RAIRO-Theor. Inf. Appl. 44 (2010) 229–255
DOI: 10.1051/ita/2010013

Available online at:
www.rairo-ita.org

EXTENDING REGULAR EXPRESSIONS WITH HOMOMORPHIC REPLACEMENT *, **

Henning Bordihn[1], Jürgen Dassow[2] and Markus Holzer[3]

Abstract. We define H- and EH-expressions as extensions of regular expressions by adding homomorphic and iterated homomorphic replacement as new operations, resp. The definition is analogous to the extension given by Gruska in order to characterize context-free languages. We compare the families of languages obtained by these extensions with the families of regular, linear context-free, context-free, and EDT0L languages. Moreover, relations to language families based on patterns, multi-patterns, pattern expressions, H-systems and uniform substitutions are also investigated. Furthermore, we present their closure properties with respect to TRIO operations and discuss the decidability status and complexity of fixed and general membership, emptiness, and the equivalence problem.

Mathematics Subject Classification. 03D40, 68Q42, 68Q45.

Keywords and phrases. Regular languages, homomorphic replacement, computational complexity.

* *Part of the work was done while the first author was at Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg, Postfach 4120, 39016 Magdeburg, Germany.*

** *Part of the work was done while the third author was at Département d'I.R.O., Université de Montréal, C.P. 6128, succ. Centre-Ville, Montréal (Québec), H3C 3J7 Canada and at Institut für Informatik, Technische Universität München, Boltzmannstraße 3, 85748 Garching bei München, Germany. He was supported in part by the Deutsche Forschungsgemeinschaft (DFG), the National Sciences and Engineering Research Council (NSERC) of Canada, and by the Fonds pour la Formation de Chercheurs et l'Aide à la Recherche (FCAR) of Québec.*

[1] Institut für Informatik, Universität Potsdam, August-Bebel-Straße 89, 14482 Potsdam, Germany.

[2] Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg, Postfach 4120, 39016 Magdeburg, Germany.

[3] Institut für Informatik, Universität Giessen, Arndtstraße 2, 35392 Giessen, Germany; holzer@informatik.uni-giessen.de

Article published by EDP Sciences

1. Introduction

The family REG of regular languages, defined as the family of languages accepted by (deterministic or nondeterministic) finite automata or, equivalently, generated by right-linear grammars, is one of the most important and well investigated classes of formal languages. Regular expressions, which were originally introduced by Kleene [21] and are a lovely set-theoretic characterization of regular languages, are better suited for human users and therefore are often used as interfaces to specify certain pattern or languages. *E.g.*, in the widely available programming environment Unix, regular (like) expressions can be found in legion of software tools like, *e.g.*, `awk`, `ed`, `emacs`, `egrep`, `lex`, `sed`, `vi`, etc., to mention a few of them. The syntax used to represent them may vary, but the concepts are very much the same everywhere.

Most of the above mentioned text-editing and searching programs add abbreviations and new operations to the basic regular expression notation from theoretical computer science, in order to make it easier to specify patterns or languages. This offers considerable convenience in both theory and practice. What concerns common abbreviations, as for instance, intersection and complement, they do not add more descriptive power to regular expressions, but give more concise descriptions. Besides the usage of *meta-characters* in Unix like expressions, the most significant difference to ordinary regular expressions is some sort of *pattern repeating operation* – see [10] for further details. More precisely, it is possible to specify patterns that are saved in a special holding space, used for further processing, on the underlying word. For instance, the Unix regular expression `\([ab]^*\)\1` describes the non-context-free language $\{ ww \mid w \in \{a,b\}^* \}$. Here `\1` serves as a holding space for the word that is matched by the regular expression enclosed in brackets `\(` and `\)`. This form of backreferencing was first introduced by [1]. A more formal treatment of regular expressions with backreferencing can be found in [24].

There have been some attempts to generalize Kleene's well-known theorem, which states that a language L is regular if and only if there is a regular expression r with $L = L(r)$, in one of the following directions: define an extension of regular expressions and determine the associated family of languages (see, *e.g.*, [15]) or find the class of expressions for a given extension of the family of regular languages (see, *e.g.*, [12,23,29]) characterizing two-dimensional regular languages, recognizable trace languages, and context-free (string) languages, respectively. On the other hand, to our knowledge nothing comes close to the repeating or copy operation mentioned above. This brings us to the aim of this paper. Inspired by Gruska's substitution expressions [13], which were used to characterize the context-free languages, we introduce regular expressions enriched by some sort of copy operation, which is close to the repeating feature of Unix regular like expressions.

A good formal language theoretic approach to those pattern repetition operations is given by the operation of *homomorphic replacement*. Homomorphic replacement is a concept well-known in computer science. We mention some areas where it appeared in literature under various names within different contexts:

for example, in van Wijngaarden grammars (W-grammars) homomorphic replacement is called "consistent substitution" or "consistent replacement" [9]. In connection with macro grammars [11] it is called "inside-out (IO) substitution", in Indian parallel grammars [27] the one-step derivation relation is nothing other then a homomorphic replacement with a finite set, and in some algebraical approach in formal language theory it appears as "call by value substitution". Another aspect of homomorphic replacement was investigated by Albert and Wegner [2], who considered H-systems.

In this paper, we study the usual language theoretic properties of regular expressions extended by homomorphic replacement, such as the descriptional power in comparison with the well-known classes in the Chomsky hierarchy as well as families of languages determined by mechanisms which are related to expressions with homomorphic replacement, closure properties and the complexity status of some decision problems for expressions with homomorphic replacement. In the next section we introduce the necessary definitions. Then in Section 3 we compare the power of substitution *versus* homomorphic replacement and in Section 4 we relate the latter to some other concepts in the literature. Sections 5 and 6 are devoted to the study of closure and decision problems as mentioned above. Finally in the penultimate section we summarize our results and state some open problems.

2. Definitions

We assume the reader to be familiar with some basic notions of formal language theory, as contained in [8]. Concerning our notations, for any set Σ, let Σ^+ be the free semigroup and Σ^* the free monoid with identity λ generated by Σ. For a word $w \in \Sigma^*$ let $|w|$ denote the length of the word; in particular, $|\lambda| = 0$. Moreover, for $w \in \Sigma^*$ and $a \in \Sigma$ let $|w|_a$ denote the number of occurrences of a in w. Set inclusion and strict set inclusion are denoted by $\subseteq$ and $\subset$, respectively. In particular we consider the following well-known formal language families generated by regular (*i.e.*, right-linear), linear context-free, context-free, and context-sensitive Chomsky grammars which are denoted by REG, LIN, CFL, and CSL, respectively. Moreover, the family of extended (extended deterministic, respectively) tabled context-free Lindenmayer languages is denoted by ET0L (EDT0L, respectively). The family of finite languages is denoted by FIN.

In this paper we are dealing with regular like expressions. *Ordinary* regular expression are defined as follows:

Definition 2.1 (R-expressions)**.** Let Σ be an alphabet. The regular expressions (R-expressions) over Σ and the sets that they denote are defined recursively as follows:

(1) $\emptyset$ is a regular expression and denotes the set $L(\emptyset) = \emptyset$.
(2) λ is a regular expression and denotes the set $L(\lambda) = \{\lambda\}$.
(3) For each $a \in \Sigma$, a is a regular expression and denotes the set $L(a) = \{a\}$.

(4) If r and s are regular expressions, then $(r + s)$, (rs), and (r^*) are regular expressions that denote the sets $L(r+s) = L(r) \cup L(s)$, $L(rs) = L(r) \cdot L(s)$, and $L(r^*) = L(r)^*$, respectively.
(5) Nothing else is a regular expression.

It is well known that regular expressions exactly characterize the family of regular languages REG. We call a language regular like expression language, if it can be described by a regular like expression, *i.e.*, a regular expression with an enhanced set of operations as, *e.g.*, union, concatenation, Kleene star, and iterated substitution or iterated homomorphic replacement. Both operations are defined formally in the next section.

Besides the expressive power of regular like expressions, we also investigate some complexity theoretical issues on these language families. We assume the reader to be familiar with some basic notions of complexity theory, as contained in [3]. In particular we consider the following well-known chain of inclusions: NL $\subseteq$ P $\subseteq$ NP $\subseteq$ PSPACE. Here NL is the set of problems accepted by nondeterministic logarithmic space bounded Turing machines, and P (NP, respectively) is the set of problems accepted by deterministic (nondeterministic, respectively) polynomially time bounded Turing machines. Moreover, PSPACE is $\bigcup_k \mathrm{DSpace}(n^k)$.

Completeness and hardness are always meant with respect to deterministic log-space many-one reducibilities. A problem A is said to be log-space many-one equivalent or as hard as B, if and only if A reduces to B and B reduces to A.

We investigate the fixed membership, the general membership, the equivalence, and the emptiness problem for regular like expression languages. The *fixed membership problem* for regular like expression languages is defined as follows:

- Fix a regular like expression r. For a given word w, is $w \in L(r)$?

A natural generalization is the *general membership problem* which is defined as follows:

- Given a regular like expression r and a word w, *i.e.*, an encoding $\langle r, w \rangle$, is $w \in L(r)$?

The *equivalence problem* is the following one:

- Given two regular like expressions r and s, does $L(r) = L(s)$ hold?

Finally, the *emptiness problem* is defined as:

- Given a regular like expression r, is $L(r) = \emptyset$?

The general membership, the equivalence, and emptiness problem have regular like expressions as inputs. Therefore we need an appropriate coding function $\langle \cdot \rangle$ which maps, *e.g.*, a regular like expression r and a string w into a word $\langle r, w \rangle$ over a fixed alphabet Σ. We do not go into the details of $\langle \cdot \rangle$, but assume that it fulfills certain standard properties; for instance, that the coding of the alphabet symbols is of logarithmic length.

3. Substitution versus homomorphic replacement

In this section we introduce the homomorphic replacement operation and study the expressive power of regular like expressions involving this new operation. We compare the induced language family to the lower classes of the Chomsky hierarchy and to the family EDT0L of languages generated by extended deterministic tabled 0L systems. Next we recall Gruska's [13] approach to characterize the context-free languages and then we define homomorphic replacement.

3.1. Substitution and iterated substitution

Recall the approach given by Gruska [13] in his seminal paper, where a-substitutions and their iteration are the additional operations to regular expressions.

Let a be a letter and L_1, L_2 be languages. The *a-substitution* of L_2 in L_1, denoted by $L_1 \leftarrow_a L_2$, is defined by

$$L_1 \leftarrow_a L_2 = \{\, u_1v_1u_2 \dots u_kv_ku_{k+1} \mid u_1au_2a \dots au_{k+1} \in L_1, \\ a \text{ does not occur in } u_1u_2 \dots u_{k+1}, \text{ and } v_1, v_2 \dots, v_k \in L_2 \,\},$$

and the *iterated a-substitution* of language L, denoted by $L^{\leftarrow_a}$, is defined by

$$L^{\leftarrow_a} = \{\, w \in L \cup (L \leftarrow_a L) \cup (L \leftarrow_a L \leftarrow_a L) \cup \cdots \mid \text{word } w \\ \text{has no occurrence of letter } a \,\}$$

where any further bracketing is omitted since a-substitution is obviously associative.

Based on these operations an extension of regular expressions is defined.

Definition 3.1 (S- and ES-expressions)**.** Let Σ be an alphabet. The *regular expressions with substitution* (S-expressions) and *regular expressions with extended substitution* (ES-expressions) over Σ and the sets they denote are defined recursively as follows:

(1) Every regular expression over Σ is an S- and ES-expression.
(2) If r and s are S- and ES-expressions, respectively, denoting the languages $L(r)$ and $L(s)$, respectively, then $(r + s)$, (rs), (r^*), and $(r \leftarrow_a s)$, for some $a \in \Sigma$, are S- and ES-expressions, respectively, that denote the sets $L(r) \cup L(s)$, $L(r) \cdot L(s)$, $L(r)^*$, and $L(r) \leftarrow_a L(s)$, respectively.
(3) Let $a \in \Sigma$. If r is an ES-expression denoting the language $L(r)$, then $(r^{\leftarrow_a})$ is an ES-expression that denotes the set $L(r)^{\leftarrow_a}$.
(4) Nothing else is an S- or ES-expressions, respectively.

The set of languages described by S- and ES-expressions is denoted by SREG and ESREG, respectively.

While SREG equals REG, which is easily seen, in [13], Gruska has shown that ESREG coincides with the family CFL of context-free languages.

3.2. Homomorphic and iterated homomorphic replacement

Homomorphic replacement was investigated by Albert and Wegner [2] and appeared in the literature under various names within different contexts. For instance, in van Wijngaarden grammars (W-grammars) homomorphic replacement is called "consistent substitution" or "consistent replacement" [9]. In connection with macro grammars [11] it is called "inside-out (IO) substitution", in Indian parallel grammars [27] the one-step derivation relation is nothing other then a homomorphic replacement with a finite set, and in some algebraical approach in formal language theory it appears as "call by value substitution". The essential feature of homomorphic replacement is copying. Thus, we introduce an operation on languages which models this feature. Our definition was inspired by Gruska's a-substitution [13]. According to the definition of a-substitution, we have to replace any occurrence of a by a word of L_2, and it is allowed that different occurrences are replaced by different words. We now modify this mechanism by the requirement that any occurrence of a has to be replaced by the same word of L_2.

Definition 3.2. Let a be a letter and L_1, L_2 be languages. The *a-homomorphic replacement* of L_2 in L_1, denoted by $L_1 \Leftarrow_a L_2$, is defined by

$$L_1 \Leftarrow_a L_2 = \{\, u_1 v u_2 \dots u_k v u_{k+1} \mid u_1 a u_2 a \dots a u_{k+1} \in L_1, \\ a \text{ does not occur in } u_1 u_2 \dots u_{k+1}, \text{ and } v \in L_2 \,\}.$$

The reader may easily verify that the following lemma is valid.

Lemma 3.3. *For each letter a, the operation $\Leftarrow_a$ is associative,* i.e.*,*

$$\Big((L_1 \Leftarrow_a L_2) \Leftarrow_a L_3\Big) = \Big(L_1 \Leftarrow_a (L_2 \Leftarrow_a L_3)\Big).$$

Observe, that the previous lemma is not true if we use different letters for the replacement operation because

$$\Big((\{b\} \Leftarrow_a \{a\}) \Leftarrow_b \{a\}\Big) = \{a\} \neq \{b\} = \Big(\{b\} \Leftarrow_a (\{a\} \Leftarrow_b \{a\})\Big).$$

We also consider the iterated version of homomorphic replacement.

Definition 3.4. Let a be a letter and L a language. The *iterated a-homomorphic replacement* of L, denoted by $L^{\Leftarrow_a}$, is defined by

$$L^{\Leftarrow_a} = \{\, w \in L \cup (L \Leftarrow_a L) \cup (L \Leftarrow_a L \Leftarrow_a L) \cup \cdots \mid \text{word } w \\ \text{has no occurrence of letter } a \,\}.$$

Due to Lemma 3.3 we do not have to specify the bracketing of the a-homomorphic replacement operations in the previous definition. Note, if a is not in Σ, then for

language $L \subseteq \Sigma^*$ we have $L^* = (La \cup \{\lambda\})^{\Leftarrow_a}$ and $L^+ = (La \cup L)^{\Leftarrow_a}$. Here λ denotes the empty word.

Homomorphic replacement is very powerful, because one can describe the non-context-free language $\{ ww \mid w \in \{a,b\}^* \}$ by $\{cc\} \Leftarrow_c \{a,b\}^*$. In fact, this shows that the low levels of the Chomsky hierarchy are not closed under a-homomorphic and iterated a-homomorphic replacement.

Theorem 3.5.

(1) *The family of finite languages is closed under a-homomorphic replacement. Neither the family of regular, linear context-free nor the family of context-free languages is closed under a-homomorphic replacement.*
(2) *Neither the family of finite languages, regular, linear context-free nor the family of context-free languages is closed under iterated a-homomorphic replacement.* □

Obviously, the family of recursively enumerable languages is closed under a-homomorphic replacement, but for the family of context-sensitive languages we have to be careful whether the replacement is λ-free or not. In the λ-free case CSL is closed under this type of operation what can readily be shown by a LBA construction. In general this family is not closed under a-homomorphic replacement, because it is possible to simulate arbitrary homomorphisms and the well-known fact that every recursively enumerable language is a homomorphic image of a context-sensitive language. We briefly summarize our results:

Theorem 3.6. *The family of context sensitive languages is not closed under arbitrary (iterated) a-homomorphic replacement, but is closed under λ-free one. Finally, the family of recursively enumerable languages is closed under a-homomorphic and iterated a-homomorphic replacement.*

Now we are ready to define the central notion of this paper, which is that of regular expressions with (iterated) homomorphic replacement.

Definition 3.7 (H- and EH-expressions)**.** Let Σ be an alphabet. The regular expressions with homomorphic replacement (H-expressions) and extended homomorphic replacement (EH-expressions), respectively, over Σ and the sets they denote are recursively defined as follows:

(1) Every regular expression over Σ is also an H- and EH-expression, respectively.
(2) If r and s are H- and EH-expressions, respectively, denoting the languages $L(r)$ and $L(s)$, respectively, then $(r + s)$, (rs), (r^*), and $(r \Leftarrow_a s)$, for some $a \in \Sigma$, are H- and EH-expressions, respectively, that denote the sets $L(r) \cup L(s)$, $L(r) \cdot L(s)$, $L(r)^*$, and $L(r) \Leftarrow_a L(s)$, respectively.
(3) Let $a \in \Sigma$. If r is an EH-expression denoting the language $L(r)$, then $(r^{\Leftarrow_a})$ is an EH-expression that denotes the set $L(r)^{\Leftarrow_a}$.
(4) Nothing else are H- and EH-expressions, respectively.

The set of languages described by H- and EH-expressions is denoted by HREG and EHREG, respectively.

If there is no danger of confusion, we omit out-most brackets. Let us give some examples:

Example 3.8.

(1) $cc \Leftarrow_c (a+b)^*$ denotes the language $\{ ww \mid w \in \{a,b\}^* \}$, which is non-context-free.
(2) $(ab + aAb)^{\Leftarrow_A}$ describes the non-regular language $\{ a^n b^n \mid n \geq 1 \}$.
(3) $(a + AA)^{\Leftarrow_A}$ denotes the non-context-free language $\{ a^{2^n} \mid n \geq 0 \}$.

Next, consider the following chain of inclusions:

Theorem 3.9. REG $\subset$ HREG $\subset$ EHREG.

Proof. The inclusions are obvious; the strictness of the first one is seen from Example 3.8, item (1) and the strictness of the second inclusion follows by Example 3.8, item (3) together with the fact that every language in HREG is semi-linear. This is because ordinary regular operations and, by easy calculations, also a-homomorphic replacement preserves semi-linearity. □

In the following theorem we relate EHREG with the linear context-free languages and the family EDT0L. For further details on EDT0L languages we refer to [25].

Theorem 3.10. LIN $\subset$ EHREG $\subseteq$ EDT0L.

Proof. Let $G = (N, T, P, S)$ be a linear context-free grammar with the set of nonterminals $N = \{A_1, A_2, \ldots, A_n\}$ and let $S = A_1$. Then for $1 \leq i \leq n$, we set

$$G_i = \left(N \setminus \{A_1, A_2, \ldots, A_{i-1}\}, T \cup \{A_1, A_2, \ldots, A_{i-1}\}, \bigcup_{j=i}^{n} P_j, A_i \right),$$

where $P_i = \{ A_i \to w \mid A_i \to w \in P \}$. Moreover, for $1 \leq i \leq n$, let s_i be the EH-expressions with $L(s_i) = \{ w \mid A_i \to w \in P \}$. Then inductively define

$$r_n = (s_n)^{\Leftarrow_{A_n}}$$

and

$$r_i = \left(\left(\ldots \left(\left(s_i^{\Leftarrow_{A_i}} \Leftarrow_{A_n} r_n \right) \Leftarrow_{A_{n-1}} r_{n-1} \right) \ldots \right) \Leftarrow_{A_{i+1}} r_{i+1} \right)^{\Leftarrow_{A_i}},$$

for $1 \leq i \leq n-1$. Then one can readily verify that $L(G_i) = L(r_i)$ for $1 \leq i \leq n$, which immediately implies $L(G) = L(r_1)$, because G_1 equals G. This proves the first inclusion which has to be strict by Example 3.8, item (3).

The second inclusion follows by the closure of EDT0L under the operations in consideration, which can be shown by standard constructions. □

In order to relate the families HREG and EHREG to the families of linear context-free, context-free, and EDT0L languages, the following to lemmata are needed.

Definition 3.11. We define the depth of an R- or H-expression over alphabet Σ inductively by

(1) $d(\emptyset) = d(\lambda) = d(a) = 0$ for any $a \in \Sigma$.
(2) If r and s are R- or H-expressions of depth $d(r)$ and $d(s)$, respectively, then $d(r + s) = d(r \cdot s) = d(r \Leftarrow_a s) = d(r) + d(s) + 1$ for $a \in \Sigma$.
(3) If r is an R- or H-expression of depth $d(r)$, then $d(r^*) = d(r) + 1$.

For a language $L \in$ HREG, we set

$$d(L) = \min\{\, d(r) \mid L(r) = L \,\}.$$

We say that an H-expression r is λ-free if it does not contain a subexpression $s \Leftarrow_a u$ with $L(u) = \{\lambda\}$.

Lemma 3.12. *For any H-expression $r = (s \Leftarrow_a u)$ with $L(u) = \{\lambda\}$ there is a λ-free H-expression t such that $L(t) = L(r)$ and $d(t) \leq d(s)$.*

Proof. Let us assume that the lemma does not hold. Let K be the set of all H-expressions r such that r is of the form $r = (s \Leftarrow_a u)$ with $L(u) = \{\lambda\}$ and there is no t for r satisfying the conditions of the lemma. By assumption, K is not empty. Let $k = \min\{\, d(r) \mid r \in K \,\}$. We consider an H-expression $r = (s \Leftarrow_a u) \in K$ such that $d(r) = k$. Obviously, if $s \Leftarrow_a u$ in K, then $s \Leftarrow_a \lambda$ is in K, too. By the minimality of r with respect to the depth, we can assume without loss of generality that $r = (u \Leftarrow_a \lambda)$.

Obviously, $k \geq 1$. In case $k = 1$, then one of the following cases holds:

(1) If $s = \emptyset$, then $L(s \Leftarrow_a \lambda) = L(\emptyset)$ and $d(\emptyset) = d(s)$.
(2) If $s = \lambda$, then $L(s \Leftarrow_a \lambda) = L(\lambda)$ and $d(\lambda) = d(s)$.
(3) If $s = a$, then $L(s \Leftarrow_a \lambda) = L(\lambda)$ and $d(\lambda) = d(s)$.
(4) If $s = b$ for $b \in \Sigma \setminus \{a\}$, then $L(s \Leftarrow_a \lambda) = L(b)$ and $d(b) = d(s)$.

Thus, let $k > 1$ and we distinguish the following four cases:

(1) Let $s = s_1 + s_2$ for some H-expressions s_1 and s_2 with $d(s_1) \leq k - 2$ and $d(s_2) \leq k - 2$. Then we define the H-expressions $t_1 = (s_1 \Leftarrow_a \lambda)$ and $t_2 = (s_2 \Leftarrow_a \lambda)$. Obviously, $d(t_1) \leq k - 1$ and $d(t_2) \leq k - 1$. By the minimality of k, there exist λ-free H-expressions t_1' and t_2' with $L(t_1') = L(s_1 \Leftarrow_a \lambda)$ and $L(t_2') = L(s_2 \Leftarrow_a \lambda)$, respectively, satisfying $d(t_1') \leq d(s_1)$ and $d(t_2') \leq d(s_2)$. Thus, $t_1' + t_2'$ fulfills

$$d(t_1' + t_2') = d(t_1') + d(t_2') + 1 \leq d(s_1) + d(s_2) + 1 = d(s)$$

and

$$\begin{aligned} L(t'_1 + t'_2) &= L(t'_1) \cup L(t'_2) \\ &= L(s_1 \Leftarrow_a \lambda) \cup L(s_2 \Leftarrow_a \lambda) = L((s_1 \Leftarrow_a \lambda) + (s_2 \Leftarrow_a \lambda)) \\ &= L((s_1 + s_2) \Leftarrow_a \lambda) = L(s \Leftarrow_a \lambda) = L(r). \end{aligned}$$

Moreover, because t'_1 and t'_2 are λ-free, expression $t'_1 + t'_2$ is λ-free, too. Hence, $t'_1 + t'_2$ fulfills all conditions of the lemma in contrast to $r \in K$.

(2) Let $s = s_1 s_2$ for some H-expressions s_1 and s_2 with $d(s_1) \leq k-2$ and $d(s_2) \leq k-2$. In analogy to the first case above, we can show a contradiction which is left to the reader.

(3) Let $s = s_1^*$ for some H-expressions s_1 with $d(s_1) \leq k-2$. Again, we can show a contradiction analogously to the first case above.

(4) Let $s = (s_1 \Leftarrow_b s_2)$ for some H-expressions s_1 and s_2 with $d(s_1) \leq k-2$ and $d(s_2) \leq k-2$. We consider the λ-free H-expressions t'_1 and t'_2 as in the first case above. Therefore

$$\begin{aligned} &L(t'_1) = L(s_1 \Leftarrow_a \lambda), \\ &L(t'_2) = L(s_2 \Leftarrow_a \lambda) \quad \text{with} \quad d(t'_1) \leq d(s_1) \quad \text{and} \quad d(t'_2) \leq d(s_2). \end{aligned} \tag{1}$$

Moreover, if $a \neq b$, then

$$\begin{aligned} L(t'_1 \Leftarrow_b t'_2) &= L((s_1 \Leftarrow_a \lambda) \Leftarrow_b (s_2 \Leftarrow_a \lambda)) \\ &= L((s_1 \Leftarrow_b s_2) \Leftarrow_a \lambda) = L(s \Leftarrow_a \lambda) = L(r). \end{aligned} \tag{2}$$

If $a = b$, for $1 \leq i \leq 2$, we modify s_i to s'_i by a renaming of a by a' where a' is a new letter and get the relations of Equations (1) and (2) for the corresponding λ-free expressions t'_1 and t'_2.

Let $L(t'_1) \neq \{\lambda\}$. Then, in analogy to the above consideration, a contradiction to the choice of r is obtained. Finally let $L(t'_2) = \{\lambda\}$. Then

$$d(t'_1 \Leftarrow_b t'_2) \leq d(s_1) + d(s_2) + 1 = d(s) < d(r). \tag{3}$$

By the minimality of k, there is a λ-free H-expression t such that $L(t) = L(t'_1 \Leftarrow_b t'_2)$ and $d(t) \leq d(t'_1)$. By Equations (1)–(3), we obtain $L(t) = L(r)$ and $d(t) \leq d(s_1) \leq d(s) \leq d(r)$. Therefore t satisfies all conditions of the lemma in contrast to the choice of $r \in K$. □

For an alphabet Σ, a partition $C = (\Sigma_1, \Sigma \setminus \Sigma_1)$ and two letters a and b not in Σ we define the morphism τ_C by

$$\tau_C(x) = \begin{cases} a & \text{if } x \in \Sigma_1 \\ b & \text{if } x \in \Sigma \setminus \Sigma_1. \end{cases}$$

Let L be a language over Σ and a and b two letters not in Σ. Then L is called an *(a,b)-language* iff there exist a partition $C = (\Sigma_1, \Sigma \setminus \Sigma_1)$ of Σ such that the following conditions hold:

Condition A1: $\tau_C(L) \subseteq a^*b^*$.
Condition A2: $\tau_C(L)$ is infinite.
Condition A3: For any natural number n, $D(a,n,L) = \{\, m \mid a^n b^m \in \tau_C(L) \,\}$ is a finite set.
Condition A4: For any natural number n, $D(b,n,L) = \{\, m \mid a^m b^n \in \tau_C(L) \,\}$ is a finite set.

We note that if there exists a constant $k \geq 0$ such that such that $a^n b^m \in \tau_c(L)$ implies $|n-m| \leq k$, then conditions A3 and A4 are satisfied. The converse is not true as one can see from the language $\{\, a^n b^m \mid n \leq m \leq 2n \,\}$.

Before showing that any (a,b)-language is not an HREG language we need the following statements on the behaviour of (a,b)-languages under the operation used in the construction of HREG languages.

Lemma 3.13.

(1) *If $L_1 \cup L_2$ is an (a,b)-language, then L_1 or L_2 are (a,b)-languages.*
(2) *If $L_1 \cdot L_2$ is an (a,b)-language, then L_1 or L_2 are (a,b)-languages.*
(3) *For any L, language L^* is not an (a,b)-language.*
(4) *If the set $L_1 \Leftarrow_c L_2$ is an (a,b)-language, for some c, and $L_2 \neq \{\lambda\}$, then L_1 or L_2 are (a,b)-languages.*

Proof.

(1) Let C be the partition for $L_1 \cup L_2$. Because $\tau_C(L_i) \subseteq \tau_C(L_1 \cup L_2) \subseteq a^*b^*$ and $D(x,n,L_i) \subseteq D(x,n,L_1 \cup L_2)$, for $i \in \{1,2\}$ and $x \in \{a,b\}$, conditions A1, A3 and A4 hold for the languages L_1 and L_2, too. Moreover, the infinity of $\tau_C(L_1 \cup L_2)$ implies that at least one of the languages $\tau_C(L_1)$ and $\tau_C(L_2)$ is infinite. Hence condition A2 holds for L_1 or L_2, too.

(2) Again, let C be the partition for $L_1 \cdot L_2$. Since $\tau_C(L_1 \cdot L_2) = \tau_C(L_1) \cdot \tau_C(L_2)$ and $L_1 \cdot L_2$ satisfies conditions A1 and A2, both factors $\tau_C(L_1)$ and $\tau_C(L_2)$ are contained in a^*b^* and one of the factors has to be infinite and the other one is non-empty. Let us assume that L_1 is infinite.

We prove that L_1 satisfies condition A3. If A3 does not hold for L_1, then there is an integer n such that $D(a,n,L_1)$ is infinite. Let $a^n b^m \in \tau_C(L_1)$ for some $m \geq 1$. Let v be a word of L_1 with $\tau_C(v) = a^n b^m$. Furthermore, let $w \in L_2$ and $\tau_C(w) = a^s b^r$. If $s \geq 0$, then $a^n b^m a^s b^r = \tau_C(vw) \in \tau_C(L_1 \cdot L_2)$ in contrast to the validity of condition A1 for $L_1 \cdot L_2$. If $s = 0$, then $m \in D(a,n,L_1)$ iff $m+r \in D(a,n,L_1 w)$, and thus $D(a,n,L_1 w)$ is infinite. By $D(a,n,L_1 w) \subseteq D(a,n,L_1 L_2)$ we obtain a contradiction to the validity of condition A3 for $L_1 \cdot L_2$.

Analogously, we prove that L_1 satisfies condition A4. Combining these facts, language L_1 is an (a,b)-language By similar arguments we can show that in case of infinity of L_2. Thus, L_2 is an (a,b)-language.

(3) Let us assume that L^* is an (a,b)-language, and let C be the partition for L^*. Since $\tau_C(L^*) = (\tau_C(L))^*$ and $\tau_C(L^*)$ is infinite by condition A2, $\tau_C(L) \neq \emptyset$ and $\tau_C(L) \neq \{\lambda\}$. Moreover, $\tau_C(L) \subseteq a^*b^*$ since condition A1 holds for L^*. If $\tau_C(L)$ contains a word a^rb^s with $r \geq 1$ and $s \geq 1$, then $a^rb^sa^rb^s \in (\tau_C(L))^2 \subseteq \tau_C(L^*)$ in contrast to the validity of condition A1 for L^*. Hence $\tau_C(L) \subseteq a^*$ or $\tau_C(L) \subseteq b^*$. In the former case we get $a^r \in \tau_C(L)$ with $r \geq 1$. Thus, $\{a^{kr} \mid k \geq 0\} \subseteq \tau_C(L^*)$ and $D(b,0,L^*)$ is infinite in contrast to the validity of condition A4 for L^*. Analogously, we show a contradiction in the case that $\tau_C(L) \subseteq b^*$.

(4) If $|w|_c = 0$ for all $w \in L_1$, then $L_1 \Leftarrow_c L_2 = L_1$ and the statement is shown. Thus, we can assume that there is a word $w \in L_1$ with $|w|_c \geq 1$.

Again, let $C = (\Sigma_1, \Sigma \setminus \Sigma_1)$ be the partition. Obviously, $\tau_C(L_2) \subseteq a^*b^*$. We consider the following three subcases:

(a) Let $\tau_C(L_2) \subseteq a^*$. If $\tau_C(L_2)$ is infinite, then, for any $w \in L_1$, language $\tau_C(w \Leftarrow_c L_2)$ is infinite, too. Therefore there is an integer n such that $D(b,n,w \Leftarrow_c L_2)$ and hence $D(b,n,L_1 \Leftarrow_c L_2)$ are infinite. This contradicts condition A4 for $L_1 \Leftarrow_c L_2$.

Thus, we can assume that $\tau_C(L_2) \subseteq a^*$ is finite. We now prove that L_1 is an (a,b)-language w.r.t. the partition $D = (\Sigma_1 \cup \{c\}, \Sigma \setminus (\Sigma_1 \cup \{c\}))$. Note that $C = D$ is possible. Since c is substituted by words of a^* in $L_1 \Leftarrow_c L_2$, we obtain $\tau_D(L_1) \subseteq a^*b^*$, *i.e.*, language L_1 satisfies condition A1. Moreover, the infinity of $\tau_C(L_1 \Leftarrow_c L_2)$ and the finiteness of $\tau_C(L_2)$ imply the infinity of $\tau_D(L_1)$. Hence condition A2 is fulfilled by L_1.

Now assume that L_1 does not satisfy condition A4. Then there is an integer n such that $D(b,n,L_1)$ is infinite. Let $k \geq 0$ be an arbitrary integer. Since $D(b,n,L_1)$ is infinite, there is an integer $k' \geq k$ such that $a^{k'}b^n \in \tau_D(L_1)$. Let u be a word in L_1 with $\tau_D(u) = a^{k'}b^n$. Then, by $L_2 \neq \{\lambda\}$, the set $\tau_C(u \Leftarrow_c L_2)$ contains a word $a^{k''}b^n$ with $k'' \geq k' \geq k$. Thus, $D(b,n,L_1 \Leftarrow_c L_2)$ is infinite, too, in contrast to the validity of condition A4 for $L_1 \Leftarrow_c L_2$.

Now assume that L_1 does not satisfy condition A3. Then there is an integer n such that $D(a,m,L_1)$ is infinite. Let w be an element of L_1 with $\tau_D(w) \in a^mb^*$. Then $w = w'w''$ for some $w' \in (V_1 \cup \{c\})^*$ and $w'' \in (V \setminus (V_1 \cup \{c\}))^*$ with $|w'| = m$. Since there is a finite number of different words w' with $w' \in (V_1 \cup \{c\})^*$ and $|w'| = m$, the infinity of $D(a,m,L_1)$ implies the existence of a word w' over $\Sigma_1 \cup \{c\}$ of length m such that

$$E = \{\, \tau_D(w'') \mid w'' \in (\Sigma \setminus (\Sigma_1 \cup \{c\}))^*, w'w'' \in L_1, \tau_D(w'w'') \in a^mb^* \,\}$$

is infinite. We set

$$F = \{\, w'w'' \mid w'w'' \in L_1 \text{ and } \tau_D(w'') \in E \,\}.$$

Let

$$w' = w_1 c^{i_1} w_2 c^{i_2} \dots w_r c^{i_r} w_{r+1}$$

for some $r \geq 0$ with $w_{r+1} \in (\Sigma_1 \setminus \{c\})^*$ and $w_j \in (\Sigma_1 \setminus \{c\})^*$, $i_j \geq 1$ for $1 \leq j \leq r$. Then

$$|w_1 w_2 \dots w_{r+1}| + (i_1 + \dots + i_r) = m.$$

Let $v \in L_2$ with $\tau_C(v) = a^s$. Then

$$D(a, |w_1 w_2 \dots w_{r+1}| + (i_1 + \dots + i_r)s, F \Leftarrow_c v)$$

and therefore

$$D(a, |w_1 w_2 \dots w_{r+1}| + (i_1 + \dots + i_r)s, L_1 \Leftarrow_c L_2)$$

are infinite which gives the desired contradiction.

(b) Let $\tau_C(L_2) \subseteq b^*$. We obtain a contradiction analogously to the first case above.

(c) Let $\tau_C(L_2) \subseteq a^+b^+$. First let us assume that there is a word $w \in L_1$ with at least two occurrences of c. Then the existence of a word $v \in L_2$ with $\tau_C(v) = a^r b^s$ with $r > 0$ and $s > 0$ implies $\tau_C(w \Leftarrow_c v) = u_1 a^r b^s u_2 a^r b^s u_3 \in \tau_C(L_1 \Leftarrow_c L_2)$ for some words $u_1, u_2, u_3 \in \{a, b\}^*$, *i.e.*, condition A1 does not hold for $L_1 \Leftarrow_c L_2$ in contrast to our supposition. Thus, we can assume that any word of L_1 contains at most one occurrence of c. Moreover, by analogous arguments, any word w of L_1 with $|w|_c = 1$ has the form $w = w_1 c w_2$ with $w_1 \in \Sigma_1^*$ and $w_2 \in \Sigma \setminus (\Sigma_1 \cup \{c\})$.

Let $\tau_C(L_2)$ be infinite. We prove that L_2 is an (a, b)-language. Language L_1 contains a word $w = w_1 c w_2$ with $w_1 \in \Sigma_1^*$ and $w_2 \in \Sigma \setminus (\Sigma_1 \cup \{c\})$. If $|w_1| = r$ and $|w_2| = s$, then $\tau_C(w) = a^{r+1} b^s$ or $\tau_C(w) = a^r b^{s+1}$. In the sequel we only discuss the former case, the latter one can be handled by analogous considerations. If L_2 is not an (a, b)-language, then one of the sets $D(a, n, L_2)$ or $D(b, n, L_2)$ is infinite. This implies the infinity of $D(a, n + r + 1, w \Leftarrow_a L_2)$ or $D(b, n + s, w \Leftarrow_a L_2)$. Therefore, $D(a, n + r + 1, L_1 \Leftarrow_a L_2)$ or $D(b, n+s, L_1 \Leftarrow_a L_2)$ is infinite in contrast to the fact that $L_1 \Leftarrow_a L_2$ is an (a, b)-language.

Thus, let $\tau_C(L_2)$ be finite. We show again, that L_1 is an (a, b)-language with respect to the partition D defined as above. Obviously, $\tau_D(L_1)$ is infinite and contained in a^*b^*. Now assume that L_1 does not satisfy condition A4. Then there is an integer n such that $D(b, n, L_1)$ is infinite. Let $k \geq 0$ be an arbitrary integer. Since $D(b, n, L_1)$ is infinite, there is an integer $k' \geq k$ such that $a^{k'} b^n \in \tau_D(L_1)$. Let u be a word in L_1 with $\tau_D(u) = a^{k'} b^n$. Then, by $L_2 \neq \{\lambda\}$, the language $\tau_C(u \Leftarrow_c L_2)$ contains a word $a^{k''} b^n$ with $k'' \geq k' \geq k$. Thus, $D(b, n, L_1 \Leftarrow_c L_2)$ is infinite, too, in contrast to the validity of

condition A4 for $L_1 \Leftarrow_c L_2$. Analogously we prove that language L_1 satisfies condition A3. □

Now we are ready to show that no (a,b)-language can be an HREG language.

Lemma 3.14. *Any (a,b)-language is not an* HREG *language.*

Proof. Let us assume that there is an (a,b)-language K in HREG. Let

$$k = \min\{\, d(K) \mid K \in \text{HREG and } K \text{ is an } (a,b)\text{-language} \,\}$$

and let L be an (a,b)-language in HREG with $d(L) = k$. By Lemma 3.12, there is an H-expression r constructed without steps of the form $s \Leftarrow_c \lambda$ such that $L(r) = L$. Then $k \geq 1$ since (a,b)-languages are infinite by condition A2. Now, by Lemma 3.13 there are H-expressions s and t with $d(s) < k$ and $d(t) < k$ such that $r = s + t$ or $r = st$ or $r = (s \Leftarrow_c t)$ for some c. By Lemma 3.13 we obtain that $L(s)$ or $L(t)$ are (a,b)-languages in contrast to the definition of k. □

With the previous lemma we can show some incomparable results for the language family HREG.

Theorem 3.15. *Let $X \in \{\text{CFL}, \text{LIN}\}$. Then the family of languages X is incomparable to the family* HREG.

Proof. By Theorem 3.9 it is sufficient to show that there is are languages $K_1 \in \text{LIN} \setminus \text{HREG}$ and $K_2 \in \text{HREG} \setminus \text{CFL}$. Obviously, the linear context-free language $K_1 = \{\, c^n d^n \mid n \geq 1 \,\}$ is an (a,b)-language. Thus, $K_1 \notin$ HREG follows from Lemma 3.14. If we choose $K_2 = \{\, wcw \mid w \in \{a,b\}^* \,\}$, we are, obviously, done. □

We have already seen that HREG contains non-context-free languages. On the other hand, it is known, that the Dyck set is not an EDT0L language ([25], Exercise 3.3, p. 205), and thus is not contained in HREG by Theorem 3.10. This proves the following corollary.

Corollary 3.16. *The language families* CFL *and* EHREG *are incomparable.* □

4. Homomorphic replacement systems and related mechanisms

In this section we discuss several aspects of homomorphic replacement which are related to H- and EH-expressions. As already mentioned, homomorphic replacement was investigated by Albert and Wegner [2] in the context of homomorphic replacement systems. As we will see, homomorphic replacement with regular languages in the sense of Albert and Wegner is a special case of H-expressions. These systems are defined as follows:

Definition 4.1 (H-systems)**.** A homomorphic replacement system (H-system) is a quadruple $H = (\Sigma_1, \Sigma_2, L_1, \varphi)$ with meta-alphabet Σ_1, terminal alphabet Σ_2, such that $\Sigma_1 \cap \Sigma_2 = \emptyset$, meta-language $L_1 \subseteq \Sigma_1^*$, and a function $\varphi : \Sigma_1 \to 2^{\Sigma_2^*}$

which assigns to each $a \in \Sigma_1$ a language $\varphi(a) \subseteq \Sigma_2^*$. Instead of $\varphi(a)$ we shall write also L_a.

The language of an H-system $H = (\Sigma_1, \Sigma_2, L_1, \varphi)$ is defined as

$$L(H) = \{h(w) \mid w \in L_1 \text{ and } h \text{ is a homomorphism with } h(a) \in \varphi(a) \text{ for all } a \in \Sigma_1\}.$$

The family of H-system languages with regular meta-languages and regular languages L_a, for every $a \in \Sigma_1$, is denoted by $\mathcal{H}(\mathrm{REG}, \mathrm{REG})$.

Recently a restricted form of homomorphic replacement systems, so called pattern or multi-pattern languages [20,22] have gained interest in the formal language community. Pattern (multi-pattern, respectively) languages are languages generated by H-systems with the following restrictions:

(1) L_1 is a singleton (or a finite language, respectively),
(2) there is a partition of Σ_1 into Σ_1' and Σ_1'', and
(3) $\varphi(a) \subseteq \Sigma_2$ is a singleton for $a \in \Sigma_1'$ and $\varphi(b) = \Sigma_2^*$ for $b \in \Sigma_1''$.

Let PAT (MPAT, respectively) denote the family of all pattern (multi-pattern, respectively) languages.

Obviously, multi-pattern languages are a subset of $\mathcal{H}(\mathrm{FIN}, \mathrm{REG})$, the family of H-system languages with finite meta-languages and regular languages L_a for every $a \in \Sigma_1$. Because the $\mathcal{H}(\mathrm{REG}, \mathrm{REG})$ language $\{ (a^n b)^m \mid n, m \geq 1 \}$ generated by the H-system $H = (\{A, B\}, \{a, b\}, L_1, \varphi)$ with $L_1 = \{ (AB)^m \mid m \geq 1 \}$ and $\varphi(A) = a^+$ and $\varphi(B) = b$, doesn't belong to $\mathcal{H}(\mathrm{FIN}, \mathrm{REG})$, which was shown in [2], we obtain the following theorem, where the first strict inclusion is due to [20]:

Theorem 4.2. $\mathrm{PAT} \subset \mathrm{MPAT} \subset \mathcal{H}(\mathrm{REG}, \mathrm{REG})$.

Moreover, by the fact that $(ab)^*$ is not a multi-pattern language but belongs to $\mathcal{H}(\mathrm{REG}, \mathrm{REG})$ one concludes that the family of pattern and multi-pattern languages are incomparable with the family REG, LIN, and CFL of regular, linear context-free, and context-free languages, respectively. Now consider the following chain of strict inclusions:

Theorem 4.3. $\mathrm{REG} \subset \mathcal{H}(\mathrm{REG}, \mathrm{REG}) \subset \mathrm{HREG}$.

Proof. The first inclusion is obvious; the strictness is seen from the non-regular language $\{ a^n b a^n \mid n \geq 1 \}$ generated by the H-system $H = (\{A, B\}, \{a, b\}, L_1, \varphi)$ with $L_1 = \{ABA\}$ and $\varphi(A) = a^*$ and $\varphi(B) = b$.

Let $L \in \mathcal{H}(\mathrm{REG}, \mathrm{REG})$. Then there is an H-system $H = (\Sigma_1, \Sigma_2, L_1, \varphi)$ with regular meta-language L_1 and regular languages L_a for all $a \in \Sigma_1$, such that $L = L(H)$. Without loss of generality we assume that $\Sigma_1 = \{a_1, \ldots, a_n\}$.

Since L_1 (L_a for $a \in \Sigma_1$, respectively) is regular there exists a regular expression r_1 (r_a for $a \in \Sigma_1$, respectively) such that $L_1 = L(r_1)$ ($\varphi(a) = L(r_a)$, respectively). Because $\Sigma_1 \cap \Sigma_2 = \emptyset$ it is easy to see that the H-expression

$$\Big(\Big(\ldots \Big(\big(r_1 \Leftarrow_{a_1} r_{a_1} \big) \Leftarrow_{a_2} r_{a_2} \Big) \ldots \Big) \Leftarrow_{a_n} r_{a_n} \Big)$$

exactly describes language L. This shows that $\mathcal{H}(\mathrm{REG},\mathrm{REG}) \subseteq \mathrm{HREG}$.

It remains to show that the inclusion is proper. By Albert and Wegner [2] it was shown that the language

$$\{\, (a^n b)^m \#(a^n b)^m \mid n, m \geq 1 \,\} \notin \mathcal{H}(\mathrm{REG},\mathrm{REG}).$$

The reader may verify, that the H-expression

$$\Big((A\#A) \Leftarrow_A \big(B^+ \Leftarrow_B (a^+ b)\big)\Big) \quad \text{or} \quad \Big(\big((A\#A) \Leftarrow_A B^+\big) \Leftarrow_B (a^+ b)\Big)$$

describes this language. Thus, the claim follows. □

We want to stress that Theorem 3.15 can be generalized as follows. We state the result without proof.

Theorem 4.4. *Let* $X \in \{\mathrm{CFL},\mathrm{LIN}\}$ *and* $Y \in \{\mathrm{HREG},\mathcal{H}(\mathrm{REG},\mathrm{REG})\}$. *Then the family of languages* X *is incomparable to the family of languages* Y.

A slightly more general class than $\mathcal{H}(\mathrm{REG},\mathrm{REG})$ was introduced and investigated by Birget and Stephen [5]. They define a *uniform substitution* to be a function $S_H : \Sigma_1 \to 2^{\Sigma_2}$, which is determined by a set H of homomorphisms $\Sigma_1^* \to \Sigma_2^*$ as follows: for $w \in \Sigma_1$, we define $S_H(w) = \{\, \varphi(w) \mid \varphi \in H \,\}$ and for a language L in Σ_1^* set $S_H(L) = \{\, \varphi(w) \mid w \in L \text{ and } \varphi \in H \,\}$. Then let RECREG be the class of languages of the form $S_H(L)$, where L is regular and H is a recognizable set of homomorphisms form Σ_1^* to Σ_2, *i.e.*, for $\Sigma_1 = \{v_1,\ldots,v_n\}$ the set $\{\varphi(v_1)\#\ldots\#\varphi(v_n) \in (\Sigma_2 \cup \{\#\})^* \mid \varphi \in H\,\}$ is a regular subset of $(\Sigma_2 \cup \{\#\})^*$, where $\#$ is a symbol not in Σ_2. By Mezei's theorem (see, *e.g.*, [5], p. 257, Thm. A.1), the set $\{\varphi(v_1)\#\ldots\#\varphi(v_n) \in (\Sigma_2 \cup \{\#\})^* \mid \varphi \in H\,\}$ is regular if and only if it is equal to a finite union of sets of the form $L_1\#\ldots\#L_n$, where each L_i, for $1 \leq i \leq n$, is regular. Using this fact, one can easy see that RECREG is a subset of HREG. Moreover, the inclusion is strict, because the above used language to separate $\mathcal{H}(\mathrm{REG},\mathrm{REG})$ from HREG is also not a member of RECREG ([5], p. 253, Ex. 1). Thus, we have shown the following theorem:

Theorem 4.5. $\mathrm{RECREG} \subset \mathrm{HREG}$.

A more direct way to generalize $\mathcal{H}(\mathrm{REG},\mathrm{REG})$ systems is to iterate the insertion process which leads us to the definition of

$$\mathcal{H}^*(\mathrm{REG},\mathrm{REG}) = \bigcup_{n=0}^{\infty} \mathcal{H}^n(\mathrm{REG},\mathrm{REG}),$$

where $\mathcal{H}^0(\mathrm{REG},\mathrm{REG}) = \mathrm{REG}$ and

$$\mathcal{H}^n(\mathrm{REG},\mathrm{REG}) = \{\, L(H) \mid H = (\Sigma_1,\Sigma_2,L_1,\varphi) \text{ with } L_1 \text{ in } \mathcal{H}^{n-1}(\mathrm{REG},\mathrm{REG}) \text{ and } \varphi(a) \text{ in REG for all } a \in \Sigma_1\}$$

if $n \geq 1$. At first glance we show that $\mathcal{H}^*(\mathrm{REG}, \mathrm{REG})$ is sandwiched in between $\mathcal{H}(\mathrm{REG}, \mathrm{REG})$ and HREG.

Theorem 4.6. $\mathcal{H}(\mathrm{REG}, \mathrm{REG}) \subset \mathcal{H}^*(\mathrm{REG}, \mathrm{REG}) \subseteq \mathrm{HREG}$.

Proof. The first inclusion is obvious and its strictness is seen as follows. By Albert and Wegner [2] it was shown that the language $\{ (a^n b)^m \# (a^n b)^m \mid n, m \geq 1 \} \notin \mathcal{H}(\mathrm{REG}, \mathrm{REG})$. The reader may verify, that the H-system $H = (\{B\}, \{a, b\}, L_1, \varphi)$ with the $\mathcal{H}(\mathrm{REG}, \mathrm{REG})$ meta-language $L_1 = \{ B^m \# B^m \mid m \geq 1 \}$ and the regular language $\varphi(B) = \{ a^n b \mid n \geq 1 \}$ describes this language.

For the inclusion $\mathcal{H}^*(\mathrm{REG}, \mathrm{REG}) \subseteq \mathrm{HREG}$ we proceed as follows. In case $n = 0$ and $n = 1$ we have already seen that $\mathcal{H}^n(\mathrm{REG}, \mathrm{REG}) \subseteq \mathrm{HREG}$. So let $n \geq 1$ and assume that $\mathcal{H}^n(\mathrm{REG}, \mathrm{REG}) \subseteq \mathrm{HREG}$ by induction hypothesis.

Let $L \in \mathcal{H}^{n+1}(\mathrm{REG}, \mathrm{REG})$. Then there is a H-system $H = (\Sigma_1, \Sigma_2, L_1, \varphi)$ with $L_1 \in \mathcal{H}^n(\mathrm{REG}, \mathrm{REG})$ and $\varphi(a) \in \mathrm{REG}$ for all $a \in \Sigma_1$ such that $L = L(H)$. We assume that $\Sigma_1 = \{a_1, \ldots, a_n\}$. By induction hypothesis there exists H-expression r_1 (r_a for $a \in \Sigma_1$, respectively) such that $L_1 = L(r_1)$ ($\varphi(a) = L(r_a)$, respectively). Because $\Sigma_1 \cap \Sigma_2 = \emptyset$ it is easy to see that the H-expression

$$\Big(\Big(\ldots \Big(\big(r_1 \Leftarrow_{a_1} r_{a_1} \big) \Leftarrow_{a_2} r_{a_2} \Big) \ldots \Big) \Leftarrow_{a_n} r_{a_n} \Big)$$

exactly describes language L. This shows that $L \in \mathrm{HREG}$. □

Recently a particular extension of regular expressions and patterns so called pattern expressions were investigated by Câmpeanu and Yu [6]. For readability we slightly adapt their notation. Pattern expressions are based on regular patterns which are defined as follows:

Definition 4.7. Let Σ and V be two disjoint alphabets. A regular expression over $\Sigma \cup V$ is called a *regular pattern* over Σ with variables from V. The language associated with a regular pattern r over $\Sigma \cup V$ is the language $L(r) \subseteq (\Sigma \cup V)^*$.

Next we define pattern expressions:

Definition 4.8. Let Σ and V be two disjoint alphabets with $V = \{x_0, x_1, \ldots, x_n\}$. A pattern expression p over Σ with variables from V is a finite set of equations of the form $x_i = p_i$, for each $0 \leq i \leq n$, where $x_i \in V$ is a variable and p_i is a regular pattern over Σ with variables from $\{x_{i+1}, \ldots, x_n\}$.

The language of the pattern expression p is defined as

$$L(p) = \Big(\Big(\ldots \Big(\big(L(p_0) \Leftarrow_{x_1} L(p_1) \big) \Leftarrow_{x_2} L(p_2) \Big) \ldots \Big) \Leftarrow_{x_n} L(p_n) \Big)$$

and the family of languages described by pattern expressions is abbreviated by PATEXP.

Observe that from the definition of pattern expressions it follows that the last regular pattern (at least p_n) is always a regular expression.

If there is no danger of confusion we simply write $p = (p_0, x_1 = p_1, \ldots, x_n = p_n)$ to denote the regular pattern expression p described by the finite set of equations $\{x_0 = p_0, x_1 = p_1, \ldots, x_n = p_n\}$ over Σ with variables from $V = \{x_0, x_1, \ldots, x_n\}$.

New we show that pattern expressions exactly describe the languages from the family $\mathcal{H}^*(\mathrm{REG}, \mathrm{REG})$ and *vice versa*.

Theorem 4.9. $\mathcal{H}^*(\mathrm{REG}, \mathrm{REG}) = \mathrm{PATEXP}$.

Proof. The inclusion from left to right is seen by induction on n. In case $n = 0$ and $n = 1$ obviously, $\mathcal{H}^n(\mathrm{REG}, \mathrm{REG}) \subseteq \mathrm{PATEXP}$. So let $n \geq 1$ and assume by induction hypothesis that $\mathcal{H}^n(\mathrm{REG}, \mathrm{REG}) \subseteq \mathrm{PATEXP}$.

Let $L \in \mathcal{H}^{n+1}(\mathrm{REG}, \mathrm{REG})$. Then there is a H-system $H = (\Sigma_1, \Sigma_2, L_1, \varphi)$ with $L_1 \in \mathcal{H}^n(\mathrm{REG}, \mathrm{REG})$ and $\varphi(a) \in \mathrm{REG}$ for all $a \in \Sigma_1$ such that $L = L(H)$. We assume that $\Sigma_1 = \{a_1, \ldots, a_s\}$. By induction hypothesis there exists a pattern expression $p = (p_0, x_1 = p_1, \ldots, x_m = p_m)$ over Σ_1 with variables from $\{x_0, x_1, \ldots, x_m\}$, for some m, such that $L_1 = L(p)$. Moreover, since $\varphi(a)$ is regular for all $a \in \Sigma_1$ we find regular patterns q_a over Σ_2 with *no* variables such that $\varphi(a) = L(q_a)$. Because $\Sigma_1 \cap \Sigma_2 = \emptyset$ it is easy to see that the pattern expression

$$p' = (p_0, x_1 = p_1, \ldots, x_m = p_m, a_1 = q_{a_1}, \ldots, a_s = q_{a_s})$$

exactly describes language L since

$$\begin{aligned} L &= \Big(\Big(\ldots \Big(\big(L_1 \Leftarrow_{a_1} \varphi(a_1) \big) \Leftarrow_{a_2} \varphi(a_2) \Big) \ldots \Big) \Leftarrow_{a_s} \varphi(a_s) \Big) \\ &= \Big(\Big(\ldots \Big(\big(L(p) \Leftarrow_{a_1} L(q_{a_1}) \big) \Leftarrow_{a_2} L(q_{a_2}) \Big) \ldots \Big) \Leftarrow_{a_s} L(q_{a_s}) \Big) \\ &= L(p'). \end{aligned}$$

This shows that $\mathcal{H}^n(\mathrm{REG}, \mathrm{REG}) \subseteq \mathrm{PATEXP}$ for each $n \geq 0$.

Next consider $\mathrm{PATEXP} \subseteq \mathcal{H}^*(\mathrm{REG}, \mathrm{REG})$. This inclusion is shown by induction on the number of variables used in a pattern expression. The base cases $n = 0$ and $n = 1$ are trivial and left to the reader. So let $n \geq 1$ and assume by induction that hypothesis that for every pattern expression p using n variables belongs to $\mathcal{H}^*(\mathrm{REG}, \mathrm{REG})$.

Let $L \in \mathrm{PATEXP}$ be a language described by a pattern expression $p = (p_0, x_1 = p_1, \ldots, x_n = p_n)$ over Σ using variables from $\{x_0, x_1, \ldots, x_n\}$. Consider the pattern expression not using variable x_n, *i.e.*, the expression

$$p' = (p_0, x_1 = p_1, \ldots, x_{n-1} = p_{n-1})$$

over $\Sigma \cup \{x_n\}$ using variables $\{x_0, x_1, \ldots, x_{n-1}\}$. By induction hypothesis there exists a H-system $H = (\Sigma_1, \Sigma \cup \{x_n\}, L_1, \varphi)$ with $L_1 \in \mathcal{H}^m(\mathrm{REG}, \mathrm{REG})$, for

some m, and $\varphi(a) \in \mathrm{REG}$ for all $a \in \Sigma_1$, such that $L(p') = L(H)$. In order to get rid of the letter x_n in the words of L we have to replace them by words from $L(p_n)$. Since it is required that the meta- and terminal language have to be disjoint we define the two H-systems as follows. Let $\Sigma' = \{\, a' \mid a \in \Sigma \,\}$ with $\Sigma \cap \Sigma' = \emptyset$ and assume that x'_n is a new variable not contained in $\{x_0, x_1, \ldots, x_n\}$. Define $H_1 = (\Sigma \cup \{x_n\}, \Sigma' \cup \{x'_n\}, L(H), \varphi_1)$ with $\varphi_1(a) = a'$ if $a \in \Sigma$ and $\varphi_1(x_n) = x'_n$ otherwise. Finally define $H_2 = (\Sigma' \cup \{x'_n\}, \Sigma, L(H_1), \varphi_2)$ with $\varphi_2(a') = a$ if $a' \in \Sigma'$ and $\varphi(x'_n) = L(p_n)$. By easy calculations one sees that $L = L(H_2)$ which proves our claim. Hence, $\mathrm{PATEXP} \subseteq \mathcal{H}^*(\mathrm{REG}, \mathrm{REG})$. □

5. Closure and non-closure properties

In this section we study some closure properties of the classes HREG and EHREG. We find that the family HREG is *not* a TRIO. First, we start our investigations with a fairly easy theorem.

Theorem 5.1. *The language families* HREG *and* EHREG *are closed under homomorphisms, reversal, union, concatenation, and Kleene star.*

Proof. The closure under union, concatenation, and Kleene star is trivial, and the closure under reversal may be easily seen by induction on H- and EH-expressions, respectively. The details are left to the reader.

For the closure under homomorphism we do as follows: let r be an EH-expression over Σ and $h : \Sigma^* \to \Sigma^*$ a homomorphism. We construct an expression r' over Σ such that $L(r') = h(L(r))$ holds.

By induction on r we argue in the following way. If r is of the form $\emptyset$ (λ, a, for some $a \in \Sigma$, respectively), then $r' = \emptyset$, ($r' = \lambda$, $r' = a_1 + \cdots + a_n$ if $h(a) = a_1 \ldots a_n$, for $a_i \in \Sigma$ and $1 \leq i \leq n$, respectively). In case $r = s + t$ ($r = st$, $r = s^*$, respectively), then by induction hypothesis, there exists s' and t' such that $L(s') = h(L(s))$ and $L(t') = h(L(t))$. Thus, we set $r' = s' + t'$ ($r' = s't'$, $r' = (s')^*$, respectively). Finally, if $r = (s \Leftarrow_a t)$ ($r = s^{\Leftarrow_a}$, respectively), then by induction hypothesis again, there exists s' and t' such that $L(s') = h'(L(s))$ and $L(t') = h(L(t))$, where $h'(b) = h(b)$ if $b \in \Sigma \setminus \{a\}$ and $h'(b) = a$ otherwise. Then, we set $r' = (s' \Leftarrow_a t')$ ($r' = s'^{\Leftarrow_a}$, respectively). This completes the construction and shows that the language families HREG and EHREG are closed under homomorphism. □

Next we consider closure under intersection with regular sets. The below given argument re-proves, in passing, also intersection closure of the family REG.

Theorem 5.2. *The family* HREG *is closed under intersection with regular languages.*

Proof. Let r be an H-expression and R a regular language over Σ. Then there exists a finite monoid $(M,\cdot)$, a homomorphism $h : \Sigma^* \to M$, and a set $F \subseteq M$, such that $w \in R$ if and only if $h(w) \in F$.

For $m \in M$ let $[m]$ denote the set $\{ w \in \Sigma^* \mid h(w) = m \}$, which is regular for any $m \in M$. Because of $R = \cup_{m\in F}[m]$, it sufficient to construct an expression r' over Σ such that $L(r') = L(r) \cap [m]$ for some $m \in M$. To this end we perform induction on r.

If r is of the form $\emptyset$ (λ, a, for $a \in \Sigma$, respectively), then set $r' = \emptyset$ ($r' = \lambda$ if $\lambda \in [m]$ and $r' = \emptyset$ otherwise, $r' = a$ if $a \in [m]$ and $r' = \emptyset$ otherwise, respectively). In case $r = s + t$, we set $r' = s' + t'$, where s' (t', respectively) is an H-expression such that $L(s') = L(s) \cap [m]$ ($L(t') = L(t) \cap [m]$, respectively), which exist by induction hypothesis. If $r = st$ or $r = s^*$, then we do as follows. Note, that by induction hypothesis again, there are H-expressions s'_{m_1} (t'_{m_2}, respectively), for $m_1, m_2 \in M$, with $L(s'_{m_1}) = L(s) \cap [m_1]$ ($L(t'_{m_2}) = L(t) \cap [m_2]$, respectively). Now in the former case, *i.e.*, $r = st$, we set

$$r' = \sum_{m=m_1\cdot m_2} (s'_{m_1} t'_{m_2}).$$

In the latter case, *i.e.*, $r = s^*$, we generalize the above given argument. Consider the language $L = \{ m = m_1 \dots m_n \mid m_1 \cdot \ldots \cdot m_n \in M \}$ over M^*. Obviously, L is regular, therefore there exists an equivalent regular expression over M. Now, we can describe r' by taking this regular expression and substitute s_{m_i}, for each $m_i \in M$, in that particular expression. As in the previous case, the reader may verify that the constructed r' satisfies $L(r') = L(r) \cap [m]$.

Finally consider $r = (s \Leftarrow_a t)$. By induction hypothesis, there exist expressions s'_{m_1,m_2}, for $m_1, m_2 \in M$, with $L(s'_{m_1,m_2}) = L(s) \cap [m_1, m_2]$, where $[m_1, m_2]$ equals the equivalence class $[m_1]$ of the regular language R', which is defined as R, *i.e.*, *via* the monoid M and the set $F \subseteq M$, except that we alter the homomorphism h on letter a such that $h(a) = m_2$. Moreover, we also have expressions t'_{m_3}, for $m_3 \in M$, such that $L(t'_{m_3}) = L(t) \cap [m_3]$. Putting all things together, expression r' reads as

$$r' = \sum_{m_1\in M} \left(s'_{m,m_1} \Leftarrow_a t'_{m_1}\right).$$

This completes our construction. □

Finally, on the remaining TRIO operation inverse homomorphism we also get a non-closure result for H-expression languages.

Theorem 5.3. *The family* HREG *is not closed under inverse homomorphisms.*

Proof. Consider the H-expression $r = (A\#A) \Leftarrow_A a^*$, which describes the language $\{ a^n\#a^n \mid n \geq 0 \}$. Define two homomorphisms $g : \{a, b, \#\}^* \to \{a, b\}^*$ and homomorphism $h : \{a, b, \#\}^* \to \{a, b\}^*$ as follows: $g(a) = a$, $g(b) = b$, and $g(\#) = \lambda$. Moreover, set $h(a) = a$, $h(b) = a$, and $h(\#) = \#$. Then $g(h^{-1}(L(r)) \cap a^*\#b^*)$ equals $\{ a^nb^n \mid n \geq 0 \}$, which does not belong to the family

TABLE 1. Closure properties of some language families.

Operations	Language family			
	REG = SREG	HREG	EHREG	CFL = ESREG
Union	Yes	Yes	Yes	Yes
Homomorphism	Yes	Yes	Yes	Yes
Inverse homomorphism	Yes	No	?	Yes
Intersection with regular sets	Yes	Yes	?	Yes
Concatenation	Yes	Yes	Yes	Yes
Kleene star	Yes	Yes	Yes	Yes
Reversal	Yes	Yes	Yes	Yes

HREG by Theorem 4.4. Since H-expressions are closed under homomorphism and intersection with regular languages, our claim follows. □

For completeness we summarize the closure properties of some considered language families in Table 1. Unfortunately, at this point it remains open whether the family EHREG is closed under intersection with regular languages and inverse homomorphisms. The non-closure under the TRIO operations destroys the hope to get a *nice* characterization of HREG languages in terms of an one-way automaton model. This is because most automata in formal language theory as, *e.g.*, pushdown automata, stack automata, queue automata, can be characterized in terms of automata with abstract storage. As shown by Dassow and Lange [7] automata with abstract storage imply a Chomsky-Schützenberger like theorem of the described language family, *i.e.*, every language from the family can be written as $h(g^{-1}(D) \cap R)$, where g and h are homomorphisms, R is a regular language, and D is protocol language of the abstract storage type.

6. Complexity theoretical issues

In this section we study some complexity theoretical problems for H- and EH-expressions. We start with the fixed membership problem, showing that it is NL-complete for both H- and EH-expression languages. The below given theorem nicely contrasts the NC^1-completeness for ordinary regular languages [4].

Theorem 6.1. *The fixed membership problem for H- and EH-expressions is* NL-*complete.*

Proof. The fixed membership problem for EDT0L systems is known to be NL-complete [18]. Since, by Theorem 3.10 we have EHREG $\subseteq$ EDT0L, the fixed membership problem for both H- and EH-expressions is in NL, too. In order to prove NL-hardness, we reduce some special case of the graph accessibility problem, which is known to be NL-complete (see, *e.g.*, [14]) to the fixed membership problem

for H-expressions. This problem is defined as follows: given an ordered directed graph $G = (V, E)$ with out-degree two, where $V = \{1, 2, \ldots, n\}$ is the set of nodes, $E \subseteq V \times V$ is the set of edges, and (i, j) in E implies that $i \leq j$. Is there a path from node 1 to node n in G?

The below given construction follows the lines of Sudborough [28]. Let

$$1\#\#1\$1^{j_{11}}\#\#1\$1^{j_{12}}\#\#1^2\$1^{j_{21}}\#\#1^2\$1^{j_{22}}\#\ldots\#1^n\$1^{j_{n1}}\#\#1^n\$1^{j_{n2}}\#\#1^n,$$

be the coding of the graph G, where (i, j_{i1}) and (i, j_{i2}) are edges in E. The graph accessibility problem for G is reduced to the fixed membership problem for the expression

$$r = \left(\left(a\#\left(\#1^+\$1^+\#\right)^*\#a\$\right) \Leftarrow_a 1^+\right)^*$$

over $\Sigma = \{0, 1, a, \#, \$\}$.

Obviously, the coding of G can be computed in logarithmic space. In words of $L(r)$, one subword of $L(s)$, where

$$s = \left(a\#\left(\#1 + \$1^+\#\right)^*\#a\$\right) \Leftarrow_a 1^+,$$

corresponds to one block between two markers, more precisely beginning with the second part of a marked couple and ending with the first part of the next marked couple. Therefore, it is easily seen that the coding of G belongs to $L(r)$ if and only if there is a (ordered) path from 1 to n in G. This proves our claim. □

In the next theorem we turn our attention to the general membership problem. There we were not able to exactly characterize its complexity, and we can only give some lower and upper bound.

Theorem 6.2. *The general membership problem both for H- and EH-expressions is* NP*-hard and belongs to* PSPACE*.*

Proof. Analogously to the argument in the proof of Theorem 6.1, the containment in PSPACE is inherited from the general membership problem for EDT0L systems [19].

For lower bound, it is sufficient to reduce the well-known NP-complete satisfiability problem for Boolean formulas in conjunctive normal form (SAT) to the general membership problem for H-expressions. Let a Boolean formula $f = C_1 \wedge C_2 \wedge \ldots \wedge C_m$, for some $m \geq 1$, be given, where C_i, for $1 \leq i \leq m$, is a disjunction of variables or negated variables from $\{x_1, \ldots, x_n\}$.

From f we compute an instance for the general membership problem of H-expressions as follows: first set for $1 \leq i \leq m$ the H-expressions

$$r_i = \sum_{x_j \text{ is in } C_i} x_j + \sum_{\bar{x}_j \text{ is in } C_j} \bar{x}_j$$

over the alphabet $\{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\}$. Then let

$$s_0 = x_1\bar{x}_1\#x_2\bar{x}_2\#\ldots\#x_n\bar{x}_n\#\$r_1\#r_2\#\ldots\#r_m\#$$

and inductively define

$$s_{i+1} = \Big(\big(s_i \Leftarrow_{x_{i+1}} (\lambda + 1)\big) \Leftarrow_{\bar{x}_{i+1}} (\lambda + 1)\Big),$$

for $0 \leq i < n$, over the alphabet $\Sigma = \{x_1, x_2, \ldots, x_n, \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n, \#, \$, 1\}$. Finally, let $\langle s_n, w\rangle$ be the instance of the general membership problem for H-expressions, where $w = (1\#)^n\$(1\#)^m$.

Clearly, the above specified instance is computed in logarithmic space from a suitable description of f. Moreover, to each literal of the form x_i occurring in f a Boolean value is assigned by replacing it consistently by 1 (λ, respectively) corresponding to *true* (*false*, respectively). Analogously, to each literal of the form $\bar{x}_i$ occurring in f a Boolean value is assigned. After these replacements, the string w belongs to $L(s_n)$ if and only if (1) the Boolean assignment is a correct one, *i.e.*, where x_i and $\bar{x}_i$ evaluate not equally, for $1 \leq i \leq n$, which is checked in the part left to the \$ in w and (2) each of the clauses C_i, for $1 \leq i \leq m$, is satisfiable, which is tested in the left-hand part of w. Therefore, we have w is in $L(s_n)$ if and only if f is satisfiable. □

The next theorem holds trivially.

Theorem 6.3. *Let r be an H-expression (EH-expression, respectively) and let r' be the S-expression (ES-expression, respectively) obtained from r by replacing every $\Leftarrow$ by $\leftarrow$ (and every $\overset{\Leftarrow}{}$ by $\overset{\leftarrow}{}$) and vice versa. Then $L(r) = \emptyset$ if and only if $L(r') = \emptyset$.* □

We use the above given theorem to prove that the emptiness problem for H- and EH-expression is P-complete.

Theorem 6.4. *The emptiness problem for both H- and EH-expressions is P-complete.*

Proof. Given an ES-expression r, one can construct an equivalent context-free grammar by induction on r, mainly following the idea given in [13], Theorem 2.7. This construction can be done in deterministic logarithmic space. Therefore, the emptiness problem for ES-expressions is not harder then the emptiness problem for context-free grammars, *i.e.*, it can be solved in polynomial time by a deterministic Turing machine [17]. Due to Theorem 6.3, even the emptiness problem for EH-expressions and hence for H-expressions can be solved within this time bound. This proves the containments in P.

In order to show P-hardness, it is sufficient to reduce the P-complete emptiness problem for context-free grammars to the emptiness problem for H-expressions or, due to Theorem 6.3, for S-expressions. The completeness for EH-expressions (ES-expressions, respectively) follows trivially, because every H-expressions (S-expression, respectively) is also an EH-expression (ES-expression).

Let $G = (N, T, P, S)$ be a context-free grammar with $N = \{A_1, \dots A_n\}$ and assume $S = A_1$. Define the homomorphism $h : (N \cup T)^* \to N^*$ as $h(A) = A$ if $A \in N$ and $h(a) = \lambda$ otherwise. Furthermore, for $A \in N$ let s_A denote the H-expressions (S-expression) with $L(s_A) = \{ h(\alpha) \mid A \to \alpha \text{ is in } P \}$.

Then let $r_0 = s_{A_1}$, inductively for $0 \leq i < n$ define

$$r_{i+1} = \left(\left(\dots \left(\left(r_i \Leftarrow_{A_1} s_{A_1} \right) \Leftarrow_{A_2} s_{A_2} \right) \dots \right) \Leftarrow_{A_n} s_{A_n} \right),$$

and let $r_{n+1} = ((\dots((r_i \Leftarrow_{A_1} \emptyset) \Leftarrow_{A_2} \emptyset) \dots) \Leftarrow_{A_n} \emptyset)$. By induction the reader may verify that $L(r_{n+1}) = \emptyset$ if and only if $L(G) = \emptyset$. Since the s_{A_i} expressions and thus also the r_i expressions, in particular the r_{n+1} expression, are deterministic logarithmic space constructible from G, we conclude that the emptiness problem for H- and EH-expressions is P-hard, too. □

Finally, we consider the equivalence problem for EH-expressions.

Theorem 6.5. *The equivalence problem for EH-expressions is undecidable.*

Proof. Given an instance of Post's correspondence problem, *i.e.*, two n-tuples of words $U = (u_1, u_2, \dots, u_n)$ and $V = (v_1, v_2, \dots, v_n)$ over the alphabet $\{a, b\}$ for a sufficiently large integer n. We associate two EH-expressions r and $t = (r + s)$ with this instance, where

$$\begin{aligned} r = & \left(\left(\left(aAa + bAb + aBb + bBa \right)^{\Leftarrow_A} \right) \Leftarrow_B \left(\left(a + b \right)^* \# \left(a + b \right)^* \right) \right) \\ & + \left(\left(\left(aAa + bAb + aB + bB \right)^{\Leftarrow_A} \right) \Leftarrow_B \left(\left(a + b \right)^* \# \right) \right) \\ & + \left(\left(\left(aAa + bAb + Ba + Bb \right)^{\Leftarrow_A} \right) \Leftarrow_B \left(\# \left(a + b \right)^* \right) \right), \end{aligned}$$

and

$$s = \left(\sum_{i=1}^{n} (u_i A v_i^R + u_i \# v_i^R) \right)^{\Uparrow_A}.$$

Then

$$L(r) = \{ x \# y \mid x, y \in \{a, b\}^* \text{ and } y^R \neq x \}$$

and

$$L(s) = \{ u_{i_1} u_{i_2} \dots u_{i_k} \# v_{i_k}^R \dots v_{i_2}^R v_{i_1}^R \mid k \geq 1 \}.$$

Clearly, if Post's correspondence problem has *no solution* for the instance U and V, then $L(s) \subseteq L(r)$ and the expressions r and t are equivalent. On the other hand, if *there is a solution*, then there is a string of the form $x \# x^R$ in $L(s) \subseteq L(t)$ which does not belong to $L(r)$, and the two expressions r and t are not equivalent. Since Post's correspondence problem is undecidable (see, *e.g.*, [16]) the equivalence problem for EH-expressions is undecidable, too. □

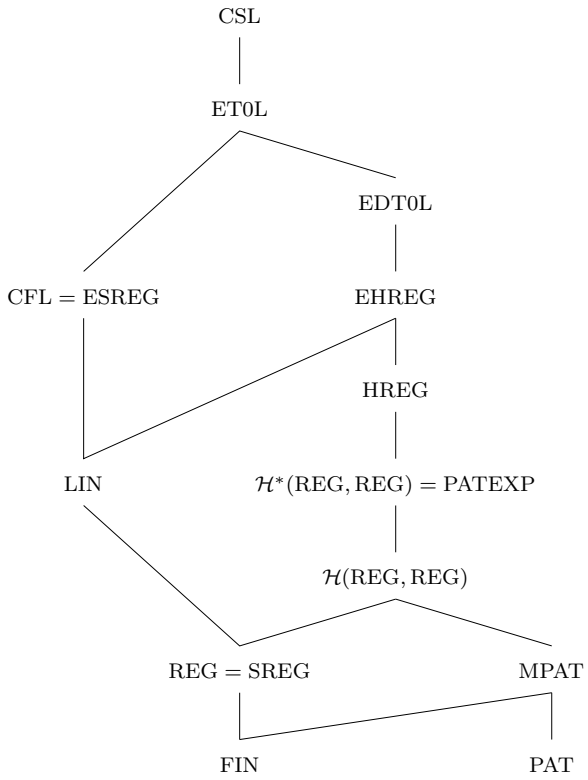

FIGURE 1. The inclusion structure of the considered language families.

The decidability status of the equivalence problem for H-expressions remains open.

7. CONCLUSIONS

In this paper we have studied the expressive power of H- and EH-expressions, which are defined as an extension of regular expressions by homomorphic and iterated homomorphic replacement. The inclusion relations among the classes considered are depicted in Figure 1. Besides the expressive power we have also investigated the closure and non-closure properties of these classes under Boolean operations, Kleene star, and TRIO operations. In most cases we classified the problems under consideration completely. Nevertheless, we left some problems open, such as whether the family of EH-expression languages is closed under intersection with regular languages and inverse homomorphism. Moreover, we also focused on some issues of computational complexity as the fixed and general membership, non-emptiness, and equivalence. The decidability status of the equivalence problem for H-expression languages remains open.

We hope that the investigation of homomorphic replacement, as one sort of pattern repeating operation, helps understand the expressive power of regular-like expressions much better. Nevertheless, regular like expressions in programming environments still lack complete theoretical understand.

References

[1] A.V. Aho, Algorithms for finding patterns in strings, in *Handbook of Theoretical Computer Science*, edited by J. van Leeuwen. Elsevier (1990) 255–300.

[2] J. Albert and L. Wegner, Languages with homomorphic replacements, in *Proceedings of the 7th International Colloquium on Automata Languages and Programming. Lect. Notes Comput. Sci.* **85** (1980) 19–29.

[3] J.L. Balcázar, J. Díaz and J. Gabarró, Structural Complexity I. *EATCS Monographs on Theoretical Computer Science*, Vol. **11**. Springer (1988).

[4] D.A. Barrington, Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. System Sci.* **38** (1989) 150–164.

[5] J.-C. Birget and J.B. Stephen, Formal languages defined by uniform substitutions. *Theoret. Comput. Sci.* **132** (1994) 243–258.

[6] C. Câmpeanu and S. Yu, Pattern expressions and pattern automata. *Inform. Process. Lett.* **92** (2004) 267–274.

[7] J. Dassow and K.-J. Lange, Complexity of automata with abstract storages, in *Proceedings of the 8th International Conference on Fundamentals of Computation Theory. Lect. Notes Comput. Sci.* **529** (1991) 200–209.

[8] J. Dassow and Gh. Păun, Regulated Rewriting in Formal Language Theory. *EATCS Monographs in Theoretical Computer Science*, Vol. **18**. Springer (1989).

[9] P. Dembiński and J. Małuszyński, Two level grammars: CF-grammars with equation schemes, in *Proceedings of the 6th International Colloquium on Automata Languages and Programming. Lect. Notes Cumput. Sci.* **71** (1979) 171–187.

[10] D. Dougherty, *sed & awk*. O'Reilly & Associates (1990).

[11] J. Engelfriet and E.M. Schmidt, IO and OI. Part I and II. *J. Comput. System Sci.* **15** (1977) 328–353; *J. Comput. System Sci.* **16** (1977) 67–99.

[12] D. Giammarresi and A. Restivo, Two-dimensional languages, in *Handbook of Formal Languages*, Vols. **1–3**, edited by G. Rozenberg and A. Salomaa. Springer (1997) 215–267.

[13] J. Gruska, A characterization of context-free languages. *J. Comput. System Sci.* **5** (1971) 353–364.

[14] J. Hartmanis, N. Immerman and S. Mahaney, One-way log-tape reductions, in *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*. IEEE Society Press, Ann Arbor, Michigan (1978) 65–72.

[15] K. Hashiguchi and H. Yoo, Extended regular expressions of degree at most two. *Theoret. Comput. Sci.* **76** (1990) 273–284.

[16] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley (1979).

[17] N.D. Jones and W.T. Laaser, Complete problems for deterministic polynomial time. *Theoret. Comput. Sci.* **3** (1977) 105–117.

[18] N.D. Jones and S. Skyum, Recognition of deterministic ET0L languages in logarithmic space. *Inform. Comput.* **35** (1977) 177–181.

[19] N.D. Jones and S. Skyum, Complexity of some problems concerning L systems. *Math. Syst. Theor.* **13** (1979) 29–43.

[20] L. Kari, A. Mateescu, Gh. Păun and A. Salomaa, Multi-pattern languages. *Theoret. Comput. Sci.* **141** (1995) 253–268.

[21] S.C. Kleene, Representation of events in nerve nets and finite automata, in *Automata studies*, *Annals of mathematics studies*, Vol. **34**, edited by C.E. Shannon and J. McCarthy. Princeton University Press (1956) 2–42.
[22] A. Mateescu and A. Salomaa, Aspects of classical language theory, in *Handbook of Formal Languages*, Vols. **1–3**, edited by G. Rozenberg and A. Salomaa. Springer (1997) 175–251.
[23] E. Ochmanski, Regular behaviour for concurrent processes. *Bulletin of the European Association for Theoretical Computer Science* **27** (1985) 56–67.
[24] G. Della Penna, B. Intrigilla, E. Tronci and M. Venturini Zilli, Synchronized regular expressions. *Acta Informatica* **39** (2003) 31–70.
[25] G. Rozenberg and A. Salomaa, *The Mathematical Theory of L Systems*, *Pure and Applied Mathematics*, Vol. **90**. Academic Press (1980).
[26] Edited by G. Rozenberg and A. Salomaa, *Handbook of Formal Languages*, Vols. **1–3**. Springer (1997).
[27] R. Siromoney and K. Krithivasan, Parallel context-free grammars. *Inform. Control.* **24** (1974) 155–162.
[28] I.H. Sudborough, A note on tape-bounded complexity classes and linear context-free languages. *J. ACM* **22** (1975) 499–500.
[29] M.K. Yntema, Cap expressions for context-free languages. *Inform. Control.* **18** (1971) 311–318.

Communicated by C. Choffrut.
Received June 5, 2009. Accepted November 19, 2009.