every $q \in Q$ and $b \in \Sigma$, $|\bigcup_{i=1}^{k} \delta(q,i)| + |\delta(q,b)| \leq 1$. In other words, M is deterministic if it is pseudo deterministic and if an input symbol can be read in a state, then no memory can be consulted and if a memory i can be consulted, then it is not possible to read an input symbol or to consult a memory other than i.

We denote the class of deterministic k-memory automata by DMFA(k) and DMFA $= \bigcup_{k \in \mathbb{N}}$ MFA(k).

The MFA(3) that is sketched in Example 6 is pseudo deterministic, since in the phase of reading the prefix $v_1 v_2 v_3$, for every state and every symbol there is exactly one transition reading this symbol and exactly one ε-transition which closes a memory and opens the next one, and in the phase of reading the suffix $v_1 v_2 v_2 v_3$ there is exactly one transition defined for every state, which consults a memory. However, it is not deterministic, since in the initial phase we have to guess whether or not we read another symbol of v_1 or whether we close the currently open memory and open the next one in order to start recording v_2. On the other hand, an example of a language that can be accepted by a deterministic MFA is $\{w\mathtt{c}w \mid w \in \{\mathtt{a}, \mathtt{b}\}^*\}$. From an intuitive point of view, one would suggest that deterministic MFA are weaker than general memory automata and this intuition shall be proved correct in Section 5, where the class $\mathcal{L}$(DMFA) is investigated in more detail.

3.2. Normal Forms of Memory Automata

For memory automata, it is possible that in the same transition the statuses of memories are changed and a part of the input is read (either by reading a single symbol or by consulting a memory). If this happens, then the *new* status of each memory determines whether or not the read input is also recorded in the memory. More precisely, if a closed memory is opened and in the same transition a symbol is read, then it is recorded by the memory; if, on the other hand, a memory is closed and a part of the input is read in the same transition, then this

ACCEPTED MANUSCRIPT

is not stored in the memory. It is also possible that in a transition a memory is opened or closed even though it is already open or closed, respectively, and in this case the instruction of opening or closing this memory has obviously the same meaning as the special instruction $\diamond$.

These peculiarities may unnecessarily complicate an MFA and from an intuitive point of view, one would assume that they can be removed. In this regard, we define the following normal form:

Definition 2. Let $M = (Q, \Sigma, \delta, q_0, F)$ be an MFA(k), $k \in \mathbb{N}$. We say that M is in *normal form* if the following conditions are satisfied. For every transition $\delta(q, b) \ni (p, s_1, \ldots, s_k)$, if $s_i \neq \diamond$ for some i, $1 \leq i \leq k$, then $b = \varepsilon$ and $s_j = \diamond$, for all j with $1 \leq j \leq k$, $i \neq j$. If M reaches configuration $(q, w, (u_1, r_1), \ldots, (u_k, r_k))$ in a computation and the transition $\delta(q, b) \ni (p, s_1, \ldots, s_k)$ is applicable next, then, for every i, $1 \leq i \leq k$, if $r_i = \mathtt{O}$, then $s_i \neq \mathtt{o}$ and if $r_i = \mathtt{C}$, then $s_i \neq \mathtt{c}$.

Any MFA can be transformed into an equivalent one in normal form by simple modifications. Let M be a pseudo deterministic MFA(k) with transition function δ. Every transition $\delta(q, b) = (p, s_1, \ldots, s_k)$ with $b \neq \varepsilon$ and $s_i \neq \diamond$ for some i, $1 \leq i \leq k$, can be replaced by transitions $\delta(q, \varepsilon) = (p', s_1, \ldots, s_k)$ and $\delta(p', b) = (p, \diamond, \ldots, \diamond)$, where p' is a new state. Then, we can replace all transitions of the form $\delta(q, \varepsilon) = (p, s_1, \ldots, s_k)$ with $s_i \neq \diamond$ for some i, $1 \leq i \leq k$, by transitions $\delta(q, \varepsilon) = (t_1, s_1, \diamond, \ldots, \diamond)$, $\delta(t_j, \varepsilon) = (t_{j+1}, \diamond, \ldots, \diamond, s_{j+1}, \diamond, \ldots, \diamond)$, $1 \leq j \leq k-2$, and $\delta(t_{k-1}, \varepsilon) = (p, \diamond, \ldots, \diamond, s_k)$, where the t_i, $1 \leq i \leq k-1$, are new states. This yields an automaton that satisfies the first condition of the normal form. In order to get rid of transitions that try to open or close memories that are already open or closed, respectively, we equip M with the capability of keeping track in its states which memories are currently open and which are closed. Now every transition that opens or closes a memory that according to the information in the state is already open or closed, respectively, can be replaced by one that does not change this memory. From these considerations, we directly conclude the following proposition.

Proposition 8. *Let $k \in \mathbb{N}$. For every $M \in$ MFA(k) there exists an $M' \in$ MFA(k) with $L(M) = L(M')$ and M' is in normal form.*

The normal form of MFA will serve the mere purpose of simplifying the technical statements of our proofs. A more important property of MFA concerns the possibility of using different memories in order to record the same part of the input, as demonstrated in Example 6. We shall now formally define this situation. Let M be a k-memory automaton, let C be a computation of M with n steps and let $1 \leq i, j \leq k$ with $i \neq j$. We say that there is an *i-j-overlap* in C if there are p, q, r, s, $1 \leq p < q < r < s < n$, such that memory i is opened in step p and closed again in step r of C and memory j is opened in step q and closed again in step s of C. A computation C is said to be *nested* if, for every i, j, $1 \leq i \leq j \leq k$, $i \neq j$, there is no i-j-overlap in C and an MFA M is *nested* if every possible computation of M is nested.

ACCEPTED MANUSCRIPT

For showing our main result, i. e., that the class of ref-regular languages coincides with the class of REGEX languages, MFA will serve as a proof tool and it will be crucial that the MFA are nested. Hence, it is our goal now to prove that every MFA can be transformed into an equivalent one that is nested. The basic idea is to keep track in the states for each two memories whether or not one has been opened before the other. In this way, it is possible to identify overlaps and we get rid of these overlaps by recording the overlapping part in a new auxiliary memory. This strategy is only applicable because, for every original memory i, at most $k-1$ auxiliary memories are needed, which also means that the number of memories is increased from k to k^2.

Lemma 9. *Let $k \in \mathbb{N}$. For every $M \in \mathrm{MFA}(k)$ there exists a nested $M' \in \mathrm{MFA}(k^2)$ with $L(M) = L(M')$.*

Proof. We assume that M is in normal form and we first transform M into an MFA(k) $M' = (Q', \Sigma, \delta', q'_0, F')$ that simulates M, but also stores in its finite state control sets $A_i \subseteq \{1, 2, \ldots, k\}$, $1 \leq i \leq k$, such that, for every i, $1 \leq i \leq k$, if $A_i = \emptyset$, then memory i is closed and if $A_i \neq \emptyset$, then memory i is open, $i \in A_i$ and A_i also contains all j, $1 \leq j \leq k$, $i \neq j$, such that memory j is currently open and it was opened after memory i was opened. In order to maintain these sets, we initially set $A_i = \emptyset$, $1 \leq i \leq k$. Whenever a memory i is closed, we set $A_i = \emptyset$ and remove i from every A_j, $1 \leq j \leq k$, $i \neq j$, and whenever a memory i is opened, we set $A_i = \{i\}$ and add i to every nonempty A_j, $1 \leq j \leq k$, $i \neq j$. Obviously, $L(M) = L(M')$ and M' is in normal form.

In the following, we shall use the information stored by the sets A_i, $1 \leq i \leq k$, in order to remove all overlaps. The general idea is that when a memory i is closed, then A_i contains all memories j such that an i-j-overlap occurs. Therefore, we can modify M' in such a way that when a memory i is closed, then, for every $j \in A_i$, $i \neq j$, we also close memory j and open an auxiliary memory instead, which now records the remaining part that was previously recorded by memory j. If we repeat this every time a memory is closed, then we eliminate all overlaps, but, since in the remaining computation the auxiliary memories can also be subject to this modification, we will need several auxiliary memories per each original memory i, $1 \leq i \leq k$. It turns out that k auxiliary memories per original memory are sufficient, which makes the above described idea applicable. If a memory i is consulted during the computation, then we have to consult all the k auxiliary memories corresponding to memory i instead and this needs to be done in the right order. We shall define this modification more formally.

We transform M' into an MFA(k^2) $M'' = (Q'', \Sigma, \delta'', q''_0, F'')$. The k^2 memories of M'' are called $\langle i, j \rangle$, $1 \leq i, j \leq k$, where, for every i, $1 \leq i \leq k$, the memories $\langle i, 1 \rangle, \langle i, 2 \rangle, \ldots, \langle i, k \rangle$ will simulate the original memory i in the way described above. Next, we define the transition function δ''. To this end, every transition $\delta'(q, b) \ni (p, s_1, s_2, \ldots, s_k)$ is transformed into the following transition:

ACCEPTED MANUSCRIPT

1. If $b \neq \varepsilon$, then, if $b \in \Sigma$, M'' changes directly from q to p by reading b from the input and if $b \in \{1, 2, \ldots, k\}$, then M'' changes from q to p by using $k-1$ new intermediate states and by consulting memories $\langle b, 1\rangle, \langle b, 2\rangle, \ldots, \langle b, k\rangle$ in this order.
2. If $b = \varepsilon$, then there exists an i, $1 \leq i \leq k$, such that $s_i \in \{\mathtt{o}, \mathtt{c}\}$ and $s_j = \diamond$, $1 \leq j \leq k$, $i \neq j$.
   (a) If $s_i = \mathtt{o}$, then M'' first changes from q to an intermediate state p' by opening all memories $\langle i, j\rangle$, $1 \leq j \leq k$, in order to make them lose their current content. Then M'' changes from p' to p by closing all memories $\langle i, j\rangle$, $2 \leq j \leq k$.
   (b) If $s_i = \mathtt{c}$, then M'' first changes from q to an intermediate state p' by closing all memories $\langle j, p_j\rangle$, where $j \in A_i$ (note that this means that memory $\langle i, p_i\rangle$ is closed as well) and $p_j = \max\{p \mid \langle j, p\rangle \text{ is open}\}$. Then M'' changes from p' to p by opening all memories $\langle j, p_j + 1\rangle$, where $j \in A_i$ with $i \neq j$.

The set Q'' contains all the states from Q' and, in addition, the finitely many intermediate states that are implicitly given by the definition of δ'' from above. Moreover, $F'' = F'$ and $q''_0 = q'_0$.

In order to conclude the definition of M'', it remains to show that in case 2b it is not possible that, for some $j \in A_i$ with $i \neq j$, $p_j = k$, since then we would try to open memory $\langle j, k+1\rangle$, which does not exist. Before we do this, we first note that at every step in every computation of M'', for every i, $1 \leq i \leq k$, there exists at most one j, $1 \leq j \leq k$, such that memory $\langle i, j\rangle$ is open. This can be concluded from the following observations. A memory $\langle i, j\rangle$ is opened in case 2a or case 2b. If case 2a applies, then $j = 1$ and if case 2b applies, then $j > 1$ and $\langle i, j-1\rangle$ is closed. By induction, this implies that, for every i, $1 \leq i \leq k$, if some of the memories $\langle i, j\rangle$, $1 \leq j \leq k$, are opened, then in such a way that first $\langle i, 1\rangle$ is opened, then $\langle i, 1\rangle$ is closed and $\langle i, 2\rangle$ is opened, then $\langle i, 2\rangle$ is closed and $\langle i, 3\rangle$ is opened and so on until, for some p, $\langle i, p\rangle$ is closed due to $s_i = \mathtt{c}$, which means that $\langle i, p+1\rangle$ is not opened.

We now assume that, for some i, $1 \leq i \leq k$, $\langle i, 1\rangle$ is currently open. If memory $\langle i, 1\rangle$ is closed, then either because M' would close memory i, which means that all memories $\langle i, j\rangle$ are closed, or because M' would close some memory j with $i \in A_j$, i. e., there is an j-i-overlap, which means that $\langle i, 1\rangle$ is closed and $\langle i, 2\rangle$ is opened. However, if the latter case applies, then, by definition, A_j is set to $\emptyset$ and i cannot be added to A_j again before all memories $\langle i, j'\rangle$ are closed due to the fact that M' would close memory i. This implies that from now on memory j cannot be responsible anymore for the situation that the currently open memory $\langle i, j'\rangle$ is closed and $\langle i, j'+1\rangle$ is opened. Hence, it can happen at most $k-1$ times that the currently open memory $\langle i, p_j\rangle$ is closed and instead $\langle i, p_j + 1\rangle$ is opened, which implies that in case 2b, always $p_j \leq k-1$.

We observe that if δ'' tries to consult a memory that is not closed, which, by definition of memory automata is not permitted, then the same problem would occur in the definition of δ', which is a contradiction. Consequently, we conclude that the automaton M'' is a valid (k^2)-memory automaton.

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

In order to show that M'' is nested, we assume to the contrary, that, for some computation $C = (c_1, c_2, \ldots, c_n)$ and i, j, i', j', $1 \leq i, j, i', j' \leq k$, $i \neq i'$, there is an $\langle i, j\rangle$-$\langle i', j'\rangle$-overlap, i. e., for some p, q, r, s with $1 \leq p < q < r < s < n$, memory $\langle i, j\rangle$ is opened by changing from c_p to c_{p+1}, closed by changing from c_r to c_{r+1} and left unchanged in between and memory $\langle i', j'\rangle$ is opened by changing from c_q to c_{q+1}, closed by changing from c_s to c_{s+1} and left unchanged in between (note that an $\langle i, j\rangle$-$\langle i, j'\rangle$-overlap is not possible, since, as observed above, for every i, $1 \leq i \leq k$, there exists at most one j, $1 \leq j \leq k$, such that memory $\langle i, j\rangle$ is open). By definition of M'', this directly implies that at configuration c_r, $i' \in A_i$, which means that memory $\langle i', j'\rangle$ is closed and $\langle i', j' + 1\rangle$ is opened instead, which is a contradiction to the assumption that there is an $\langle i, j\rangle$-$\langle i', j'\rangle$-overlap.

In order to conclude the proof, we only have to show that $L(M') = L(M'')$. The automaton M'' simulates M', i. e., every transition of M' that does not close or consult a memory can be performed by M'' in the same way, the only difference is that if M' opens memory i, then M'' opens memory $\langle i, 1\rangle$. If M' closes a memory, then this is translated into a transition of M'' that, by using an intermediate state, closes and then opens several memories and if M' consults memory i, then, again by using intermediate states, M'' consults memories $\langle i, 1\rangle, \langle i, 2\rangle, \ldots, \langle i, k\rangle$. Hence, there is a one-to-one correspondence of computations of M' and M'' and in order to prove $L(M') = L(M'')$, it is sufficient to show that at every step of corresponding computations of M' and M'' on some input w, the following property $(*)$ holds: M' is in state p with memory contents u_i, $1 \leq i \leq k$, and memory statuses r_i, $1 \leq i \leq k$, if and only if M'' is in state p with memory contents $v_{i,j}$, $1 \leq i, j \leq k$, and memory statuses $r_{i,j}$, $1 \leq i, j \leq k$, such that, for every i, $1 \leq i \leq k$, $r_i = \mathtt{C}$ implies $r_{i,j} = \mathtt{C}$, $1 \leq j \leq k$, $r_i = \mathtt{O}$ implies $r_{i,j} = \mathtt{O}$ for exactly one j, $1 \leq j \leq k$, and $u_i = v_{i,1}v_{i,2}\ldots v_{i,j}$ and $v_{i,j+1} = v_{i,j+2} = \ldots = v_{i,k} = \varepsilon$. At the beginning of a computation of M' and M'' on any input w the property $(*)$ obviously holds; thus, it only remains to show that carrying out a transition of M' and the corresponding transition of M'' maintains property $(*)$. If M' opens a memory i, then M'' opens memory $\langle i, 1\rangle$ and if M' closes memory i, then M'' closes all memories $\langle j, p_j\rangle$, where $j \in A_i$ and $p_j = \max\{p \mid \langle j, p\rangle \text{ is open}\}$ and then opens all memories $\langle j, p_j + 1\rangle$, where $j \in A_i$ with $i \neq j$. We observe that in these cases property $(*)$ is maintained. Next, we consider the case that the transition is such that a part of the input is consumed and stored in all open memories. If M' reads an input symbol $b \in \Sigma$, then M'' directly simulates M' and if M' consults memory i with content u_i, then u_i is consumed from the input and M'' consults memories $\langle i, 1\rangle, \langle i, 2\rangle, \ldots, \langle i, k\rangle$ and, as $u_i = v_{i,1}v_{i,2}\ldots v_{i,k}$, the same prefix of the input is consumed. Furthermore, M' adds b or u_i to all currently open memories $p_1, p_2, \ldots, p_\ell$. Now since property $(*)$ holds before the application of the transition, there are $q_1, q_2, \ldots, q_\ell$ with $1 \leq q_j \leq k$, $1 \leq j \leq \ell$, such that $\langle p_j, q_j\rangle$ is open and all other memories $\langle p_j, t_j\rangle$, $1 \leq t_j \leq k$, $t_j \neq q_j$, are closed. This implies that b or u_i is added to all these memories $\langle p_j, q_j\rangle$, $1 \leq j \leq \ell$. Furthermore, since $u_{p_j} = v_{p_j,1}v_{p_j,2}\ldots v_{p_j,q_j}$ and $v_{p_j,q_j+1} = v_{p_j,q_j+2} = \ldots = v_{p_j,k} = \varepsilon$, $1 \leq j \leq \ell$, holds before the application of the transition, this is

ACCEPTED MANUSCRIPT

also satisfied after the application of the transition, because b or u_i is added to memories p_j of M' and v_{p_j,q_j} of M''. This concludes the proof of the lemma. $\square$

The construction of the nested MFA in the proof of Lemma 9 does not completely preserve the normal form, since we use transitions that open or close several memories in one step. This is merely for the sake of a better presentation and every such transition could be replaced by a sequence of transitions that are consistent with the normal form (or, in other words, the nested automaton constructed in the proof of Lemma 9 can be transformed into an equivalent one in normal form without losing the nested property). Furthermore, we can note that the possible pseudo determinism of the MFA is not affected by this construction. Finally, it can be easily seen that transforming a pseudo deterministic MFA in normal form yields an MFA that is still pseudo deterministic, which implies the following.

Theorem 10. *Let $k \in \mathbb{N}$. For every $M \in \mathrm{MFA}(k)$ there exists a pseudo deterministic and nested $M' \in \mathrm{MFA}(k^2)$ in normal form with $L(M) = L(M')$.*

4. Equivalence of ref-REG, $\mathcal{L}$(MFA) and $\mathcal{L}$(REGEX)

In this section, we show that the classes of ref-regular languages, MFA languages and REGEX languages are identical. To this end, we shall first prove the equality $\mathcal{L}(\mathrm{MFA}) = \text{ref-REG}$ and then the inclusion $\mathcal{L}(\mathrm{MFA}) \subseteq \mathcal{L}(\mathrm{REGEX})$, which, together with Proposition 4, implies our main result:

Theorem 11. ref-REG $= \mathcal{L}(\mathrm{MFA}) = \mathcal{L}(\mathrm{REGEX})$.

In the following, we associate with every NFA that accepts a ref-language L an MFA(k) that accepts $\mathcal{D}(L)$, where k is the maximum number of references in each word of L. This is done as follows. Whenever the NFA reads symbols from Σ, the MFA will do the same thing, but whenever the NFA would read an occurrence of $[_{x_i}$, $]_{x_i}$ or x_i, this triggers the MFA to open, to close or to consult memory i, respectively. In this way, the MFA will accept more or less the same words as the NFA, with the only difference that the references are dereferenced "on-the-fly", i. e., the MFA accepts the dereference of the language accepted by the NFA. Moreover, this transformation can be reversed, as long as the MFA is in normal form, i. e., every ε-transition in which a memory is opened, closed or consulted is turned into one that reads the input symbol $[_{x_i}$, $]_{x_i}$ or x_i, respectively, which transforms the MFA into an NFA that accepts a regular ref-language, the dereference of which equals the language accepted by the MFA.

From a slightly different point of view, we can interpret a regular ref-language as the protocol language of an MFA accepting $\mathcal{D}(L)$, where the symbols $[_{x_i}$, $]_{x_i}$ or x_i encode the communication between finite state control and the additional storage, i. e., the symbols $[_{x_i}$ and $]_{x_i}$ mark the positions where memories are opened or closed, respectively, and the occurrences of variables x_i indicate that a factor is read by consulting a memory.

ACCEPTED MANUSCRIPT

In order to formally define this translation between MFA and NFA, we recall that if an MFA is in normal form, then every transition $\delta(q, b) \ni (p, s_1, s_2, \ldots, s_k)$ is of one of the following four types:

Σ-transition: $b \in \Sigma$ and $s_i = \diamond, 1 \leq i \leq k$.

$\mathtt{o}_i$-transition: $b = \varepsilon, s_i = \mathtt{o}$ and, for every $j, 1 \leq j \leq k, i \neq j, s_j = \diamond$.

$\mathtt{c}_i$-transition: $b = \varepsilon, s_i = \mathtt{c}$ and, for every $j, 1 \leq j \leq k, i \neq j, s_j = \diamond$.

m_i-transition: $b \in \{1, 2, \ldots, k\}$ and $s_i = \diamond, 1 \leq i \leq k$.

Let $\text{NFA}_{\text{ref}} = \{M \mid M \in \text{NFA}, L(M) \subseteq \Sigma^{[*]}, \Sigma \text{ is an alphabet}\}$ and $\text{MFA}_{\text{nf}} = \{M \mid M \in \text{MFA}, M \text{ is in normal form}\}$. We now define a mapping from NFA_{ref} to MFA_{nf}.

Definition 3. The mapping $\psi_{\mathcal{D}} : \text{NFA}_{\text{ref}} \to \text{MFA}_{\text{nf}}$ is defined in the following way. Let $N = (Q, \Sigma \cup \Gamma_k, \delta, q_0, F) \in \text{NFA}_{\text{ref}}$. Then $\psi_{\mathcal{D}}(N) := (Q, \Sigma, \delta', q_0, F) \in \text{MFA}(k)$, where δ' is defined as follows. For every transition $\delta(q, b) \ni p$ of N, we add a transition $\delta'(q, b) \ni (p, s_1, s_2, \ldots, s_k)$ to $\psi_{\mathcal{D}}(N)$, where this transition is a Σ-transition if $b \in \Sigma$, an $\mathtt{o}_i$-transition if $b = [_{x_i}$, a $\mathtt{c}_i$-transition if $b =]_{x_i}$ and an m_i-transition if $b = x_i$.

It is easy to see that $\psi_{\mathcal{D}}(N)$ is in fact an MFA in normal form. We further note that for any two $N_1, N_2 \in \text{NFA}_{\text{ref}}$ with $N_1 \neq N_2$, $\psi_{\mathcal{D}}(N_1) \neq \psi_{\mathcal{D}}(N_2)$ is implied, which means that $\psi_{\mathcal{D}}$ is injective, and since every transition of some $\text{MFA}(k)$ in normal form is of one of the four types described above, we can conclude that $\psi_{\mathcal{D}}$ is also surjective, which implies that $\psi_{\mathcal{D}}$ is a bijection. The following lemma directly implies ref-REG $= \mathcal{L}(\text{MFA})$.

Lemma 12. *Let $N \in \text{NFA}_{ref}$ and $M \in \text{MFA}_{nf}$. Then $\mathcal{D}(L(N)) = L(\psi_{\mathcal{D}}(N))$ and $L(M) = \mathcal{D}(L(\psi_{\mathcal{D}}^{-1}(M)))$.*

Proof. Let $\Sigma \cup \Gamma_k$ be the input alphabet of N. For some word $w \in L(N)$, $\psi_{\mathcal{D}}(N)$ can accept the word $\mathcal{D}(w)$ by simulating the computation of N on w and every time a transition in the computation of N is chosen that reads a symbol $[_{x_i}$, $]_{x_i}$ or x_i, the MFA $\psi_{\mathcal{D}}(N)$ instead uses the corresponding ε transition, which is an $\mathtt{o}_i$-, $\mathtt{c}_i$- or m_i-transition, respectively. In this way, every factor that is the value of a reference for variable x_i, i. e., every factor that occurs between two matching occurrences of parentheses $[_{x_i}$ and $]_{x_i}$, is recorded in memory i and, for every occurrence of a variable x_i in w, $\psi_{\mathcal{D}}(N)$ consults memory i, which currently stores the value of the reference that this occurrence of x_i refers to. In particular, this means that if an occurrence of a variable is undefined, then $\psi_{\mathcal{D}}(N)$ consults a memory that is empty. These considerations show that $\mathcal{D}(w) \in L(\psi_{\mathcal{D}}(N))$ and therefore $\mathcal{D}(L(N)) \subseteq L(\psi_{\mathcal{D}}(N))$.

Now let w be a word that is accepted by M. If we construct a word v from w by replacing every factor of w that M reads by consulting memory i by an occurrence of variable x_i and inserting an occurrence of $[_{x_i}$ and $]_{x_i}$ at every position of w that correspond to a transition that opens memory i or closes

ACCEPTED MANUSCRIPT

memory i, respectively, then the thus obtained word is accepted by $\psi_{\mathcal{D}}^{-1}(M)$ and $\mathcal{D}(v) = w$. Hence, $L(M) \subseteq \mathcal{D}(L(\psi_{\mathcal{D}}^{-1}(M)))$.

Since $\psi_{\mathcal{D}}(N) \in \mathrm{MFA}_{\mathrm{nf}}$, we can substitute M by $\psi_{\mathcal{D}}(N)$ in the statement $L(M) \subseteq \mathcal{D}(L(\psi_{\mathcal{D}}^{-1}(M)))$, which leads to $L(\psi_{\mathcal{D}}(N)) \subseteq \mathcal{D}(L(\psi_{\mathcal{D}}^{-1}(\psi_{\mathcal{D}}(N))))$. Moreover, since $\psi_{\mathcal{D}}$ is a bijection, i. e., $\psi_{\mathcal{D}}^{-1}(\psi_{\mathcal{D}}(N)) = N$, we can conclude that $\mathcal{D}(L(\psi_{\mathcal{D}}^{-1}(\psi_{\mathcal{D}}(N)))) = \mathcal{D}(L(N))$, which implies $L(\psi_{\mathcal{D}}(N)) \subseteq \mathcal{D}(L(N))$.

Analogously, since $\psi_{\mathcal{D}}^{-1}(M) \in \mathrm{NFA}_{\mathrm{ref}}$, we can substitute N by $\psi_{\mathcal{D}}^{-1}(M)$ in the statement $\mathcal{D}(L(N)) \subseteq L(\psi_{\mathcal{D}}(N))$. Thus, $\mathcal{D}(L(\psi_{\mathcal{D}}^{-1}(M))) \subseteq L(\psi_{\mathcal{D}}(\psi_{\mathcal{D}}^{-1}(M))) = L(M)$.

Consequently, we have proven all the subset relations that are necessary to conclude $\mathcal{D}(L(N)) = L(\psi_{\mathcal{D}}(N))$ and $L(M) = \mathcal{D}(L(\psi_{\mathcal{D}}^{-1}(M)))$. □

Having established the characterisation of the class of ref-regular languages by memory automata, it is now sufficient to show that every MFA language can be expressed by a REGEX in order to conclude the proof of Theorem 11. The basic idea of how this can be done is as follows. Given a memory automaton, we first transform it into an equivalent nested k-memory automaton M that is in normal form. Then we can obtain an NFA N over the alphabet $(\Sigma \cup \Gamma_k)$ with $\mathcal{D}(L(N)) = L(M)$ by applying the mapping $\psi_{\mathcal{D}}^{-1}$. The objective is now to transform N into a regular expression r with $L(N) = L(r)$ in such a way that r has the REGEX-property. Proposition 5 then ensures that $\mathcal{D}(L(r)) = L(M)$ is a REGEX language. The crucial part of this construction is to make sure that r really has the REGEX property, for which it is important that M is nested. Before we carry out this plan, we observe an obvious, but important property of nested MFA.

Proposition 13. *Let M be a nested* MFA(k) *in normal form. There is no word $w \in L(\psi_{\mathcal{D}}^{-1}(M))$ with overlapping references.*

In the following, let M be a fixed nested MFA(k) in normal form and let $N = \psi_{\mathcal{D}}^{-1}(M)$ with transition function δ. Without loss of generality, we can assume that every transition of the form $\delta(p, b) \ni q$ with $b \in \{[_{x_i},]_{x_i} \mid 1 \leq i \leq k\}$ is such that at least one accepting state is reachable from q (if not, then we could delete q and all states reachable from q along with all adjacent transitions) and, furthermore, that every state of N is reachable from the initial state (non-reachable states can be deleted as well). For every i, $1 \leq i \leq k$, let $n_i, m_i \in \mathbb{N}$ be such that $\delta(p_{i,j}, [_{x_i}) \ni q_{i,j}$, $1 \leq j \leq n_i$, are exactly the transitions of N labeled with $[_{x_i}$ and $\delta(r_{i,\ell},]_{x_i}) \ni s_{i,\ell}$, $1 \leq \ell \leq m_i$, are exactly the transitions of N labeled with $]_{x_i}$. For every i, j, ℓ, $1 \leq i \leq k$, $1 \leq j \leq n_i$, $1 \leq \ell \leq m_i$, let $R_{i,j,\ell}$ be the set of words that can take N from $q_{i,j}$ to $r_{i,\ell}$ without reading any occurrence of $]_{x_i}$. Every $v \in R_{i,j,\ell}$ occurs as a factor of some word $w \in L(N)$, enclosed by parentheses $[_{x_i}$ and $]_{x_i}$ (which are read by the transitions $\delta(p_{i,j}, [_{x_i}) \ni q_{i,j}$ and $\delta(r_{i,\ell},]_{x_i}) \ni s_{i,\ell}$, respectively). Furthermore, as pointed out by the following lemma, the factor v, considered individually, must be a ref-word (note that this particularly implies that $[_{x_i}\, v\,]_{x_i}$ is in fact a reference for variable x_i).

Lemma 14. *For every i, j, ℓ, $1 \leq i \leq k$, $1 \leq j \leq n_i$, $1 \leq \ell \leq m_i$, $R_{i,j,\ell} \subseteq \Sigma^{[*]}$.*

ACCEPTED MANUSCRIPT

Proof. We assume to the contrary that there is a $w \in R_{i,j,\ell}$ that is not a ref-word. If, for some h, $1 \leq h \leq k$, $w = v[_{x_h} v' [_{x_h} v''$ with $|v'|_{]_{x_h}} = 0$ or $w = v]_{x_h} v']_{x_h} v''$ with $|v'|_{[_{x_h}} = 0$, then $u[_{x_i} w]_{x_i} u'$ is accepted by $L(N)$, where u is a word that takes N from the initial state to $p_{i,j}$ and u' is a word that takes N from $s_{i,\ell}$ to an accepting state. This is a contradiction, since $L(N)$ is a ref-language, but $u[_{x_i} w]_{x_i} u'$ is not a ref-word. If, for some h, $1 \leq h \leq k$, $w = v[_{x_h} v']_{x_h} v''$ with $|v'|_{\{[_{x_h},]_{x_h}\}} = 0$ and $|v'|_{x_h} \geq 1$, then we obtain a contradiction in the same way. Hence, the only possible way of how the ref-word property can be violated is that, for some h, $1 \leq h \leq k$, $w = v]_{x_h} v'$ with $|v|_{\{[_{x_h},]_{x_h}\}} = 0$ or $w = v[_{x_h} v'$ with $|v'|_{\{[_{x_h},]_{x_h}\}} = 0$. Similar as before, this implies that there exists a word $u[_{x_i} v]_{x_h} v']_{x_i} u' \in L(N)$ with $|v|_{\{[_{x_h},]_{x_h}\}} = 0$ or a word $u[_{x_i} v[_{x_h} v']_{x_i} u' \in L(N)$ with $|v'|_{\{[_{x_h},]_{x_h}\}} = 0$. Hence, in $L(N)$ there is a word with overlapping references, which, by Proposition 13, is a contradiction. □

In the following, we transform N into a regular expression r and, since we want r to have the REGEX property, this has to be done in such a way that in r every $[_{x_i}$ matches a $]_{x_i}$ and such matching parentheses enclose a subexpression. The idea to achieve this is that, for each pair of transitions $\delta(p_{i,j}, [_{x_i}) \ni q_{i,j}$ and $\delta(r_{i,\ell},]_{x_i}) \ni s_{i,\ell}$, we transform the set of words that take N from $q_{i,j}$ to $r_{i,\ell}$, i. e., the set $R_{i,j,\ell}$, into a regular expression individually. For the correctness of this construction, it is crucial that M is nested and that we transform the $R_{i,j,\ell}$ into regular expressions in a specific order, which is defined next.

Let the binary relation $\prec$ over the set $\Phi = \{(i, j, \ell) \mid 1 \leq i \leq k, 1 \leq j \leq n_i, 1 \leq \ell \leq m_i\}$ be defined as follows. For every $(i_1, j_1, \ell_1), (i_2, j_2, \ell_2) \in \Phi$, we define $(i_1, j_1, \ell_1) \prec (i_2, j_2, \ell_2)$ if there is a computation of N that starts in p_{i_2,j_2} and reaches s_{i_2,ℓ_2} and takes the transitions (1) $\delta(p_{i_2,j_2}, [_{x_{i_2}}) \ni q_{i_2,j_2}$, (2) $\delta(p_{i_1,j_1}, [_{x_{i_1}}) \ni q_{i_1,j_1}$, (3) $\delta(r_{i_1,\ell_1},]_{x_{i_1}}) \ni s_{i_1,\ell_1}$ and (4) $\delta(r_{i_2,\ell_2},]_{x_{i_2}}) \ni s_{i_2,\ell_2}$ in exactly this order and no $]_{x_{i_2}}$ is read between performing transitions 1 and 4 and no $]_{x_{i_1}}$ is read between performing transitions 2 and 3.

So in other words, $(i_1, j_1, \ell_1) \prec (i_2, j_2, \ell_2)$ means that N accepts a ref-word w that contains nested references with respect to variables x_{i_1} and x_{i_2}, i. e., w contains a factor $[_{x_{i_2}} \ldots [_{x_{i_1}} \ldots]_{x_{i_1}} \ldots]_{x_{i_2}}$, where the occurrences of the parentheses are due to the transitions $(1) - (4)$. We shall next see that $\prec$ in fact semi-orders the set Φ.

Lemma 15. *If $(i_1, j_1, \ell_1) \prec (i_2, j_2, \ell_2)$, then $i_1 \neq i_2$. The relation $\prec$ is irreflexive, transitive and antisymmetric.*

Proof. For the sake of convenience, we say that an $[(i_1, j_1, \ell_1) \prec (i_2, j_2, \ell_2)]$-*computation* is any computation that witnesses $(i_1, j_1, \ell_1) \prec (i_2, j_2, \ell_2)$ according to the definition of the relation $\prec$.

We assume that $(i_1, j_1, \ell_1) \prec (i_2, j_2, \ell_2)$. By definition, this implies that there is an $[(i_1, j_1, \ell_1) \prec (i_2, j_2, \ell_2)]$-computation, which takes transition $\delta(p_{i_2,j_2}, [_{x_{i_2}}) \ni q_{i_2,j_2}$ and then transition $\delta(p_{i_1,j_1}, [_{x_{i_1}}) \ni q_{i_1,j_1}$ and reads no occurrence of $]_{x_{i_2}}$ in between and finally reaches state s_{i_2,ℓ_2} by taking $\delta(r_{i_2,\ell_2},]_{x_{i_2}}) \ni s_{i_2,\ell_2}$. This

ACCEPTED MANUSCRIPT

means that there is a word $u[_{x_{i_2}} u' [_{x_{i_1}} u'' \in L(N)$ with $|u'|_{]_{x_{i_2}}} = 0$. If $i_1 = i_2$, then $u[_{x_{i_2}} u' [_{x_{i_2}} u'' \in L(N)$ with $|u'|_{]_{x_{i_2}}} = 0$, which is a contradiction, since $L(N)$ is a ref-language. This shows the first statement of the lemma as well as that $\prec$ is irreflexive.

Next, we assume that $(i_1, j_1, \ell_1) \prec (i_2, j_2, \ell_2)$ and $(i_2, j_2, \ell_2) \prec (i_3, j_3, \ell_3)$. This implies that there is an $[(i_1, j_1, \ell_1) \prec (i_2, j_2, \ell_2)]$-computation $C_{1,2}$ and an $[(i_2, j_2, \ell_2) \prec (i_3, j_3, \ell_3)]$-computation $C_{2,3}$. We can now construct an $[(i_1, j_1, \ell_1) \prec (i_3, j_3, \ell_3)]$-computation by starting the $C_{2,3}$ computation up to the application of $\delta(p_{i_2,j_2}, [_{x_{i_2}}) \ni q_{i_2,j_2}$. Then we interrupt $C_{2,3}$ and carry out $C_{1,2}$ completely, after which we have reached state s_{i_2,ℓ_2}. Now we continue with computation $C_{2,3}$ until it terminates in state s_{i_3,ℓ_3}. This constructed computation is an $[(i_1, j_1, \ell_1) \prec (i_3, j_3, \ell_3)]$-computation; thus, $(i_1, j_1, \ell_1) \prec (i_3, j_3, \ell_3)$. Hence, $\prec$ is transitive.

In order to prove the antisymmetry of $\prec$, let $(i_1, j_1, \ell_1) \prec (i_2, j_2, \ell_2)$ and $(i_2, j_2, \ell_2) \prec (i_1, j_1, \ell_1)$. As stated above, this implies $i_1 \neq i_2$. Furthermore, there is an $[(i_1, j_1, \ell_1) \prec (i_2, j_2, \ell_2)]$-computation $C_{1,2}$ and an $[(i_2, j_2, \ell_2) \prec (i_1, j_1, \ell_1)]$-computation $C_{2,1}$. We can now combine these computations in an analogous way as done in the proof of the transitivity. More precisely, we perform the $C_{1,2}$ computation up to the application of $\delta(p_{i_1,j_1}, [_{x_{i_1}}) \ni q_{i_1,j_1}$, we interrupt $C_{1,2}$ and carry out $C_{2,1}$ completely and then continue with computation $C_{1,2}$ until it terminates in state s_{i_2,ℓ_2}. Hence, there is a word of the form $u_1 [_{x_{i_2}}\ u_2\ [_{x_{i_1}}\ u_3\ [_{x_{i_2}}\ u_4\ ]_{x_{i_2}}\ u_5]_{x_{i_1}}\ u_6\ ]_{x_{i_2}}\ u_7 \in L(N)$ with $|u_2|_{]_{x_{i_2}}} = 0$. This implies that w is either not a ref-word or it has overlapping references, which, by Proposition 13, is a contradiction. This shows that $\prec$ is antisymmetric. □

We are now ready to define a procedure that turns N into a regular expression that is based on the well-known state elimination technique (see Yu [18] for details). To this end, we need the concept of an *extended finite automaton*, which is an NFA whose transitions can be labeled by regular expressions over the input alphabet. A transition $\delta(p, r) \ni q$, where r is a regular expression, means that the automaton can change from p into q by reading any word from $L(r)$.

For every $(i, j, \ell) \in \Phi$, we define $\Delta(i, j, l) = \{(i', j', \ell') \mid (i', j', \ell') \prec (i, j, \ell)\}$ and $\Phi' = \Phi$. We iterate the following steps as long as Φ' is non-empty.

Step 1 For some $(i, j, \ell) \in \Phi'$ with $|\Delta(i, j, l)| = 0$, we obtain an automaton $K_{i,j,\ell}$ from (the current version of) N by deleting all transitions $\delta(r_{i,\ell'},]_{x_i}) \ni s_{i,\ell'}$, $1 \leq \ell' \leq m_i$, and by defining $q_{i,j}$ to be the initial state and $r_{i,\ell}$ to be the only accepting state. Then, we transform $K_{i,j,\ell}$ into a regular expression $t_{i,j,\ell}$ by applying the state elimination technique.

Step 2 For every $(i', j', \ell') \in \Phi$, we delete (i, j, ℓ) from $\Delta(i', j', \ell')$.

Step 3 We add the transition $\delta(p_{i,j}, [_{x_i} t_{i,j,\ell}]_{x_i}) \ni s_{i,\ell}$ to N.

Step 4 We delete (i, j, l) from Φ'.

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

Step 5 If, for every ℓ', $1 \leq \ell' \leq m_i$, $(i, j, \ell') \notin \Phi'$, then we delete the transition $\delta(p_{i,j}, [_{x_i}) \ni q_{i,j}$ from N.

Step 6 If, for every j', $1 \leq j' \leq n_i$, $(i, j', \ell) \notin \Phi'$, then we delete the transition $\delta(r_{i,j},]_{x_i}) \ni s_{i,\ell}$ from N.

In order to see that this procedure is well-defined, we observe that as long as $\Phi' \neq \emptyset$, there is at least one element $(i, j, \ell) \in \Phi'$ with $|\Delta(i, j, l)| = 0$, which follows directly from the transitivity and antisymmetry of $\prec$ (see Lemma 15). The next lemma states that the regular expressions $t_{i,j,\ell}$ constructed in Step 1 describe the sets $R_{i,j,\ell}$.

Lemma 16. *For every $t_{i,j,\ell}$ constructed in Step 1, $L(t_{i,j,\ell}) = R_{i,j,\ell}$.*

Proof. Since $L(K_{i,j,\ell}) = L(t_{i,j,\ell})$, it is sufficient to show $L(K_{i,j,\ell}) = R_{i,j,\ell}$. Let $w \in R_{i,j,\ell}$. By definition, w can take N from state $q_{i,j}$ to $r_{i,\ell}$ without reading any occurrence of $]_{x_i}$; thus, $w \in L(K_{i,j,\ell})$. Let $w \in L(K_{i,j,\ell})$. Then there is a computation of $K_{i,j,\ell}$ that starts in state $q_{i,j}$ and terminates in state $r_{i,\ell}$. Furthermore, in this computation, no symbol $]_{x_i}$ is read, since all transitions $\delta(r_{i,\ell'},]_{x_i}) \ni s_{i,\ell'}$, $1 \leq \ell' \leq m_i$, have been deleted. Hence, $w \in R_{i,j,\ell}$. □

The automaton obtained by this procedure, denoted by N', does not contain any transitions labeled with symbols from $\{[_{x_i},]_{x_i} \mid 1 \leq i \leq k\}$. We can now transform N' into a regular expression r by the state elimination technique. The next lemma concludes the proof of Theorem 11.

Lemma 17. *The regular expression r has the* REGEX*-property and $L(r) = L(N)$.*

Proof. We first show that the regular expression r has the REGEX-property by induction. To this end, let n be the total number of iterations of the Steps 1 to 6 and by N_i, we denote the constructed automaton after the i^{th} iteration, i.e., $N_0 = N$ and $N_n = N'$. We show that, for every i, $0 \leq i \leq n$, all transitions $\delta(p, t) \ni q$ of N_i are such that t is a regular expression with the REGEX-property. For $i = 0$, this is obviously true. We assume that for some i, $0 \leq i \leq n-1$, all transitions $\delta(p, t) \ni q$ of N_i are such that t is a regular expression with the REGEX-property and that at the beginning of the $(i+1)^{\text{th}}$ iteration, we choose $(i, j, \ell) \in \Phi'$ with $|\Delta(i, j, l)| = 0$. By definition, $K_{i,j,\ell}$ contains only transitions of N_i, thus, all transitions $\delta(p, t) \ni q$ of $K_{i,j,\ell}$ are such that t is a regular expression with the REGEX-property. Since in the state elimination technique, we combine existing labels of transitions to new regular expressions only by applying the three regular expression operations of concatenation, alternation and the star operator, $t_{i,j,\ell}$ and also $[_{x_i} t_{i,j,\ell}]_{x_i}$ are regular expressions with the REGEX-property. Therefore, all transitions $\delta(p, t) \ni q$ of N_{i+1} are such that t is a regular expression with the REGEX-property, which particularly implies that r is a regular expression with the REGEX-property.

ACCEPTED MANUSCRIPT

It remains to prove that $L(r) = L(N)$, which again can be done by induction. To this end, let the N_i, $1 \leq i \leq n$, be defined as above. We show that, for every i, $0 \leq i \leq n-1$, $L(N_i) = L(N_{i+1})$. Since $N_0 = N$ and $N_n = N'$, this implies $L(N) = L(N')$ and, since $L(N') = L(r)$, $L(r) = L(N)$ follows.

We assume that, for some i, $0 \leq i \leq n-1$, at the beginning of the $(i+1)^{\text{th}}$ iteration, we choose $(i,j,\ell) \in \Phi'$ with $|\Delta(i,j,l)| = 0$ and we construct $t_{i,j,\ell}$. Next, we note that this implies that the transitions $\delta(p_{i,j}, [_{x_i}) \ni q_{i,j}$ and $\delta(r_{i,j},]_{x_i}) \ni s_{i,\ell}$ are still existing in the automaton, since if one of these transitions has been deleted in an earlier iteration, then $(i,j,\ell) \notin \Phi'$ is implied, which is a contradiction to the assumption that $(i,j,\ell) \in \Phi'$.

Now let $w \in L(N_{i+1})$. If w is accepted by a computation that does not use the transition $\delta(p_{i,j}, [_{x_i} t_{i,j,\ell}]_{x_i}) \ni s_{i,\ell}$, then N_i can accept w in the same way. If, on the other hand w is accepted by a computation that uses the transition $\delta(p_{i,j}, [_{x_i} t_{i,j,\ell}]_{x_i}) \ni s_{i,\ell}$ by reading a word $[_{x_i} v]_{x_i} \in L([_{x_i} t_{i,j,\ell}]_{x_i})$, then N_i can accept w by applying the same computation, but instead of moving from $p_{i,j}$ to $s_{i,\ell}$ by taking the transition $\delta(p_{i,j}, [_{x_i} t_{i,j,\ell}]_{x_i}) \ni s_{i,\ell}$, we take the transition $\delta(p_{i,j}, [_{x_i}) \ni q_{i,j}$, then read the word v and finally take transition $\delta(r_{i,j},]_{x_i}) \ni s_{i,\ell}$ in order to reach state $s_{i,\ell}$. This is possible since, by Lemma 16, $L(t_{i,j,\ell}) = R_{i,j,\ell}$ and, as stated above, both transitions $\delta(p_{i,j}, [_{x_i}) \ni q_{i,j}$ and $\delta(r_{i,j},]_{x_i}) \ni s_{i,\ell}$ are present in N_i. Hence, $L(N_{i+1}) \subseteq L(N_i)$.

If none of the transitions $\delta(p_{i,j}, [_{x_i}) \ni q_{i,j}$ and $\delta(r_{i,j},]_{x_i}) \ni s_{i,\ell}$ are deleted in Steps 5 and 6, then every word accepted by N_i can still be accepted by N_{i+1} in the same way. So we assume that $\delta(p_{i,j}, [_{x_i}) \ni q_{i,j}$ or $\delta(r_{i,j},]_{x_i}) \ni s_{i,\ell}$ is deleted in Step 5 or 6, respectively. Let $w \in L(N_i)$. If w is accepted by a computation that does not use the transitions $\delta(p_{i,j}, [_{x_i}) \ni q_{i,j}$ and $\delta(r_{i,j},]_{x_i}) \ni s_{i,\ell}$, then again it can be accepted by N_{i+1} in the same way. In the following, we assume that w is accepted by N_i by a computation that uses the transition $\delta(p_{i,j}, [_{x_i}) \ni q_{i,j}$, but this transition is deleted in Step 5, i. e., it is not present in N_{i+1}. Since w is a ref-word, after using transition $\delta(p_{i,j}, [_{x_i}) \ni q_{i,j}$, at some point a transition $\delta(r_{i,\ell'},]_{x_i}) \ni s_{i,\ell'}$ has to be used. If $\ell' = \ell$, then N_{i+1} can accept w by using transition $\delta(p_{i,j}, [_{x_i} t_{i,j,\ell}]_{x_i}) \ni s_{i,\ell}$ instead of the transitions $\delta(p_{i,j}, [_{x_i}) \ni q_{i,j}$, $\delta(r_{i,\ell},]_{x_i}) \ni s_{i,\ell}$ and the computation in between. On the other hand, if $\ell \neq \ell'$, then, since $\delta(p_{i,j}, [_{x_i}) \ni q_{i,j}$ is deleted in Step 5, we can conclude that $(i,j,\ell') \notin \Phi'$. Hence, in an earlier iteration, (i,j,ℓ') has been removed from Φ' in Step 4, which implies that in this earlier iteration a transition $\delta(p_{i,j}, [_{x_i} t_{i,j,\ell'}]_{x_i}) \ni s_{i,\ell'}$ has been introduced. Thus, N_{i+1} can accept w by using this transition $\delta(p_{i,j}, [_{x_i} t_{i,j,\ell'}]_{x_i}) \ni s_{i,\ell'}$ instead of the transitions $\delta(p_{i,j}, [_{x_i}) \ni q_{i,j}$, $\delta(r_{i,\ell'}, [_{x_i}) \ni s_{i,\ell'}$ and the computation in between. This shows that $L(N_i) \subseteq L(N_{i+1})$; thus, $L(N_i) = L(N_{i+1})$, which concludes the proof. □

We shall conclude this section by a brief discussion of Theorem 11. Our proof shows that every regular expression r that describes a ref-language can be transformed in a regular expression r' with the REGEX-property that may describe a different regular language, but $\mathcal{D}(L(r)) = \mathcal{D}(L(r'))$ is satisfied. More precisely, this can be done by first transforming r into an NFA N accepting $L(r)$, then obtaining the memory automaton $\psi_{\mathcal{D}}(N)$, which can then be transformed

ACCEPTED MANUSCRIPT

into a nested MFA M, and, finally, M can be translated into a regular expression with the REGEX-property as demonstrated above. In this regard, REGEX (or regular expression with the REGEX-property) can be interpreted as a normal form of regular expressions that describe ref-languages.

5. DMFA Languages

We now take a closer look at the class of languages accepted by DMFA. Deterministic versions of machine models are of special interest, since they usually provide fast algorithms for solving the *membership problem* for a language class, i. e., checking whether a given word is a member of a given language. Nondeterministic devices, on the other hand, have first to be determinised, if possible, or their nondeterminism has to be handled by backtracking. For REGEX languages we observe the special situation that no polynomial time algorithms are known for solving the membership problem (this problem is NP-complete) and therefore identifying and investigating subclasses with a polynomial time membership problem is a worthwhile research task.

As an example for a language in $\mathcal{L}(\text{DMFA})$, we consider $\{w\mathtt{c}w \mid w \in \{\mathtt{a},\mathtt{b}\}^*\}$, which can be accepted by a DMFA(1) as follows. We initially open the memory and start reading any word over $\{\mathtt{a},\mathtt{b}\}$. As soon as an occurrence of $\mathtt{c}$ is read, we change into a different state and close the memory in the same transition. Now, by consulting the memory, we change into an accepting state. However, as shown next, $L_{\text{copy}} = \{ww \mid w \in \{\mathtt{a},\mathtt{b}\}^*\} \notin \mathcal{L}(\text{DMFA})$.

Theorem 18. $\mathcal{L}(\text{DMFA}) \subset \mathcal{L}(\text{MFA})$.

Proof. We prove the statement of the theorem by showing $L_{\text{copy}} = \{ww \mid w \in \{\mathtt{a},\mathtt{b}\}^*\} \notin \mathcal{L}(\text{DMFA})$. To this end, we assume to the contrary, that there exists a deterministic k-memory automaton M with $L(M) = L_{\text{copy}}$. Since L_{copy} is not a regular language, we can assume that there exists a word $w \in L_{\text{copy}}$, such that in the computation $c_1, c_2, \dots, c_n$ of M on w, M uses one of its memories, i. e., there exists an i, $1 \leq i \leq k$, and a j, $1 \leq j \leq n-1$, such that $c_j = (p, u_i\, v, (u_1, r_1), \dots, (u_k, r_k))$, $c_{j+1} = (p', v, (u'_1, r'_1), \dots, (u'_k, r'_k))$, $u_i \neq \varepsilon$ and $\delta(p, i) = (p', s_1, \dots, s_k)$ with $r_i = \mathtt{C}$ and $(r'_1, \dots, r'_k) = (r_1, \dots, r_k) \oplus (s_1, \dots, s_k)$. We assume that j is the smallest such number and we can further assume that $j \geq 2$, since if $j = 1$, then $u_i = \varepsilon$. Now let $w = v'\, u_i\, v$ and let $\mathtt{a}$ be the first symbol of u_i. We consider now a computation of M on the word $w' = (v'\,\mathtt{b})^2$. In this computation, since M is deterministic, the configuration $(p, \mathtt{b}\, v'\, \mathtt{b}, (u_1, r_1), \dots, (u_k, r_k))$ is reached, where the only possible transition is $\delta(p, i) = (p', s_1, \dots, s_k)$. Since u_i is not a prefix of the remaining word $\mathtt{b}\, v'\, \mathtt{b}$, this transition is not applicable, which means that M does not accept w'. This is a contradiction, since $w' \in L_{\text{copy}}$. □

Next, we note that the membership problem of this language class can be solved in linear time by simply running the DMFA.

ACCEPTED MANUSCRIPT

Theorem 19. *For a given* DMFA M *and a word* $w \in \Sigma^*$*, we can decide in time* $\mathrm{O}(|w|)$ *whether or not* $w \in L(M)$*.*

This contrasts the situation that for other prominent subclasses of REGEX languages (e. g., the ones investigated in [16, 1, 3, 6]) the membership problem is usually NP-complete. Another exception are REGEX with a bounded number of backreferences (see [16]).

It is known that REGEX languages are closed under union, but not under intersection or complementation (see [4, 7]). For the subclass $\mathcal{L}$(DMFA), we observe a slightly different situation:

Theorem 20. $\mathcal{L}$(DMFA) *is not closed under union or intersection.*

Proof. We define $L_1 = \{\mathtt{a}^n\mathtt{b}\mathtt{a}^n \mid n \in \mathbb{N}\}$, $L_2 = \{\mathtt{a}^m\mathtt{b}\mathtt{a}^n\mathtt{b}\mathtt{a}^\ell \mid m, n, \ell \in \mathbb{N}_0\}$ and we note that these languages are both in $\mathcal{L}$(DMFA). We shall now show that $\mathcal{L}$(DMFA) is not closed under union by proving that $L_1 \cup L_2 \notin \mathcal{L}$(DMFA). To this end, we assume to the contrary that there is a DMFA(k) M with $L(M) = L_1 \cup L_2$. We note that there must exists a word $w \in L_1$, such that for the accepting computation $C = (c_1, c_2, \ldots, c_n)$ of M on w, there is a j, $1 < j \leq n-1$, such that, for some i, $1 \leq i \leq k$, configuration c_j changes into configuration c_{j+1} by an application of a transition $\delta(q, i) = (p, s_1, \ldots, s_k)$ and the content u_i of memory i is non-empty in configuration c_j. This is due to the fact that if all words $w \in L_1$ are accepted by a computation that does not consult a memory, then, for long enough w, M enters a loop which yields the acceptance of a word with an arbitrary number of occurrences of symbol $\mathtt{b}$ or a word of the form $\mathtt{a}^n\mathtt{b}\mathtt{a}^{n'}$ with $n \neq n'$. Since such words are neither in L_1 nor in L_2, this is a contradiction. Let $\widehat{j}$, $1 < \widehat{j} \leq n-1$, be the smallest number that satisfies that there is an i, $1 \leq i \leq k$, such that configuration c_j changes into configuration c_{j+1} by an application of a transition $\delta(q, i) = (p, s_1, \ldots, s_k)$. We now consider the factorisation $w = vu_iv'$, where $c_{\widehat{j}} = (q, u_iv', (r_1, u_1), \ldots, (r_k, u_k))$. Next, we note that $|u_i|_{\mathtt{b}} = 0$, since otherwise $|w|_{\mathtt{b}} \geq 2$, which is a contradiction to $w \in L_1$. If $v = \mathtt{a}^m$, then $v' = \mathtt{a}^n\mathtt{b}\mathtt{a}^{(m+n+|u_i|)}$. We consider now the word $w' = v\mathtt{b}v'$, which is in L_2. Since M is deterministic, on input w', it reaches configuration $(q, \mathtt{b}v', (r_1, u_1), \ldots, (r_k, u_k))$. However, since the only applicable transition is $\delta(q, i) = (p, s_1, \ldots, s_k)$ and u_i is not a prefix of $\mathtt{b}v'$, the computation is not accepting. This is a contradiction, since $w' \in L_2$.

If v is not of the form $\mathtt{a}^m$, then $v = \mathtt{a}^m\mathtt{b}\mathtt{a}^n$ and $v' = \mathtt{a}^{(m-n-|u_i|)}$. Similar as before, on input w', M reaches configuration $(q, \mathtt{b}v', (r_1, u_1), \ldots, (r_k, u_k))$ and we can obtain a contradiction in the same way. This concludes the proof of the non-closure of $\mathcal{L}$(DMFA) under union.

In [7], Carle and Narendran show that the intersection of the languages $L'_1 = \{\mathtt{a}^n\mathtt{b}\mathtt{a}^{n+1}\mathtt{b}\mathtt{a}^{nk} \mid n, k \in \mathbb{N}\}$ and $L'_2 = \{\mathtt{a}^n\mathtt{b}\mathtt{a}^{n+1}\mathtt{b}\mathtt{a}^{(n+1)k} \mid n, k \in \mathbb{N}\}$ is not a REGEX language. Hence, $L'_1 \cap L'_2$ is also not a DMFA language. In order to conclude that $\mathcal{L}$(DMFA) is not closed under intersection, we need to show that $L'_1 \in \mathcal{L}$(DMFA) and $L'_2 \in \mathcal{L}$(DMFA). The language L'_1 can be accepted by a DMFA(1) by first opening the memory and in the same transition reading

ACCEPTED MANUSCRIPT

a single occurrence of a (which is stored in the memory). Then we can read an arbitrary number of occurrences of symbol a and as soon as a b is read the memory is closed, such that it now contains $\mathtt{a}^n$. Next, we read $\mathtt{a}^n$ by consulting the memory and then reading a single occurrence of a. Finally, we read an occurrence of b, we consult the memory and change into an accepting state in which we can consult the memory arbitrarily often by a loop.

In order to accept the language L'_2, we need a DMFA(2). In the same way as done for language L'_1, we first read $\mathtt{a}^n\mathtt{b}$ and store $\mathtt{a}^n$ in memory 1, which is closed. Then we read a and open memory 2 at the same time, and then consult memory 1, which means that $\mathtt{a}^{n+1}$ is stored in memory 2. Similar as before, we read next an occurrence of b, we consult memory 2 and change into an accepting state in which we can consult memory 2 arbitrarily often by a loop. □

In [4], which marks the beginning of the formal investigation of REGEX, Câmpeanu et al. ask whether REGEX languages are closed under intersection with regular languages, which has been answered in the positive by Câmpeanu and Santean in [5]. In the following, we present a much simpler alternative proof for this result, which demonstrates that MFA are a convenient tool to handle REGEX languages and which also shows the closure of DMFA languages under intersection with regular languages.

The closure under intersection with regular languages follows from the fact that we can simulate an MFA(k) M and a DFA N in parallel by an MFA M'. The difficulty that we encounter is that if N is currently in a state q and M consults memory i and therefore consumes a whole prefix u_i from the input, then we do not know in which state N needs to change. Hence, for every memory i and every state q of N, we need to store the state that is reached when N starts in q and reads the current content of memory i. When a new word is stored in memory i, then we have to update this information, which is possible, since memory i is filled by either consuming single symbols (in this case, the transitions function of N must be consulted) or by consuming a prefix u_j, which is the content of some memory j and for which the state that is reached by N is correctly stored.

Theorem 21. *$\mathcal{L}$(MFA) and $\mathcal{L}$(DMFA) are closed under intersection with regular languages.*

Proof. Let $M = (Q_M, \Sigma, \delta_M, q_{0,M}, F_M) \in$ MFA(k) be pseudo deterministic and in normal form and let $N = (Q_N, \Sigma, \delta_N, q_{0,N}, F_N)$ be a DFA with $Q_N = (q_1, q_2, \dots q_n)$. Furthermore, we assume that the finite state control of M contains the information which memories are currently open and which are closed. We construct an MFA(k) $M' = (Q_{M'}, \Sigma, \delta_{M'}, q_{0,M'}, F_{M'})$ with $L(M') = L(M) \cap L(N)$.

Let $Q_{M'}$ be the set of all elements $\langle q, q', P_1, \dots, P_k\rangle$, where $q \in Q_M$, $q' \in Q_N$, and, for every i, $1 \leq i \leq k$, $P_i \in \{1, 2, \dots, n\}^n$. For every i, $1 \leq i \leq k$, and j, $1 \leq j \leq n$, $P_i[j]$ denotes the jth element of tuple P_i. The idea of the states $\langle q, q', P_1, \dots, P_k\rangle$ is that q and q' represent the current states of M and N,

ACCEPTED MANUSCRIPT

respectively, and, for every i, $1 \leq i \leq k$, ℓ, $1 \leq \ell \leq n$, $q_{P_i[\ell]}$ is the state that is reached when N starts in q_ℓ and reads the content of memory i. Hence, if M' consults memory i in state $\langle q, q_\ell, P_1, \ldots, P_k\rangle$, then the component corresponding to the state of M is updated according to δ_M and the component corresponding to the state of N is set to $q_{P_i[\ell]}$, since this is the state N would reach if it reads the word that is currently stored in memory i.

We now formally define the transition function of M'. To this end, we first recall that M is in normal form, which means that, for every transition $\delta_M(q, y) \ni (p, s_1, \ldots, s_k)$, if $s_i \neq \diamond$ for some i, $1 \leq i \leq k$, then $y = \varepsilon$ and $s_j = \diamond$, for all j with $1 \leq j \leq k$ and $i \neq j$. For every transition $\delta_M(q, y) = (p, s_1, \ldots, s_k)$, for every ℓ, $1 \leq \ell \leq n$, and for every $P_i \in \{1, 2, \ldots, n\}^n$, $1 \leq i \leq k$, we define

$$\delta_{M'}(\langle q, q_\ell, P_1, \ldots, P_k\rangle, y) \ni (\langle p, q_{\ell'}, P'_1, \ldots, P'_k\rangle, s_1, \ldots, s_k)$$

where ℓ' and the P'_i, $1 \leq i \leq k$, are defined as follows.

- If $y = \varepsilon$, then $\ell' = \ell$ and, furthermore,
  - if $\mathsf{o} \notin \{s_1, s_2, \ldots, s_k\}$, then $P'_i = P_i$, $1 \leq i \leq k$, and
  - if, for some i, $1 \leq i \leq k$, $s_i = \mathsf{o}$, then $P'_i = (1, 2, \ldots, n)$ and $P'_{i'} = P_{i'}$, $1 \leq i' \leq k$, $i \neq i'$.
- If $y \in \Sigma$, then $q_{\ell'} = \delta_N(q_\ell, y)$ and, for every i, $1 \leq i \leq k$, j, $1 \leq j \leq n$, $P'_i[j] = m$, where $q_m = \delta_N(q_{P_i[j]}, y)$, if memory i is open and $P'_i[j] = P_i[j]$ if it is closed.
- If $y \in \{1, 2, \ldots, k\}$, then $\ell' = P_y[\ell]$ and, for every i, $1 \leq i \leq k$, j, $1 \leq j \leq n$, $P'_i[j] = P_y[P_i[j]]$ if memory i is open and $P'_i[j] = P_i[j]$ if it is closed.

The initial state of M' is $q_{0,M'} = \langle q_{0,M}, q_{0,N}, P_1, P_2, \ldots, P_k\rangle$, where, for every i, $1 \leq i \leq k$, $P_i = (1, 2, \ldots, n)$, and the set of final states is $F_{M'} = \{\langle q, q', P_1, \ldots, P_k\rangle \mid q \in F_M, q' \in F_N, P_i \in \{1, 2, \ldots, n\}^n, 1 \leq i \leq k\}$.

Claim 1: Every configuration $(\langle q, q_\ell, P_1, \ldots, P_k\rangle, w, (u_1, r_1), \ldots, (u_k, r_k))$ reached by M' on some input satisfies the following condition $(*)$: for every i, $1 \leq i \leq k$, and j, $1 \leq j \leq n$, $\delta_N(q_j, u_i) = q_{P_i[j]}$.

Proof of Claim 1: We prove this claim by induction and first note that the initial configuration of M' satisfies $(*)$. We now have to show that $(*)$ is maintained by changing into a new configuration $(\langle q', q'_\ell, P'_1, \ldots, P'_k\rangle, w', (u'_1, r'_1), \ldots, (u'_k, r'_k))$ by some transition.

An ε-transition that does not open any memory does not changed the P_i, $1 \leq i \leq k$, and an ε-transition that opens memory i does not change the $P_{i'}$, $1 \leq i' \leq k$, $i \neq i'$, and sets $P'_i = (1, 2, \ldots, n)$, which means that, since $u'_i = \varepsilon$, $\delta_N(q_j, u'_i) = P'_i[j]$, $1 \leq j \leq n$. Hence, ε-transitions maintain $(*)$. If we apply a transition that reads $b \in \Sigma$ from the input, then we do not change the P_i, $1 \leq i \leq k$, where memory i is closed, which means that for these P_i condition $(*)$ is satisfied, since the memory contents do not change. For the P_i, $1 \leq i \leq k$, where memory i is open, we set $P'_i[j] = m$, where $q_m = \delta_N(q_{P_i[j]}, b)$, $1 \leq j \leq n$,

ACCEPTED MANUSCRIPT

which, since $\delta_N(q_j, u_i) = q_{P_i[j]}$, implies $\delta_N(q_j, u'_i) = \delta_N(q_j, u_i b) = \delta_N(q_{P_i[j]}, b) = q_{P'_i[j]}$. Hence, for these P_i condition $(*)$ is satisfied as well. The last case to consider is when a memory i' is consulted. Again, we do not change the P_i, $1 \leq i \leq k$, where memory i is closed. For the P_i, $1 \leq i \leq k$, where memory i is open, we set $P'_i[j] = P_{i'}[P_i[j]]$, which, since $\delta_N(q_j, u_i) = q_{P_i[j]}$, implies $\delta_N(q_j, u_i u_{i'}) = \delta_N(q_{P_i[j]}, u_{i'}) = q_{P_{i'}[P_i[j]]} = q_{P'_i[j]}$. Consequently, all transitions that consult memories also maintain condition $(*)$. This concludes the proof of Claim 1.

If M' can reach a configuration $(\langle q, q_\ell, P_1, \ldots, P_k\rangle, v, (u_1, r_1), \ldots, (u_k, r_k))$ on some input w, then M can reach the configuration $(q, v, (u_1, r_1), \ldots, (u_k, r_k))$ on input w and N can reach the configuration (q_ℓ, v) on input w. The first part of this statement follows easily from the definition of $\delta_{M'}$ and the second statement can be concluded with moderate effort from Claim 1. Similarly, we can conclude that if, on some input w, M can reach the configuration $(q, v, (u_1, r_1), \ldots, (u_k, r_k))$ and N can reach the configuration (q_ℓ, v), then there are $P_i \in \{1, 2, \ldots, n\}^n$, $1 \leq i \leq k$, such that M' can reach the configuration $(\langle q, q_\ell, P_1, \ldots, P_k\rangle, v, (u_1, r_1), \ldots, (u_k, r_k))$ on input w.

Since $F_{M'} = \{\langle q, q', P_1, \ldots, P_k\rangle \mid q \in F_M, q' \in F_N\}$, this directly implies $L(M') = L(M) \cap L(N)$, which concludes the proof of the claim that $\mathcal{L}(\text{MFA})$ is closed under intersection with regular languages.

If M is deterministic (and, therefore, not in normal form, since it does not contain any ε transitions), we can apply a similar construction. We only have to make sure that whenever a memory i is opened in some transition, then the corresponding P_i is set to $(1, 2, \ldots, n)$. Let us assume that M is deterministic and M' is the MFA obtained from M and N in the above described way (adapted to the special case where M is deterministic). If M' is not deterministic, then for some state $t = \langle q, q_\ell, P_1, \ldots, P_k\rangle \in Q_{M'}$ and for some $b \in \Sigma$, $|\bigcup_{i=1}^{k} \delta_{M'}(t, i)| + |\delta_{M'}(t, b)| > 1$. Since N is deterministic, this directly implies that $|\bigcup_{i=1}^{k} \delta_M(q, i)| + |\delta_M(q, b)| > 1$, which leads to the contradiction that M is not deterministic. Thus, if M is deterministic, then so is M', which concludes the proof of the statement that $\mathcal{L}(\text{DMFA})$ is closed under intersection with regular languages. $\square$

In [4], it is shown that $\{\mathtt{a}^n \mid m > 1 \text{ is not prime}\}$ is a REGEX language, but its complement cannot be expressed by any REGEX. This shows that the class of MFA languages is not closed under complementation. Unfortunately, the class $\mathcal{L}(\text{DMFA})$ is also not closed under complementation.[2] This is surprising, since $\mathcal{L}(\text{DMFA})$ is characterised by a deterministic automaton model; thus, one might expect that it is sufficient to interchange the accepting and non-accepting states in order to obtain an automaton that accepts the complement. However, this is only possible if the automaton model is complete, which is not the case for DMFA; in fact, the definition of determinism for DMFA requires

[2]This corrects an error in the preliminary conference version [17] of this paper, in which it is stated that $\mathcal{L}(\text{DMFA})$ is closed under complementation.

ACCEPTED MANUSCRIPT

that no consulting transitions are defined if there is a transition that reads a single symbol and, furthermore, if a memory can be consulted, then it is not allowed that a transition is defined that reads a symbol. That this inherent incompleteness of DMFA is problematic for accepting complements of DMFA languages by a DMFA becomes evident, if we try to find a DMFA for the complement of $\{w\mathtt{c}w \mid w \in \{\mathtt{a}, \mathtt{b}\}^*\}$.

Theorem 22. $\mathcal{L}$(DMFA) *is not closed under complementation.*

Proof. In this proof, we denote the complement of a language $L \subseteq \Sigma^*$ by $\overline{L}$. We first note that if $\mathcal{L}$(DMFA) is closed under complementation, then, for every $L \in \mathcal{L}$(DMFA) and $R \in$ REG, $\overline{\overline{L} \cap \overline{R}} \in \mathcal{L}$(DMFA), since $\mathcal{L}$(DMFA) is closed under intersection with regular languages. As $\overline{\overline{L} \cap \overline{R}} = L \cup R$, this means that $\mathcal{L}$(DMFA) is closed under union with regular languages. However, in the proof of Theorem 20, it is shown that $L_1 = \{\mathtt{a}^n\mathtt{b}\mathtt{a}^n \mid n \in \mathbb{N}\}$ and $L_2 = \{\mathtt{a}^m\mathtt{b}\mathtt{a}^n\mathtt{b}\mathtt{a}^\ell \mid m, n, \ell \in \mathbb{N}_0\}$ are both DMFA languages, but their union is not a DMFA language and since L_2 is a regular language, this proves that $\mathcal{L}$(DMFA) is not closed under union with regular languages. Consequently, $\mathcal{L}$(DMFA) is not closed under complementation. □

6. Conclusions

Any REGEX language is composed of two fundamental parts: the regular operators of concatenation, alternation and Kleene star on the one hand and reduplications on the other. If such a language is represented by a REGEX, it can be difficult to distinguish between these two parts, since the backreferences (which define reduplications) and the regular operators are intertwined. It seems that this is what makes REGEX (and therefore the corresponding language class) difficult to analyse. In [6], Câmpeanu and Yu try to separate these two aspects by introducing pattern expressions, which describe an important, but proper subclass of $\mathcal{L}$(REGEX) (see [16]).

In this paper, we show that all REGEX languages (and only these) can be represented by a regular language L (more precisely, a regular ref-language) and the dereference function, which turns L into the REGEX language by carrying out reduplications. Consequently, we obtain a clear separation between the regular part and the reduplications part of a REGEX language. This REGEX-independent characterisation of $\mathcal{L}$(REGEX) is accompanied by a second characterisation in terms of an automaton model (i. e., the MFA).

For working with a class of formal languages, an automaton model is usually a very useful tool and we hope that this is also true with respect to REGEX languages and memory automata. However, we wish to emphasise that the characterisation in terms of ref-REG is particularly versatile, since it allows any characterisation of regular languages in order to define the regular ref-languages. Classical finite automata and regular expressions is what we have used in Section 4, but any of the many ways to define the class of regular

ACCEPTED MANUSCRIPT

languages could be used and will lead to different individual characterisations of the class of REGEX languages.

Finally, we point out that the general concept of equipping a language class (in our case, the regular languages) with references and then consider the image of this language class under the dereference function $\mathcal{D}$ can be applied to any language class. In this regard, we have presented a general way of how reduplications (an inherently non-context-free feature that often cannot be handled by typical language descriptors in an easy way) can be added to any kind of language class $\mathfrak{L}$ (more precisely, in our definitions we only have to substitute the class of regular languages by $\mathfrak{L}$). For example, ref-CF is then the class of all languages that are images under $\mathcal{D}$ of some context-free ref-language. While we have a clear motivation for investigating ref-REG (i. e., they coincide with the practically applied REGEX languages) it is a question for which language classes $\mathfrak{L}$ the class ref-$\mathfrak{L}$ is a worthwhile object of research. Considering once more ref-CF, it seems reasonable that an automaton model for ref-CF can be obtained by equipping pushdown automata with a constant number of memories, just as we have equipped finite automata with memories in order to obtain MFA. Consequently, we hope that the techniques developed in this paper may lead to a more general framework towards the language theoretical investigation of the power of reduplication.

Acknowledgements

The author wishes to thank Andreas Malcher for pointing out that it is wrongly stated in [17] that $\mathcal{L}$(DMFA) is closed under complementation. This error has now been corrected.

[1] J. Albert and L. Wegner. Languages with homomorphic replacements. *Theoretical Computer Science*, 16:291–305, 1981.

[2] D. Angluin. Finding patterns common to a set of strings. In *Proc. 11th Annual ACM Symposium on Theory of Computing*, pages 130–141, 1979.

[3] H. Bordihn, J. Dassow, and M. Holzer. Extending regular expressions with homomorphic replacement. *RAIRO Theoretical Informatics and Applications*, 44:229–255, 2010.

[4] C. Câmpeanu, K. Salomaa, and S. Yu. A formal study of practical regular expressions. *Int. Journal of Foundations of Computer Science*, 14:1007–1018, 2003.

[5] C. Câmpeanu and N. Santean. On the intersection of regex languages with regular languages. *Theoretical Computer Science*, 410:2336–2344, 2009.

[6] C. Câmpeanu and S. Yu. Pattern expressions and pattern automata. *Information Processing Letters*, 92:267–274, 2004.

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

[7] B. Carle and P. Narendran. On extended regular expressions. In *Proc. LATA 2009*, volume 5457 of *LNCS*, pages 279–289, 2009.

[8] J. Dassow and G. Păun. *Regulated Rewriting in Formal Language Theory.* Springer-Verlag Berlin, 1989.

[9] J. Dassow, G. Păun, and A. Salomaa. Grammars with controlled derivations. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 2, pages 101–154. Springer, 1997.

[10] R. W. Floyd. On the nonexistence of a phrase structure grammar for algol 60. *Communications of the ACM*, 5:483–484, 1962.

[11] D. D. Freydenberger. Extended regular expressions: Succinctness and decidability. *Theory of Computing Systems*, 53:159–193, 2013.

[12] J. E. F. Friedl. *Mastering Regular Expressions.* O'Reilly, Sebastopol, CA, third edition, 2006.

[13] L. Kallmeyer. *Parsing Beyond Context-Free Grammars.* Springer-Verlag, 2010.

[14] L. Kari, G. Rozenberg, and A. Salomaa. L systems. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, chapter 5, pages 253–328. Springer, 1997.

[15] G. Della Penna, B. Intrigila, E. Tronci, and M. Venturini Zilli. Synchronized regular expressions. *Acta Informatica*, 39:31–70, 2003.

[16] M. L. Schmid. Inside the class of REGEX languages. *International Journal of Foundations of Computer Science*, 24:1117–1134, 2013.

[17] M. L. Schmid. Characterising REGEX languages by regular languages equipped with factor-referencing. In *Proc. 18th International Conference on Developments in Language Theory, DLT 2014*, volume 8633 of *Lecture Notes in Computer Science*, pages 142–153, 2014.

[18] S. Yu. Regular languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, chapter 2, pages 41–110. Springer, 1997.

ACCEPTED MANUSCRIPT