T U M

I N S T I T U T F Ü R I N F O R M A T I K

Extending Regular Expressions with Homomorphic Replacement

Henning Bordihn and Jürgen Dassow and Markus Holzer

TUM-I0102
August 01

T E C H N I S C H E U N I V E R S I T Ä T M Ü N C H E N

TUM-INFO-08-I0102-0/1.-FI

Druck: Institut für Informatik der
Technischen Universität München

Extending Regular Expressions with Homomorphic Replacement*

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

[1] Institut für Informatik, Universität Potsdam,
Am Neuen Palais 10, D-14469 Postdam, Germany
email: bordihn@cs.uni-potsdam.de

[2] Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg,
Postfach 4120, D-39016 Magdeburg, Germany
email: dassow@iws.cs.uni-magdeburg.de

[3] Institut für Informatik, Technische Universität München,
Arcisstraße 21, D-80290 München, Germany
email: holzer@informatik.tu-muenchen.de

Abstract. We define H-expressions and EH-expressions as extensions of regular expressions by adding homomorphic and iterated homomorphic replacement as new operations, respectively. 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. 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. Some of our proofs are only based on expressions, and thus can be used in order to get new proofs for regular sets, too.

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 [22] and are a lovely set-theoretic characterization of regular languages, are better suited for human users and therefore are often used

* An extended abstract of this paper appeared in the proceedings of the 9th Theoryday of the GI-Workinggroup 0.1.5 "Automaten und Formale Sprachen" held at Schauenburg-Elmshagen near Kassel, Germany, in September 1999 [6].

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

*** Part of the work was done while the author was at Département d'I.R.O., Université de Montréal, C.P. 6128, succ. Centre-Ville, Montréal (Québec), H3C 3J7 Canada. 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.

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*. 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 $\{ ww \mid w \in \{a,b\}^* \}$. For more details we refer to [11] and to the appendix, where we give a natural semantics for so called regular expressions with back referencing[1], a model for UNIX regular expressions, first briefly discussed in [1].

Kleene's well-known theorem, which states that a language L is regular if and only if there is a regular expression r with $L = L(r)$. There have been some attempts to generalize this theorem in one of the following directions: Define an extension of regular expressions and determine the associated family of languages (see, e.g., [16]) or find the class of expressions for a given extension of the family of regular languages (see, e.g., [13, 24, 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 [14], 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" [10]. In connection with macro grammars [12] 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."

[1] To our knowledge this was not done before.

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 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 [9]. 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**, **CF**, and **CS**, respectively. Moreover, the family of extended (extended deterministic, respectively) tabled context-free Lindenmayer languages is denoted by **ET0L** (**EDT0L**, respectively). The class 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 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 [4]. In particular we consider the following well-known chain of inclusions: $\mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSpace}$. 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 $\bigcup_k \mathbf{DSpace}(n^k)$.

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 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 [14] 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 [14] 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 $L_1 \downarrow_a L_2$, is defined by

$$\begin{aligned} L_1 \downarrow_a L_2 = \{\, u_1 v_1 u_2 \dots u_k v_k u_{k+1} \mid {} & u_1 a u_2 a \dots a u_{k+1} \in L_1, \\ & a \text{ does not occur in } u_1 u_2 \dots u_{k+1}, \text{ and } v_1, v_2 \dots, v_k \in L_2 \,\}, \end{aligned}$$

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

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

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

Based on these operations an extension of regular expressions is defined. 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, resp., denoting the languages $L(r)$ and $L(s)$, resp., then $(r + s)$, (rs), (r^*), and $(r \downarrow_a s)$, 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 $L(r) \downarrow_a L(s)$, respectively.
3. Let $a \in \Sigma$. If r is an ES-expression denoting the language $L(r)$, then $(r^{\downarrow_a})$ is an ES-expression that denotes the set $L(r)^{\downarrow_a}$.
4. Nothing else is an S- or ES-expressions, respectively.

The families of languages described by S- and ES-expressions are denoted by **SREG** and **ESREG**, respectively. While **SREG** equals **REG**, which is easily seen, In [14] Gruska has shown that **ESREG** coincides with the family **CF** 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" [10]. In connection with macro grammars [12] 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 [14]. 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 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 $L_1 \Uparrow_a L_2$, is defined by*

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

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

Lemma 1. *For each letter a, the operation $\Uparrow_a$ is associative, i.e.,*

$$(L_1 \Uparrow_a L_2) \Uparrow_a L_3 = L_1 \Uparrow_a (L_2 \Uparrow_a L_3).$$

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

$$(\{b\} \Uparrow_a \{a\}) \Uparrow_b \{a\} = \{a\} \neq \{b\} = \{b\} \Uparrow_a (\{a\} \Uparrow_b \{a\})\,.$$

We also consider the iterated version of homomorphic replacement.

Definition 3. *Let a be a letter and L a language. The* iterated a-homomorphic replacement *of L, denoted by $L^{\Uparrow_a}$, is defined by*

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

Due to Lemma 1 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 $L^* = (La \cup \{\lambda\})^{\Uparrow_a}$ and $L^+ = (La \cup L)^{\Uparrow_a}$. 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 $\{cc\} \Uparrow_c \{a,b\}^*$. In fact, this shows that the low levels of the Chomsky hierarchy are not closed under a-homomorphic and iterated a-homomorphic replacement.

Theorem 1. 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 **CS** is closed under this type of operation what can readily be shown by 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 2. *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 4. *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, resp., denoting the languages $L(r)$ and $L(s)$, resp., then $(r + s)$, (rs), (r^*), and $(r \Uparrow_a s)$, 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 $L(r) \Uparrow_a L(s)$, respectively.*
3. *Let $a \in \Sigma$. If r is an EH-expression denoting the language $L(r)$, then $(r^{\Uparrow_a})$ is an EH-expression that denotes the set $L(r)^{\Uparrow_a}$.*
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 1. 1. $cc \Uparrow_c (a + b)^*$ denotes the language $\{\, ww \mid w \in \{a, b\}^* \,\}$, which is non-context-free.
2. $(ab + aAb)^{\Uparrow_A}$ describes the non-regular language $\{\, a^n b^n \mid n \geq 1 \,\}$.
3. $(a + AA)^{\Uparrow_A}$ denotes the non-context-free language $\{\, a^{2^n} \mid n \geq 0 \,\}$.

Next, consider the following chain of inclusions:

Theorem 3. **REG** $\subset$ **HREG** $\subset$ **EHREG**.

Proof. The inclusions are obvious; the strictness of the first one is seen from Example 1. 1 and the strictness of the second inclusion follows by Example 1.3 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 4. **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 $1 \le i \le n$, we set

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

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

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

and

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

for $1 \le i \le n-1$. Then one can readily verify that $L(G_i) = L(r_i)$ for $1 \le i \le n$, 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 1.3.

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

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 5. *We define the depth of an R-expression 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* $d(r + s) = d(r \cdot s) = d(r \Uparrow_a s) = d(r) + d(s) + 1$ *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 $s \Uparrow_a u$ with $L(u) = \{\lambda\}$.

Lemma 2. *For any H-expression $r = s \Uparrow_a u$ with $L(u) = \{\lambda\}$ there is a λ-free H-expression t such that $L(t) = L(r)$ and $d(t) \le d(s)$.*

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 $r = s \Uparrow_a u$ 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 $r = s \Uparrow_a u \in K$ such that $d(r) = k$. Obviously, if $s \Uparrow_a u$ in K, then $s \Uparrow_a \lambda$ is in K, too. By the minimality of r with respect to the depth, we can assume without loss of generality that $r = u \Uparrow_a \lambda$.

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

1. If $s = \emptyset$, then $L(s \Uparrow_a \lambda) = L(\emptyset)$ and $d(\emptyset) = d(s)$.
2. If $s = \lambda$, then $L(s \Uparrow_a \lambda) = L(\lambda)$ and $d(\lambda) = d(s)$.
3. If $s = a$, then $L(s \Uparrow_a \lambda) = L(\lambda)$ and $d(\lambda) = d(s)$.
4. If $s = b$ for $b \in \Sigma \setminus \{a\}$, then $L(s \Uparrow_a \lambda) = L(b)$ 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 $t_1 = s_1 \Uparrow_a \lambda$ and $t_2 = s_2 \Uparrow_a \lambda$. 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 $L(t_1') = L(s_1 \Uparrow_a \lambda)$ and $L(t_2') = L(s_2 \Uparrow_a \lambda)$, 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
$$\begin{aligned} L(t_1' + t_2') &= L(t_1') \cup L(t_2') \\ &= L(s_1 \Uparrow_a \lambda) \cup L(s_2 \Uparrow_a \lambda) = L((s_1 \Uparrow_a \lambda) + (s_2 \Uparrow_a \lambda)) \\ &= L((s_1 + s_2) \Uparrow_a \lambda) = L(s \Uparrow_a \lambda) = L(r). \end{aligned}$$
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 $s = s_1 \Uparrow_b s_2$ 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
$$\begin{aligned} &L(t_1') = L(s_1 \Uparrow_a \lambda), \\ &L(t_2') = L(s_2 \Uparrow_a \lambda) \quad \text{with} \quad d(t_1') \leq d(s_1) \quad \text{and} \quad d(t_2') \leq d(s_2) \end{aligned} \tag{1}$$
Moreover, if $a \neq b$, then
$$\begin{aligned} L(t_1' \Uparrow_b t_2') &= L((s_1 \Uparrow_a \lambda) \Uparrow_b (s_2 \Uparrow_a \lambda)) \\ &= L((s_1 \Uparrow_b s_2) \Uparrow_a \lambda) = L(s \Uparrow_a \lambda) = L(r). \end{aligned} \tag{2}$$

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 (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

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

By the minimality of k, there is a λ-free H-expression t such that $L(t) = L(t'_1 \Uparrow_b t'_2)$ and $d(t) \leq d(t'_1)$. By (1), (2), and (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

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

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:

A1 $\tau_C(L) \subseteq a^*b^*$,
A2 $\tau_C(L)$ is infinite,
A3 for any natural number n, $D(a,n,L) = \{\, m \mid a^n b^m \in \tau_C(L) \,\}$ is a finite set, and
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 the conditions A3 and A4 are equivalent to the existence of a constant $k \geq 0$ such that $a^n b^m \in \tau_c(L)$ implies $|n - m| \leq k$.

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. *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 $L_1 \Uparrow_c L_2$ 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 $a^r b^s$ with $r \geq 1$ and $s \geq 1$, then $a^r b^s a^r b^s \in (\tau_C(L))^2 \subseteq \tau_C(L^*)$ 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 $\#_c(w) = 0$ for all $w \in L_1$, then $L_1 \Uparrow_c L_2 = L_1$ and the statement is shown. Thus, we can assume that there is a word $w \in L_1$ with $\#_c(w) \geq 1$.
   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 $\tau_C(w \Uparrow_c L_2)$ is infinite, too. Therefore there is an integer n such that $D(b, n, w \Uparrow_c L_2)$ and hence $D(b, n, L_1 \Uparrow_c L_2)$ are infinite. This contradicts condition A4 for $L_1 \Uparrow_c L_2$.
   Thus, we can assume that $\tau_C(L_2) \subseteq a^*$ is finite. We now prove that L_1 is an (a, b)-language with respect to 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 $L_1 \Uparrow_c L_2$, we obtain $\tau_D(L_1) \subseteq a^*b^*$, i.e., language L_1 satisfies condition A1. Moreover, the infinity of $\tau_C(L_1 \Uparrow_c L_2)$ 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 $\tau_C(u \Uparrow_c L_2)$ contains a word $a^{k''} b^n$ with $k'' \geq k' \geq k$. Thus, $D(b, n, L_1 \Uparrow_c L_2)$ is infinite, too, in contrast to the validity of condition A4 for $L_1 \Uparrow_c L_2$.

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 $\tau_D(w) \in a^m b^*$. 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

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

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 $D(a, |w_1 w_2 \dots w_{r+1}| + (i_1 + \dots + i_r)s, F \Uparrow_c v)$ and therefore $D(a, |w_1 w_2 \dots w_{r+1}| + (i_1 + \dots + i_r)s, L_1 \Uparrow_c L_2)$ 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 $\tau_C(w \Uparrow_c v) = u_1 a^r b^s u_2 a^r b^s u_3 \in \tau_C(L_1 \Uparrow_c L_2)$ for some words $u_1, u_2, u_3 \in \{a, b\}^*$, i.e., condition A1 does not hold for $L_1 \Uparrow_c L_2$ 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 $\#_c(w) = 1$ 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 $D(a, n + r + 1, w \Uparrow_a L_2)$ or $D(b, n + s, w \Uparrow_a L_2)$. Therefore, $D(a, n + r + 1, L_1 \Uparrow_a L_2)$ or $D(b, n + s, L_1 \Uparrow_a L_2)$ is infinite in contrast to the fact that $L_1 \Uparrow_a L_2$ 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 set $\tau_C(u \Uparrow_c L_2)$ contains a word $a^{k''} b^n$ with $k'' \geq k' \geq k$. Thus, $D(b, n, L_1 \Uparrow_c L_2)$ is infinite, too, in contrast to the validity of condition A4 for $L_1 \Uparrow_c L_2$. Analogously we prove that L_1 satisfies condition A3. □

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

Lemma 4. *Any (a, b)-language is not an* **HREG** *language.*

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

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

and let L be an (a, b)-language in **HREG** with $d(L) = k$. By Lemma 2, there is an H-expression r constructed without steps of the form $s \Uparrow_c \lambda$ such that $L(r) = L$. Then $k \geq 1$ since (a, b)-languages are infinite by condition A2. Now, by Lemma 3 there are H-expressions s and t with $d(s) < k$ and $d(t) < k$ such that $r = s+t$ or $r = st$ or $r = s \Uparrow_c t$ for some c. By Lemma 3 we obtain that $L(s)$ or $L(t)$ are (a, b)-languages in contrast to the definition of k. □

Theorem 5. *Let $X \in \{\mathbf{CF}, \mathbf{LIN}\}$. Then the family of languages X is incomparable to the family* **HREG**.

Proof. By Theorem 3 it is sufficient to show that there is are languages $K_1 \in \mathbf{LIN} \setminus \mathbf{HREG}$ and $K_2 \in \mathbf{HREG} \setminus \mathbf{CF}$. Obviously, the linear context-free language $K_1 = \{\, c^n d^n \mid n \geq 1 \,\}$ is an (a, b)-language. Thus, $K_1 \notin \mathbf{HREG}$ follows from Lemma 4. 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, page 205], and thus is not contained in **HREG** by Theorem 4. This proves the following corollary.

Corollary 1. *The language families* **CF** *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 6 (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* $\mathcal{H}(\mathbf{REG}, \mathbf{REG})$*.*

Recently a restricted form of homomorphic replacement systems, so called pattern or multi-pattern languages [21, 23] 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 $\mathcal{H}(\mathbf{FIN}, \mathbf{REG})$, the family of H-system languages with finite meta-languages and regular languages L_a for every $a \in \Sigma_1$. Because the $\mathcal{H}(\mathbf{REG}, \mathbf{REG})$ 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 $\mathcal{H}(\mathbf{FIN}, \mathbf{REG})$, which was shown in [2], we obtain the following theorem, where the first strict inclusion is due to [21]:

Theorem 6. $\mathbf{PAT} \subset \mathbf{MPAT} \subset \mathcal{H}(\mathbf{REG}, \mathbf{REG})$. □

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

Theorem 7. $\mathbf{REG} \subset \mathcal{H}(\mathbf{REG}, \mathbf{REG}) \subset \mathbf{HREG}$.

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 $L \in \mathcal{H}(\mathbf{REG}, \mathbf{REG})$. 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 $a \in \Sigma_1$, resp.) is regular there exists a regular expression r_1 (r_a for $a \in \Sigma_1$, resp.) such that $L_1 = L(r_1)$ ($\varphi(a) = L(r_a)$, resp.). Because $\Sigma_1 \cap \Sigma_2 = \emptyset$ it is easy to see that the H-expression

$$\left(\left(\dots\left((r_1 \Uparrow_{a_1} r_{a_1}) \Uparrow_{a_2} r_{a_2}\right)\dots\right) \Uparrow_{a_n} r_{a_n}\right)$$

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

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

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

The reader may verify, that the H-expression

$$\left((A\#A) \Uparrow_A \left(B^+ \Uparrow_B (a^+ b)\right)\right) \quad \text{or} \quad \left(\left((A\#A) \Uparrow_A B^+\right) \Uparrow_B (a^+ b)\right)$$

describes this language. Thus, the claim follows. □

We want to stress that Theorem 5 can be generalized as follows. We state the result without proof.

Theorem 8. *Let $X \in \{\mathbf{CF}, \mathbf{LIN}\}$ and $Y \in \{\mathbf{HREG}, \mathcal{H}(\mathbf{REG}, \mathbf{REG})\}$. Then the family of languages X is incomparable to the family of languages Y.* □

A slightly more general class than $\mathcal{H}(\mathbf{REG}, \mathbf{REG})$ was introduced and investigated by Birget and Stephen [5]. They define a *uniform sustitution* 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 $\Sigma_1 = \{v_1, \dots, v_n\}$ the set $\{\varphi(v_1)\# \dots \#\varphi(v_n) \in (\Sigma_2 \cup \{\#\})^* \mid \varphi \in H \,\}$ is a regular subset of $(\Sigma_2 \cup \{\#\})^*$, where $\#$ is a symbol not in Σ_2. By Mezei's theorem, see, e.g., [5, page257, Theorem A.1], the set $\{\varphi(v_1)\# \dots \#\varphi(v_n) \in (\Sigma_2 \cup \{\#\})^* \mid \varphi \in H \,\}$ is regular if and only if it is equal to a finite union of sets of the form $L_1\# \dots \#L_n$, 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 $\mathcal{H}(\mathbf{REG}, \mathbf{REG})$ from **HREG** is also not a member of **RecREG** [5, page 253, Example 1]. Thus, we have shown the following theorem:

Theorem 9. **RecREG** $\subset$ **HREG**. □

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

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

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

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

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

Theorem 10. $\mathcal{H}(\mathbf{REG}, \mathbf{REG}) \subset \mathcal{H}^*(\mathbf{REG}, \mathbf{REG}) \subseteq \mathbf{HREG}$.

Proof. The first inclusion is obvious and its strictness is seen as follows. By Albert and Wegner [2] it was shown that the language $\{\, (a^n b)^m \#(a^n b)^m \mid n, m \geq 1 \,\} \notin \mathcal{H}(\mathbf{REG}, \mathbf{REG})$. The reader may verify, that the H-system $H = (\{B\}, \{a, b\}, L_1, \varphi)$ with the $\mathcal{H}(\mathbf{REG}, \mathbf{REG})$ 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 $\mathcal{H}^*(\mathbf{REG}, \mathbf{REG}) \subseteq \mathbf{HREG}$ we proceed as follows. In case $n = 0$ and $n = 1$ we have already seen that $\mathcal{H}^n(\mathbf{REG}, \mathbf{REG}) \subseteq \mathbf{HREG}$. So let $n \geq 1$ and assume by induction hypothesis that $\mathcal{H}^n(\mathbf{REG}, \mathbf{REG}) \in \mathbf{HREG}$.

Let $L \in \mathcal{H}^{n+1}(\mathbf{REG}, \mathbf{REG})$. Then there is a H-system $H = (\Sigma_1, \Sigma_2, L_1, \varphi)$ with $L_1 \in \mathcal{H}^n(\mathbf{REG}, \mathbf{REG})$ and $\varphi(a) \in \mathbf{REG}$ 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 $a \in \Sigma_1$, resp.) such that $L_1 = L(r_1)$ ($\varphi(a) = L(r_a)$, resp.). Because $\Sigma_1 \cap \Sigma_2 = \emptyset$ it is easy to see that the H-expression

$$\left(\left(\ldots\left((r_1 \Uparrow_{a_1} r_{a_1}) \Uparrow_{a_2} r_{a_2}\right)\ldots\right) \Uparrow_{a_n} r_{a_n}\right)$$

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

Recently a particular extension of regular expressions and patterns so called pattern expressions were investigated by Campeanu and Yu [7]. For readability we slightly adapt their notation. Pattern expressions are based on regular patterns which are defined as follows:

Definition 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 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

$$L(p) = \left(\left(\ldots\left((L(p_0) \Uparrow_{x_1} L(p_1)) \Uparrow_{x_2} L(p_2)\right)\ldots\right) \Uparrow_{x_n} L(p_n)\right)$$

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

Remark 1. 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 $\mathcal{H}^*(\mathbf{REG}, \mathbf{REG})$ and *vice versa*.

Theorem 11. $\mathcal{H}^*(\mathbf{REG}, \mathbf{REG}) = \mathbf{PATEXP}$.

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

Let $L \in \mathcal{H}^{n+1}(\mathbf{REG}, \mathbf{REG})$. Then there is a H-system $H = (\Sigma_1, \Sigma_2, L_1, \varphi)$ with $L_1 \in \mathcal{H}^n(\mathbf{REG}, \mathbf{REG})$ and $\varphi(a) \in \mathbf{REG}$ 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

$$\begin{aligned} L &= \left(\left(\ldots\left(\left(L_1 \Uparrow_{a_1} \varphi(a_1)\right) \Uparrow_{a_2} \varphi(a_2)\right) \ldots\right) \Uparrow_{a_s} \varphi(a_s)\right) \\ &\quad \left(\left(\ldots\left(\left(L(p) \Uparrow_{a_1} L(q_{a_1})\right) \Uparrow_{a_2} L(q_{a_2})\right) \ldots\right) \Uparrow_{a_s} L(q_{a_s})\right) \\ &= L(p'). \end{aligned}$$

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

Next consider $\mathbf{PATEXP} \subseteq \mathcal{H}^*(\mathbf{REG}, \mathbf{REG})$. 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 $\mathcal{H}^*(\mathbf{REG}, \mathbf{REG})$.

Let $L \in \mathbf{PATEXP}$ 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 $L_1 \in \mathcal{H}^m(\mathbf{REG}, \mathbf{REG})$, for some m, and $\varphi(a) \in \mathbf{REG}$ for all $a \in \Sigma_1$, such that $L(p') = L(H)$. In order to get rid-off 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, $\mathbf{PATEXP} \subseteq \mathcal{H}^*(\mathbf{REG}, \mathbf{REG})$. □

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 12. *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 $r = s \Uparrow_a t$ ($r = s^{\Uparrow_a}$, 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 $r' = s' \Uparrow_a t'$ ($r' = s'^{\Uparrow_a}$, 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**, using expressions only.

Theorem 13. *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 $L = \{ m = m_1 \ldots m_n \mid m_1 \cdot \ldots \cdot m_n \in M \}$ 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 $r = s \Uparrow_a t$. 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

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

This completes our construction. ☐

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

Corollary 2. *The family* **HREG** *is not closed under inverse homomorphisms.*

Proof. Consider the H-expression $r = (A\#A) \Uparrow_A a^*$, 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 $\{ a^n b^n \mid n \geq 0 \}$, which does not belong to the family **HREG** by Theorem 8. Since H-expressions are closed under homomorphism and intersection with regular languages, our claim follows. □

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 [8] 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.

Theorem 14. *The fixed membership problem for H- and EH-expressions is* **NL**-*complete.*

Proof. The fixed membership problem for **EDT0L** systems is **NL**-complete [19]. Since, by Theorem 4 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 to (see, e.g., [15]) 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

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

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

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

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 15. *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 14, the containment in **PSpace** is inherited from the general membership problem for **EDT0L** systems [20].

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

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

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 16. *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 $\Uparrow$ by $\downarrow$ (and every $^{\Uparrow}$ by $^{\downarrow}$) and vice versa. Then $L(r) = \emptyset$ iff $L(r') = \emptyset$.* □

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

Theorem 17. *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 [14, 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 [18]. Due to Theorem 16, 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 16, 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 nonterminals $N = \{A_1, \ldots A_n\}$ 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

$$r_{i+1} = \Bigg(\bigg(\ldots\Big(\big(r_i \Uparrow_{A_1} s_{A_1}\big) \Uparrow_{A_2} s_{A_2}\Big)\ldots\bigg) \Uparrow_{A_n} s_{A_n}\Bigg),$$

and let $r_{n+1} = ((\ldots((r_i \Uparrow_{A_1} \emptyset) \Uparrow_{A_2} \emptyset)\ldots) \Uparrow_{A_n} \emptyset)$. 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. □

Theorem 18. *The equivalence problem for EH-expressions is undecidable.*

The proof can be given by reduction of Post's correspondence problem (see, e.g., [17]) which is rather standard and therefore omitted here. 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.

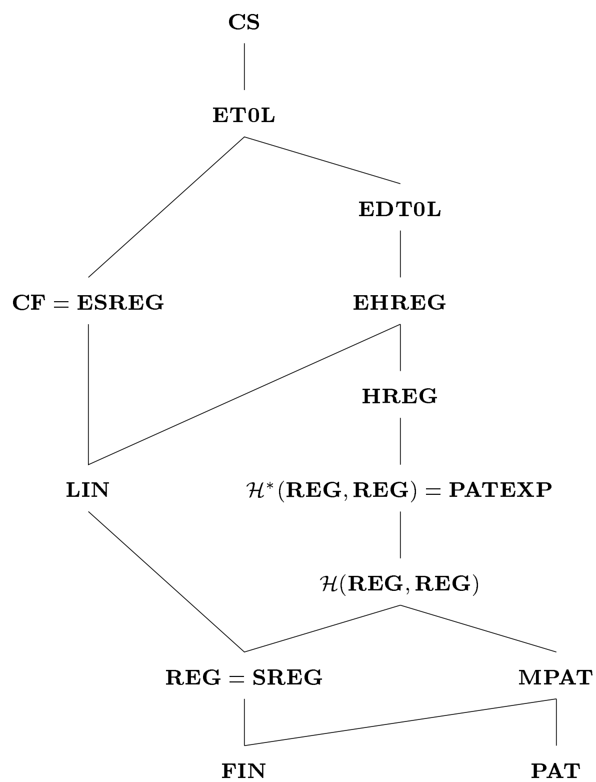

Fig. 1. The inclusion structure of the considered language families.

We hope that the investigation of homomorphic replacement, as one sort of pattern repeating operation, helps to 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 J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*. Elsevier Science Publishers B.V., 1990.
2. J. Albert and L. Wegner. Languages with homomorphic replacements. In *Proceedings of the 7th International Colloquium on Automata Languages and Programming*, number 85 in LNCS, pages 19–29. Springer, July 1980.
3. J. C. M. Baeten and W. P. Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1991.
4. J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*, volume 11 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1988.
5. J.-C. Birget and J. B. Stephen. Formal languages defined by uniform substitutions. *Theoretical Computer Science*, 132:243–258, 1994.
6. H. Bordihn, J. Dassow, and M. Holzer. H-Ausdücke—eine Erweiterung regulärer Ausdrücke. In F. Otto and G. Niemann, editors, *9. Theorietag der GI-Fachgruppe 0.1.5 "'Automaten und Formale Sprachen"'*, page 26. Fachbereich 17 Mathematik/Informatik, Universität Gesamthochschule Kassel, Heinrich-Plett Straße 40, D-34132 Kassel, Germany, September 1999.
7. C. Campeanu and S. Yu. Pattern expressions. Unpublished manuscript, 1999.
8. J. Dassow and K.-J. Lange. Complexity of automata with abstract storages. In L. Brim, J. Gruska, and J. Zlatuska, editors, *Proceedings of the 8th International Conference on Fundamentals of Computation Theory*, number 529 in LNCS, pages 200–209, Brno, Czech Republic, September 1991. Springer.
9. J. Dassow and Gh. Păun. *Regulated Rewriting in Formal Language Theory*, volume 18 of *EATCS Monographs in Theoretical Computer Science*. Springer, 1989.
10. 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*, number 71 in LNCS, pages 171–187. Springer, 1979.
11. D. Dougherty. *sed & awk*. O'Reilly & Associates, Inc., 1990.
12. J. Engelfriet and E. M. Schmidt. IO and OI. Part I and II. *Journal of Computer and System Sciences*, 15 and 16:328–353, 67–99, 1977.
13. D. Giammarresi and A. Restivo. Two-dimensional languages. In Rozenberg and Salomaa [26], pages 215–267.
14. J. Gruska. A characterization of context-free languages. *Journal of Computer and System Sciences*, 5:353–364, 1971.
15. J. Hartmanis, N. Immerman, and S. Mahaney. One-way log-tape reductions. In *Proceedings of the* 19*th Annual Symposium on Foundations of Computer Science*, pages 65–72, Ann Arbor, Michigan, October 1978. IEEE Society Press.
16. K. Hashiguchi and H. Yoo. Extended regular expressions of degree at most two. *Theoretical Computer Science*, 76:273–284, 1990.
17. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.
18. N. D. Jones and W. T. Laaser. Complete problems for deterministic polynomial time. *Theoretical Computer Science*, 3:105–117, 1977.
19. N. D. Jones and S. Skyum. Recognition of deterministic ET0L languages in logarithmic space. *Information and Computation*, 35:177–181, November 1977.
20. N. D. Jones and S. Skyum. Complexity of some problems concerning L systems. *Mathematical Systems Theory*, 13:29–43, 1979.
21. L Kari, A. Mateescu, Gh. Păun, and A. Salomaa. Multi-pattern languages. *Theoretical Computer Science*, 141:253–268, 1995.

22. S. C. Kleene. Representation of events in nerve nets and finite automata. In C. E. Shannon and J. McCarthy, editors, *Automata studies*, volume 34 of *Annals of mathematics studies*, pages 2–42. Princeton University Press, 1956.
23. A. Mateescu and A. Salomaa. Aspects of classical language theory. In Rozenberg and Salomaa [26], pages 175–251.
24. E. Ochmanski. Regular behaviour for concurrent processes. *Bulletin of the European Association for Theoretical Computer Science*, 27:56–67, 1985.
25. G. Rozenberg and A. Salomaa. *The Mathematical Theory of L Systems*, volume 90 of *Pure and Applied Mathematics*. Academic Press, 1980.
26. G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*, volume 1–3. Springer, 1997.
27. R. Siromoney and K. Krithivasan. Parallel context-free grammars. *Information and Control*, 24:155–162, 1974.
28. I. H. Sudborough. A note on tape-bounded complexity classes and linear context-free languages. *Journal of the ACM*, 22(4):499–500, October 1975.
29. M. K. Yntema. Cap expressions for context-free languages. *Information and Control*, 18(4):311–318, May 1971.

Appendix

In this appendix we give an operational semantics for regular expressions with *back referencing*. Back referencing is an assignment operator was introduced in [1] and allows to repeat certain patterns. Below we give a definition of regular expressions with back referencing which is suitable to model UNIX regular like expressions.

Definition 9 (BR-expressions). *Let Σ be an alphabet and Δ a set of variables with $\Sigma \cap \Delta = \emptyset$. The regular expressions with back referencing (BR-expression) over Σ and Δ and their set of assigned variables, called asg, are defined recursively as follows:*

1. *$\emptyset$ is a BR-expression and $asg(\emptyset) = \emptyset$.*
2. *λ is a BR-expression and $asg(\lambda) = \emptyset$.*
3. *For each $a \in \Sigma$, a is a BR-expression and $asg(a) = \emptyset$.*
4. *For each $v \in \Delta$, v is a BR-expression and $asg(v) = \emptyset$.*
5. *Let $v \in \Delta$ be a variable and r, s are BR-expressions. Then the following strings are BR-expressions:*
   (a) *$(r + s)$ is a BR-expression if $asg(r) \cap asg(s) = \emptyset$, and $asg(r + s) = asg(r) \cup asg(s)$.*
   (b) *(rs) is a BR-expression if $asg(r) \cap asg(s) = \emptyset$, and $asg(rs) = asg(r) \cup asg(s)$.*
   (c) *(r^*) is a BR-expression with $asg(r^*) = asg(r)$.*
   (d) *$(r\%v)$ is a BR-expressions if $v \notin asg(r)$, and $asg(r\%v) = asg(r) \cup \{v\}$.*
6. *Nothing else is a BR-expression.*

The set of languages described by BR-expressions is denoted by **BREG**.

As for regular expression, redundant parenthesis can be avoided using the same precedences and associatives as in regular expressions. The back referencing operator % is left-associative and has the highest precedence. Thus, if there is no danger of confusion, we omit out-most brackets. Let us give some examples.

Example 2. 1. $r = ((a + b)\%v)v$ is a BR-expression with $asg(r) = \{v\}$.
2. $s = (((a + b)^*\%v)v)^*v$ is a BR-expression with $asg(s) = \{v\}$.
3. $t = v((a + b)\%v)$ is a BR-expression with $asg(t) = \{v\}$.
4. $u = (v\%v)$ is a BR-expression with $asg(u) = \{v\}$.

The reader familiar with UNIX regular expression may have noticed, that our definition is more relaxed then in the UNIX case. For instance, we allow the usage of a variable before it is defined by an assignment as in cases 3 and 4 of our examples. As we will see later, these BR-expressions will denote the empty-set only. Nevertheless, BR-expressions can be seen as one possible model for UNIX regular expression.

It remains to define a semantic for BR-expressions. In contrast to ordinary regular expression, where usually the semantics is given by an inductive definition, here we run into problems because the variable v as a BR-expression will have no value, but the BR-expression $((a + b)^*\%v)v$ shall have one. To overcome this situation we give a operational semantics of BR-expressions based on action relations as used in process algebra (see, e.g., [3]). Before we need some notations:

Let $f : \Delta \to \Sigma^*$ be a partial function from the set of variables into the free monoid build by the input alphabet. Sometimes f is called the *memory* function on Σ and Δ. Let f be a memory function on Σ and Δ and r a BR-expressions over Σ and Δ, then we call the tuple $[f, r]$ and $[f, \surd]$ a *configuration*. The latter configuration is the so called *terminal* configuration. Now we are ready to define a reduction system based on a (generalized) action relation as follows:

Definition 10. *1. The* action relation *on configurations is defined as follows:*

(a) $[f, \lambda] \xrightarrow{\lambda} [f, \surd]$ *and* $[f, a] \xrightarrow{a} [f, \surd]$

(b) $[f, r^*] \xrightarrow{\lambda} [f, \surd]$ *and* $[f, r^*] \xrightarrow{\lambda} [f, rr^*]$.

2. The generalized action relation *is inductively defined:*

(a) $[f, r] \xrightarrow{a} [g, \surd]$ *implies* $[f, r] \overset{a}{\Rightarrow} [g, \surd]$ *and* $[f, r] \xrightarrow{a} [g, s]$ *implies* $[f, r] \overset{a}{\Rightarrow} [g, s]$.

(b) $[f, r] \overset{w}{\Rightarrow} [g, \surd]$ *implies* $[f, r + s] \overset{w}{\Rightarrow} [g, \surd]$ *and* $[f, s + r] \overset{w}{\Rightarrow} [g, \surd]$.

(c) $[f, r] \overset{w}{\Rightarrow} [g, \surd]$ *implies* $[f, rs] \overset{w}{\Rightarrow} [g, s]$.

(d) $[f, v] \overset{w}{\Rightarrow} [f, \surd]$ *if* $f(v)$ *is defined and evaluates to* w.

(e) $[f, r] \overset{w}{\Rightarrow} [g, \surd]$ *implies* $[f, r\%v] \overset{w}{\Rightarrow} [g', \surd]$*, where* g' *is identical to* g *except that* $g'(v)$ *evaluates to* w.

(f) $[f, r] \overset{w}{\Rightarrow} [g, s]$ *and* $[g, s] \overset{x}{\Rightarrow} [h, \surd]$ *implies* $[f, r] \overset{wx}{\Rightarrow} [h, \surd]$*. Moreover,* $[f, r] \overset{w}{\Rightarrow} [g, s]$ *and* $[g, s] \overset{x}{\Rightarrow} [h, t]$ *implies* $[f, r] \overset{wx}{\Rightarrow} [h, t]$.

Observe, that by definition the configuration $[f, \emptyset]$ does not derive any other configuration and $[f, \emptyset^*] \xrightarrow{\lambda} [f, \surd]$ regardless of the chosen memory function f. Moreover, note that a derivation that starts with $[f, \emptyset^*] \xrightarrow{\lambda} [f, \emptyset\emptyset^*]$ does not terminate at all, because the leading term $\emptyset$ can not be eliminated anymore.

Now we are ready to define the language associated to a particular BR-expression as follows.

Definition 11. *Let* $\perp$ *be the everywhere undefined memory function on alphabet* Σ *and variables* Δ*. Then the language defined by a BR-expression* r *over* Σ *and* Δ *is the language* $L(r) = \{\, w \in \Sigma^* \mid [\perp, r] \overset{w}{\Rightarrow} [g, \surd] \,\}$.

At this point we should give a larger example.

Example 3. Let $r = ((a+b)^*\%v)v$ be one of the BR-expression from our previous example. The below given derivations hold for an arbitrary memory function f.

1. $[f, a] \xrightarrow{a} [f, \surd]$ and $[f, b] \xrightarrow{a} [f, \surd]$.
2. $[f, a + b] \overset{a}{\Rightarrow} [f, \surd]$ and $[f, a + b] \overset{b}{\Rightarrow} [f, \surd]$.
3. In general we find $[f, (a+b)^*] \overset{w}{\Rightarrow} [f, \surd]$ if $w \in \{a, b\}^*$, because of the following three derivations:
   (a) $[f, (a + b)^*] \xrightarrow{\lambda} [f, \surd]$,

(b) $[f,(a+b)^*] \xrightarrow{\lambda} [f,(a+b)(a+b)^*] \overset{a}{\Rightarrow} [f,(a+b)^*]$, and

(c) $[f,(a+b)^*] \xrightarrow{\lambda} [f,(a+b)(a+b)^*] \overset{b}{\Rightarrow} [f,(a+b)^*]$.

4. $[f,(a+b)^*\%v] \overset{w}{\Rightarrow} [f',\surd]$ if $w \in \{a,b\}^*$ and f' is identical with f except that $f(v) = w$.
5. $[f,(a+b)^*\%v)v] \overset{w}{\Rightarrow} [f',v]$, where f' and w are defined as in (4). Moreover, $[f',v] \overset{w}{\Rightarrow} [f',\surd]$. Thus,

$$[f,((a+b)^*\%v)v] \overset{w}{\Rightarrow} [f',v] \overset{w}{\Rightarrow} [f',\surd],$$

and hence $[f,((a+b)^*\%v)v] \overset{ww}{\Rightarrow} [f',\surd]$. This completes our example.

Thus, we have shown that $L(r) = \{\, ww \mid w \in \{a,b\}^* \,\}$.

A closer look on the other examples shows that $s = (((a+b)^*\%v)v)^*v$ denotes the set $L(s) = \{\, w_1w_1w_2w_2 \ldots w_{n-1}w_{n-1}w_nw_nw_n \mid w_i \in \{a,b\}^* \text{ and } n \geq 1 \,\}$, while the remaining two BR-expressions $t = v((a+b)\%v)$ and $u = (v\%v)$ both denote the empty-set only, i.e., $L(t) = L(u) = \emptyset$. The latter is due to the fact that variable v is used before some value is assigned to it, and thus the generalized action relation terminates abnormally.

We have just seen that BR-expressions are quite powerful and can describe non-context-free languages. On the other hand, we conjecture that even the linear context-free language $\{\, a^nb^n \mid n \geq 0 \,\}$ is not a BR-expression language. Up to now we were not able to give a formal prove of this statement based on our formalism on the semantics of BR-expressions.