Information and Computation 249 (2016) 1–17

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

# Characterising REGEX languages by regular languages equipped with factor-referencing [☆]

Markus L. Schmid

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

ARTICLE INFO

*Article history:*
Received 27 November 2014
Received in revised form 7 January 2016
Available online 20 February 2016

*Keywords:*
REGEX languages
Regular languages
Memory automata

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).

## 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 form $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 1\}$ and *crossed agreements*, as modelled 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], synchronised 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

[☆] A preliminary version [17] of this paper was presented at the conference DLT 2014.
*E-mail address:* MSchmid@uni-trier.de.

http://dx.doi.org/10.1016/j.ic.2016.02.003
0890-5401/© 2016 Elsevier Inc. All rights reserved.

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 renders 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 extent, these conceptional problems inevitably creep into the theoretical literature as well.

Regular expressions often serve as a 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.,

$$w = \mathtt{a}\,\underbrace{\mathtt{b}\,\mathtt{a}\,\overbrace{\mathtt{c}}^{y}}_{x}\overbrace{\,\mathtt{b}\,\mathtt{c}\,}^{}\overbrace{x\,\mathtt{c}\,\mathtt{b}}^{z}\,z\,y\,\mathtt{a}\,,$$

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 $\mathtt{abacbcbaccbzcba}$ 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 $\varepsilon$ 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 $\Sigma$ 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* $\Sigma$ (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 $z \notin \{[_{x_i}, ]_{x_i}, x_i\}$. 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