

Fixed Point Languages, Equality Languages, and Representation of Recursively Enumerable Languages

J. ENGELFRIET

Twente University of Technology, Enschede, The Netherlands

AND

G. ROZENBERG

University of Antwerp UIA, Wilrijk, Belgium

ABSTRACT. Fixed point languages and equality languages of homomorphisms and dgsm mappings are considered. Some basic properties of these classes of languages are proved, and it is shown how to use them to represent recursively enumerable sets. In particular, very simple languages are introduced which play the same role for the class of recursively enumerable languages that the Dyck languages play for the class of context-free languages. Finally, a new type of acceptor for defining equality languages is introduced.

KEY WORDS AND PHRASES: equality language, fixed point language, recursively enumerable language, deterministic sequential machine, Turing machine, Post correspondence problem, shuffle, AFL generator, representation of languages

CR CATEGORIES: 5.22, 5.23, 5.26, 5.27

1. Introduction

A considerable part of formal language theory deals with mappings on free monoids. Given two mappings α, β on the free monoid Σ^* generated by an alphabet Σ , the equality language of α and β denoted by $\text{Eq}(\alpha, \beta)$ consists of all words x from Σ^* such that $\alpha(x) = \beta(x)$. Thus equality languages of mappings measure the degree of their similarity (for example, if $\text{Eq}(\alpha, \beta) = \Sigma^*$, then α and β are identical; if $\text{Eq}(\alpha, \beta) = \emptyset$, then they are "totally different;" and if $\text{Eq}(\alpha, \beta) \neq \emptyset$, then they "have something in common"). If we consider homomorphisms of free monoids, then their equality sets represent sets of solutions of instances of the Post correspondence problem; in this sense equality sets of homomorphisms constitute a classical topic in formal language theory.

A revival of interest in these languages was stimulated recently by research concerning some basic decision problems in the theory of L systems (see, e.g., [8, 10]). It became apparent that equality languages of homomorphisms play a vital role in solutions of some of these decision problems. In fact, as opposed to the usual applications of the Post correspondence problem, one could prove (using the regularity of the equality languages involved) that these problems are decidable. Hence equality languages of mappings form not only a very natural subject to investigate from the mathematical point of view, but also a quite well-motivated topic within formal language theory.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

Authors' addresses: J. Engelfriet, Department of Applied Mathematics, Twente University of Technology, P.O. Box 217, 7500 AE Enschede, The Netherlands; G. Rozenberg, Department of Mathematics, University of Antwerp UIA, B-2610 Wilrijk, Belgium.

© 1980 ACM 0004-5411/80/0700-0499 \$00.75

Investigating the set of fixed points of a mapping (or a relation) is also very natural from the mathematical point of view. This set measures the degree of similarity of a given mapping with the identity mapping on the same domain. Also, because there exist rather simple relationships between fixed point languages and equality languages, various properties of the former are quite often closely connected to the latter, and vice versa. Furthermore, investigation of the fixed point languages of mappings and relations has a very special (biological) motivation in the theory of L systems [26]. In addition, it turns out that using fixed point languages allows one to characterize various traditional families of languages in the framework of L systems (see, e.g., [25]).

This paper is a sequel to [9, 12, 31] in that it concentrates on the systematic investigation of fixed point languages and equality languages for some very basic mappings encountered in formal language theory, namely, homomorphisms and mappings realized by deterministic generalized sequential machines (dgsms). This time, however, we focus on representations of recursively enumerable languages. Related, independent work appears in [6, 7].

Representing the class of recursively enumerable languages (RE) through operations on its subclasses is a traditional topic within formal language theory (see, e.g., [3, 14–16, 22–24]). Whenever one finds a representation of the class RE through one of its subclasses X , such a result sheds light on both RE and X . In this paper we discuss several representations of RE sets, in particular through fixed point languages of dgsms mappings, equality languages of homomorphisms, and fixed point languages of finite substitutions. We introduce a type of language (for each alphabet Σ one concrete language L_Σ , in the same way as for each Σ there is one Dyck language over Σ) of a very simple form, which plays an important role. These languages characterize the class of equality languages of homomorphisms (for Σ with two elements, L_Σ is the hardest language in the sense of [20] for this class). They also characterize the class of equality languages of dgsms mappings, and moreover they play for the class RE the same role as Dyck languages play for the class of context-free languages (e.g., we get an analog of the Chomsky–Schutzenberger theorem for the class RE). In particular, for Σ with two elements, L_Σ is a (very simple) full semi-AFL generator of RE. These results concerning L_Σ can also be explained in the framework of machine theory (in the sense of [15]). With this aim we introduce a machine type for which L_Σ equals the set of all instruction sequences which lead from initial to final storage. It turns out that these machines accept the class of recursively enumerable languages.

The emphasis of this paper is on showing how several (known or unknown) representation results can *easily* be derived using *simple constructions only*. At the same time we stress the role which the concept of equality (as formalized through equality languages and fixed point languages) plays in characterizations of RE. The paper is organized as follows. After providing preliminaries (concerning notation and terminology) in Section 2, in Section 3 we acquaint the reader with the topic of equality languages and fixed point languages of homomorphisms and dgsms mappings. We provide several examples of these and establish relationships between the corresponding classes, as well as position them within the classical Chomsky hierarchy. To this end we prove several structural theorems on equality languages. Section 3 may be skipped by the reader who is primarily interested in representation theorems.

In Section 4 we provide (effective) representations of RE through fixed point languages of dgsms mappings, equality languages of homomorphisms, fixed point languages of finite substitutions, and intersections of context-free languages.

Section 5 introduces complete twin shuffles—these are the aforementioned languages L_Σ . We use them to provide characterizations of each of the following classes: equality languages of homomorphisms, equality languages of dgsms mappings, and RE.

In Section 6 we consider several basic decision problems concerning fixed point languages of dgsms mappings. As an application of the results obtained we are able to locate “concrete” languages between the class of context-sensitive languages and some of its subclasses (e.g., IO macro languages and indexed languages).

In Section 7 we take a “machine point of view.” We define a new type of acceptor (called an equality machine) and show how various versions of it accept the following classes: equality languages of homomorphisms, equality languages of dgsm mappings, and RE. We also demonstrate how various notions and results considered in the paper can be understood within the framework of this machine model.

We assume the reader to be familiar with the basics of computability theory and formal language theory.

2. Preliminaries

Mostly we will use standard formal language-theoretic notation and terminology. However, the following perhaps deserve special mention.

(1) For a finite set Z , $\#Z$ denotes its cardinality. λ denotes the empty word. For a word x , $|x|$ denotes its length and x^R the mirror image of x . For a letter a , $\#_a x$ denotes the number of occurrences of a in x . A language K is called a star event if $K = K^*$, i.e., K is closed under concatenation and contains λ .

(2) Given alphabets Σ and Δ , $\text{HOM}(\Sigma, \Delta)$ denotes the set of all homomorphisms from Σ^* into Δ^* . The union of all $\text{HOM}(\Sigma, \Delta)$ is denoted by HOM . If α is a homomorphism that maps each letter into a letter, then we call it a coding; if it maps each letter into a letter or into the empty word, then we call it a weak coding; and if it maps each letter into itself or into the empty word, then we call it a weak identity. Given an alphabet V and a subset Σ of V we use $\text{Pres}_{V, \Sigma}$, or simply Pres_Σ if V is understood, to denote the weak identity preserving all letters of Σ and erasing letters from $V - \Sigma$.

(3) We will often identify a singleton set with its element; hence, for example, we write x^* rather than $\{x\}^*$. Also, as usual in formal language theory, we identify languages that differ at most by λ . Throughout the paper we will use the following “barring convention”: Given an alphabet Σ , we reserve $\bar{\Sigma}$ to denote $\{\bar{a} \mid a \in \Sigma\}$, $\bar{\Sigma} \cap \Sigma = \emptyset$. Then for every word x in Σ^* , \bar{x} denotes the word resulting from x by barring each occurrence of each letter in x .

(4) Let $A = (Q, \Sigma, \Delta, \delta, q_{\text{in}}, F)$ be a dgsm (deterministic generalized sequential machine with accepting states), so that δ maps $Q \times \Sigma$ to $Q \times \Sigma^*$. Then

- (i) δ_s and δ_o denote the state and the output component of δ , respectively;
- (ii) the translation of A is defined by $\text{Tr}(A) = \{(x, \delta_o(q_{\text{in}}, x)) \mid x \in \Sigma^* \text{ and } \delta_s(q_{\text{in}}, x) \in F\}$; and
- (iii) D GSM denotes the class of all dgsm mappings (all translations of dgsm's).

(5) We use FINSUB to denote the class of all finite substitutions. We use FIN, REG, CF, CS, and RE to denote the classes of finite, regular, context-free, context-sensitive, and recursively enumerable languages, respectively.

Now we define two notions that are basic for this paper.

(6) Let α be a (possibly partial) mapping, $\alpha: \Sigma^* \rightarrow \Delta^*$. A word x in Σ^* is called a *fixed point of α* if $\alpha(x) = x$. The *fixed point language of α* , denoted as $\text{Fp}(\alpha)$, is defined by $\text{Fp}(\alpha) = \{x \in \Sigma^* \mid \alpha(x) = x\}$. Analogously for a dgsm A , a word x is a *fixed point of A* if x is a fixed point of $\text{Tr}(A)$. The *fixed point language of A* , denoted as $\text{Fp}(A)$, is defined by $\text{Fp}(A) = \text{Fp}(\text{Tr}(A))$. For a relation $\alpha \subseteq \Sigma^* \times \Delta^*$ the *fixed point language* is defined by $\text{Fp}(\alpha) = \{x \in \Sigma^* \mid x \in \alpha(x)\}$.

For a class X of mappings or relations, $\text{FP}(X)$ denotes the family of all languages of the form $\text{Fp}(\alpha)$ for α in X .

(7) Let α, β be (possibly partial) mappings on Σ^* . The *equality language of α, β* , denoted as $\text{Eq}(\alpha, \beta)$, is defined by $\text{Eq}(\alpha, \beta) = \{x \in \Sigma^* \mid \alpha(x) = \beta(x)\}$. For relations α and β we define $\text{Eq}(\alpha, \beta) = \{x \in \Sigma^* \mid \alpha(x) \cap \beta(x) \neq \emptyset\}$. For dgsm's A and B , $\text{Eq}(A, B)$ denotes $\text{Eq}(\text{Tr}(A), \text{Tr}(B))$. For a class X of mappings or relations, $\text{EQ}(X)$ is the family of all languages of the form $\text{Eq}(\alpha, \beta)$ with $\alpha, \beta \in X$.

(8) The following basic property of concatenation of words will be used in the sequel.

LEMMA 1. *Let $x_1, x_2, y_1, y_2, u_1, u_2, v_1, v_2$ be arbitrary words. If $x_1y_1 = x_2y_2$, $u_1y_1 = u_2y_2$, and $x_1v_1 = x_2v_2$, then $u_1v_1 = u_2v_2$.*

PROOF. Without loss of generality we may assume that $|x_1| \leq |x_2|$. Since $x_1y_1 = x_2y_2$, x_1 is a prefix of x_2 , i.e., there exists a word w such that $x_2 = x_1w$.

Then $x_1y_1 = x_2y_2$ implies $x_1y_1 = x_1wy_2$ and so $y_1 = wy_2$. Now $u_1y_1 = u_2y_2$ implies $u_1wy_2 = u_2y_2$ and so $u_1w = u_2$; and $x_1v_1 = x_2v_2$ implies $x_1v_1 = x_1wv_2$ and so $v_1 = wv_2$. Hence $u_1v_1 = u_1wv_2 = u_2v_2$. \square

3. Fixed Point Languages and Equality Languages

In this section the fixed point languages and equality languages of homomorphisms and dgsm mappings are illustrated by examples and some of their basic properties are studied. This section may be skipped by the reader who is primarily interested in representation theorems.

We start by noticing the following basic and useful relationship between fixed point languages and equality languages of mappings (see also [7]). Since this result is obvious, it is given without proof.

LEMMA 2. *Let α, β be mappings or relations on Σ^* and let id_Σ be the identity mapping on Σ^* . Then $Eq(\alpha, \beta) = Fp(\beta^{-1}\alpha)$ and $Fp(\alpha) = Eq(\alpha, id_\Sigma)$. \square*

Next we give examples of equality languages of homomorphisms and dgsm mappings.

Example 1. Let α, β in $HOM(\{a, b, c\}, \{a, b, c, d\})$ be defined by $\alpha(a) = a$, $\alpha(b) = bc$, $\alpha(c) = bd$, $\beta(a) = ab$, $\beta(b) = cb$, and $\beta(c) = d$. Then clearly $Eq(\alpha, \beta)$ is the regular language $(ab^*c)^*$.

Example 2. Let α, β in $HOM(\{a, b\}, \{a\})$ be defined by $\alpha(a) = a$, $\alpha(b) = \lambda$, $\beta(a) = \lambda$, and $\beta(b) = a$. Then clearly $Eq(\alpha, \beta)$ is the context-free language $\{x \in \{a, b\}^* \mid \#_a x = \#_b x\}$.

Example 3. Let α, β in $HOM(\{a, b, c\}, \{a, b\})$ be defined by $\alpha(a) = a$, $\alpha(b) = b$, $\alpha(c) = \lambda$, $\beta(a) = \lambda$, $\beta(b) = a$, and $\beta(c) = b$. Then $Eq(\alpha, \beta) \cap a^*b^*c^* = \{a^n b^n c^n \mid n \geq 0\}$ and so $Eq(\alpha, \beta)$ is a non-context-free language.

Note that all languages in $EQ(HOM)$ are star events (in particular they contain the empty word). They have a more complicated structural property which is expressed in the following theorem.

THEOREM 1. *Let $K \in EQ(HOM)$ and let x, y, u, v be words over the alphabet of K . If $xy, uy, xv \in K$, then $uv \in K$.*

PROOF. Let $K = Eq(\alpha_1, \alpha_2)$. Define $x_i = \alpha_i(x)$ for $i = 1, 2$, and similarly for y, u , and v . Use Lemma 1. \square

The next result [12, 31] is an easy consequence of this theorem.

THEOREM 2. *Let $K \in EQ(HOM)$ over the alphabet Σ and let $x, y \in \Sigma^*$.*

- (i) *If $xy \in K$, then $\{w \in \Sigma^* \mid xwy \in K\}$ is a star event.*
- (ii) *If $x \in K$, then $\{v \in \Sigma^* \mid xv \in K\} = K$ and $\{w \in \Sigma^* \mid wx \in K\} = K$.*

PROOF

(i) To show that if $xw_1y, xw_2y \in K$, then $xw_1w_2y \in K$, let $u = xw_1$ and $v = w_2y$, and apply Theorem 1.

(ii) Taking $y = \lambda = u$ in Theorem 1 shows that if $x, xv \in K$, then $v \in K$. This and the fact that K is a star event proves the first equation; the proof of the second is symmetric. \square

Languages in EQ(DGSM) need not be star events. One can easily see that every regular language is in EQ(DGSM): Simply consider all dgsm's which perform only identity translations on words in their domain, and then consider equality languages of the form $\text{Eq}(\alpha, \alpha)$ where α is such a dgsm mapping. It should also be clear that EQ(DGSM) is closed under intersection with regular languages (modify both dgsm's in such a way that words which are not in the given regular language are rejected). Consequently, if we intersect the language from Example 2 with a^*b^* we get the result that the context-free language $\{a^n b^n \mid n \geq 0\}$ is in EQ(DGSM); note that this language is not a star event. Similarly, Example 3 implies that the context-sensitive language $\{a^n b^n c^n \mid n \geq 0\}$ is in EQ(DGSM). Our next example provides a context-free star event which is in EQ(DGSM) but not in EQ(HOM).

Example 4. Let A be a dgsm with domain $(a^+b^+)^*$ which translates each word $w = a^{m_1}b^{n_1}a^{m_2}b^{n_2} \dots a^{m_k}b^{n_k}$ into $ba^{m_1}ba^{m_2} \dots ba^{m_k}$. Let B be a dgsm with the same domain which translates w into $ba^{n_1}ba^{n_2} \dots ba^{n_k}$. A formal definition of A and B is left to the reader. Obviously $w \in \text{Eq}(A, B)$ if and only if $m_i = n_i$ for $1 \leq i \leq k$. Hence $\text{Eq}(A, B)$ is the context-free star event $K = \{a^n b^n \mid n \geq 1\}^*$. Note that $K \notin \text{EQ(HOM)}$ by Theorem 1: take, e.g., $x = a$, $y = b$, $u = aab$, and $v = abb$.

Languages in EQ(DGSM) have a structural property which generalizes the one of Theorem 1 concerning EQ(HOM). It says that if K is in EQ(DGSM) and words w_1y , w_2y , \dots , w_ny are in K (with n sufficiently large), then there exist w_i and w_j which play the role of x and u in Theorem 1. Formally this is stated in the next theorem.

THEOREM 3. *Let $K \in \text{EQ(DGSM)}$. There exists an integer N such that for all $n > N$ and for all words y, w_1, w_2, \dots, w_n , if $w_1y, w_2y, \dots, w_ny \in K$, then there exist i, j , $1 \leq i < j \leq n$, with the property that for all words $v, w_iv \in K$ if and only if $w_jv \in K$.*

PROOF. Let $K = \text{Eq}(A_1, A_2)$ with $A_k = (Q^k, \Sigma, \Delta, \delta^k, q_{\text{in}}^k, F^k)$ for $k = 1, 2$. Let $N = \#Q^1 \cdot \#Q^2$. Then there exist i and j such that A_k reaches the same state, say q_k , after reading w_i and w_j (for both $k = 1$ and $k = 2$); i.e., $\delta_s^1(q_{\text{in}}^1, w_i) = \delta_s^1(q_{\text{in}}^1, w_j) = q_1$ and $\delta_s^2(q_{\text{in}}^2, w_i) = \delta_s^2(q_{\text{in}}^2, w_j) = q_2$. Renaming w_i as x and w_j as u (or vice versa), the result now easily follows from an application of Lemma 1 to the words $x_k = \delta_o^k(q_{\text{in}}^k, x)$, $u_k = \delta_o^k(q_{\text{in}}^k, u)$, $y_k = \delta_o^k(q_k, y)$, and $v_k = \delta_o^k(q_k, v)$, for $k = 1, 2$. \square

As a consequence we obtain the following "pumping theorem" for EQ(DGSM).

THEOREM 4. *Let $K \in \text{EQ(DGSM)}$. There exists an integer N such that for all $n > N$ and for all words y, z_1, \dots, z_n , if $z_1y, z_1z_2y, z_1z_2z_3y, \dots, z_1z_2 \dots z_ny \in K$, then there exist i, j , $1 \leq i < j \leq n$, such that, for all $m \geq 0$, $z_1 \dots z_i(z_{i+1} \dots z_j)^m z_{j+1} \dots z_ny \in K$.*

PROOF. Let $w_k = z_1 \dots z_k$. By Theorem 3 there exist i, j such that $w_iv \in K$ if and only if $w_jv \in K$. Taking $v = z_{j+1} \dots z_ny$, $w_jv \in K$ implies $w_iv \in K$, i.e., $z_1 \dots z_i z_{j+1} \dots z_ny \in K$. This proves the result for $m = 0$. Now let $v = (z_{i+1} \dots z_j)^m z_{j+1} \dots z_ny$ and assume by induction that $w_iv \in K$. Then $w_jv \in K$, i.e., $w_i z_{i+1} \dots z_j v = z_1 \dots z_i(z_{i+1} \dots z_j)^{m+1} z_{j+1} \dots z_ny \in K$. \square

Example 5. Consider the language $K = \{\$a_1 \epsilon a_1 a_2 \epsilon a_1 a_2 a_3 \epsilon \dots \epsilon a_1 a_2 \dots a_n \$ \mid a_i \in \Sigma \text{ for } 1 \leq i \leq n\}$, where Σ is an alphabet with $\epsilon, \$ \notin \Sigma$. Application of Theorem 4 with $z_1 = \$a_1$, $z_i = \epsilon a_1 \dots a_i$ for $2 \leq i \leq n$ and $y = \$$ shows that $K \notin \text{EQ(DGSM)}$.

Now we turn to fixed point languages.

The fixed point languages of homomorphisms are rather simple objects, as is shown by the following (effective) result from [25].

THEOREM 5. *For every homomorphism α there exists a finite language K such that $\text{Fp}(\alpha) = K^*$.*

However, the fixed point languages of dgsm mappings form a more interesting class. Clearly this class contains all regular languages (consider all languages of the form $\text{Fp}(\alpha)$ where α is a dgsm which performs the identity translation on its domain). Furthermore, our next example provides a non-context-free language in $\text{FP}(\text{DGSM})$.

Example 6. Let Σ be an alphabet with $\epsilon, \$ \notin \Sigma$. Let A be a dgsm with domain $\Sigma^*(\epsilon\Sigma^*)^*\$$ which translates each word $w = \$w_1\epsilon w_2\epsilon w_3 \dots \epsilon w_n\$$ into $\$a_1w_2\epsilon a_2w_3 \dots \epsilon a_{n-1}w_n\epsilon a_n\$$, where a_i is the first letter of w_i . Clearly w is a fixed point of A if and only if $w_n = a_n$ and $w_i = a_iw_{i+1}$ for $1 \leq i \leq n-1$. Hence $\text{Fp}(A) = \{\$w_1\epsilon w_2\epsilon \dots \epsilon w_n\$ \mid \text{there exist } a_1, \dots, a_n \in \Sigma, n \geq 1, \text{ such that } w_i = a_i \dots a_n \text{ for } 1 \leq i \leq n\}$, which is not a context-free language.

The language K of Example 5 ($K \notin \text{EQ}(\text{DGSM})$) is the mirror image of the language $\text{Fp}(A)$ of Example 6. This shows that $\text{FP}(\text{DGSM})$ and $\text{EQ}(\text{DGSM})$ are not closed under mirror image, or, in other words, the class of fixed point languages of "reversed" dgsm's (which move from right to left on the input word; cf. [12]) is incomparable to $\text{FP}(\text{DGSM})$.

The following is another instructive example of a non-context-free language in $\text{FP}(\text{DGSM})$.

Example 7. Let A be a dgsm with domain $(a^*b^*)^*c$ which translates each word $a^{n_0}b^{n_1}a^{n_2}b^{n_3} \dots b^{n_k}c$ into $a^{2n_1}b^{2n_2}a^{2n_3} \dots a^{2n_k}bc$. It is easy to see that $\text{Fp}(A) = \{a^{2^k}b^{2^{k-1}}a^{2^{k-2}} \dots b^4a^2bc \mid k \geq 1 \text{ and } k \text{ is odd}\}$. Note that the mirror image of $\text{Fp}(A)$ is not in $\text{EQ}(\text{DGSM})$ by Theorem 4.

Unfortunately we have not been able to find a structural property specific for languages in $\text{FP}(\text{DGSM})$. The following combinatorial lemma allows us to provide a few examples of languages not in $\text{FP}(\text{DGSM})$.

LEMMA 3. *Let $K \subseteq \Sigma^*$ and let a, b be two different elements of Σ . If K contains an infinite subset of $\{a^n b^n \mid n \geq 1\}$ and $K \subseteq \{w \in \Sigma^* \mid \#_a w = \#_b w\}$, then $K \notin \text{FP}(\text{DGSM})$.*

PROOF. We will prove this result by contradiction.

Let us assume that K satisfies the assumptions of the lemma and that $A = (Q, \Sigma, \Delta, \delta, q_{\text{in}}, F)$ is a dgsm such that $K = \text{Fp}(A)$.

Let $n > \#Q$ be such that $a^n b^n \in K$. Let i, j be integers, $i < j$, such that A reads both the i th and the j th occurrence of a in the same state. Let x be the word produced by A on reading occurrences of a from the i th one to the $(j-1)$ st one. Clearly $x = a^r$ for some positive integer r (otherwise A would translate $a^m b^m$ into a word with prefix $a^n b$ for each $m > n$).

Note that when reading a^m for some $m \geq n$, A gets into a loop which translates each a^{j-i} into a^r . If $r > j - i$, then there exists an m_0 such that for every $m > m_0$, $a^m b^m$ is translated into a word with a prefix a^k for $k > m$, which contradicts the fact that K contains an infinite subset of $\{a^n b^n \mid n \geq 1\}$. On the other hand if $r < j - i$, then for m "much larger" than n , a^m is translated into a^k with $m - k$ large enough so that in reading b 's following a 's, A will get into a loop and will translate almost all b 's into a^* , leaving at most $\#Q - 1$ of them to be translated into b 's. This again implies that K contains only a finite subset of $\{a^n b^n \mid n \geq 1\}$, a contradiction.

Thus it must be that $r = j - i$. Then, however, we get the result that $a^{n+r} b^n \in \text{Fp}(A)$, which contradicts the fact that $\text{Fp}(A) \subseteq \{w \in \Sigma^* \mid \#_a w = \#_b w\}$. \square

As a direct corollary of the above lemma we get the result that, e.g., the language $\{x \in \{a, b\}^* \mid \#_a x = \#_b x\}$ from Example 2 is not in $\text{FP}(\text{DGSM})$ and that Dyck languages are not in $\text{FP}(\text{DGSM})$.

In the foregoing we have seen various examples of languages that are and are not in the classes $\text{EQ}(\text{HOM})$, $\text{FP}(\text{HOM})$, $\text{EQ}(\text{DGSM})$, and $\text{FP}(\text{DGSM})$. To broaden this picture, we now establish the interrelationships between these classes of languages and furthermore we locate them within the classical Chomsky hierarchy.

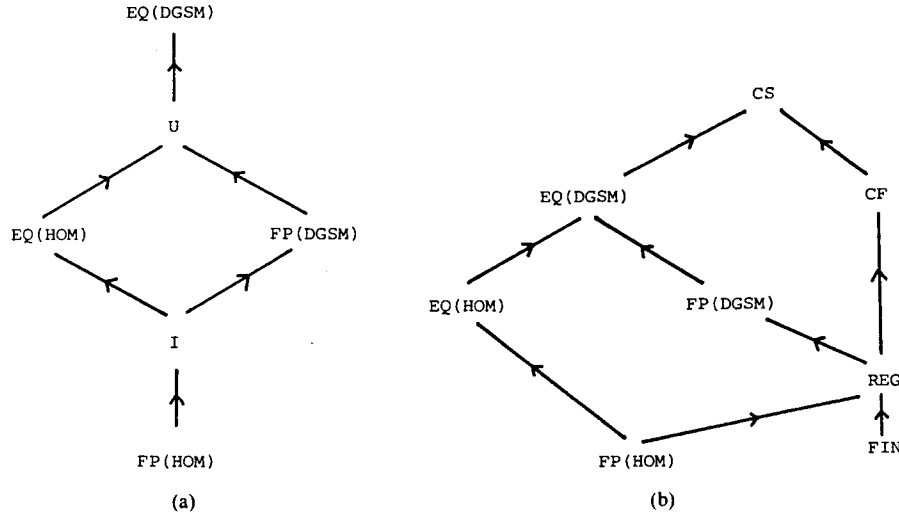


FIGURE 1

THEOREM 6. *The diagrams in Figure 1 hold, where U and I denote the union and intersection of $EQ(HOM)$ and $FP(DGSM)$, respectively.*

(In these diagrams a directed solid line leading from class X to class Y stands for the strict inclusion of X in Y . The absence of a line between classes X and Y stands for the fact that X and Y are incomparable but not disjoint classes.)

PROOF. The weak inclusions in Figure 1a and b follow directly from Lemma 2, Theorem 5, and the obvious weak inclusions $HOM \subseteq DGSM$, $REG \subseteq FP(DGSM)$, and $EQ(DGSM) \subseteq CS$.

The strict inclusions and incomparability results are proved as follows.

Figure 1a

(1) Example 2 provides a language in $EQ(HOM)$ which, by Lemma 3, is not in $FP(DGSM)$.

(2) $(ab \cup acb)^*$ is a (regular) language in $FP(DGSM)$ which, by Theorem 2(i), is not in $EQ(HOM)$.

(3) Example 1 provides a language in I which, by Theorem 5, is not in $FP(HOM)$.

(4) Example 4 provides a language in $EQ(DGSM) - EQ(HOM)$. By Lemma 3 the same language is not in $FP(DGSM)$.

Note that all languages used above to prove (1) through (4) are star events.

Figure 1b. Now it suffices to prove the following.

(5) Incomparability of $EQ(HOM)$ and $FP(HOM)$ with FIN follows from the fact that $a^* \in FP(HOM)$ and from Theorem 2(i), which implies that λ is the only finite language in $EQ(HOM)$.

(6) Example 3 provides a non-context-free language in $EQ(HOM)$, while Example 6 (as well as Example 7) provides a non-context-free language in $FP(DGSM)$.

(7) The context-free language $\{a^m b^n \mid 1 \leq m \leq n\}$ is not in $EQ(DGSM)$ by Theorem 4 (take $y = b^n$ and $z_i = a$, for $1 \leq i \leq n$). \square

To get a still clearer picture of the situation, we note that it was proved in [12] that there are no context-free languages in $FP(DGSM)$ which are not regular.

It is also easy to see (cf. [12]) that $EQ(DGSM) \subseteq DSPACE(\log n)$. In fact, every language $Eq(A_1, A_2) \in EQ(DGSM)$ can be simulated by a one-way deterministic two-head finite

state automaton which uses its i th head to simulate A_i , alternates the simulation of A_1 and A_2 in such a way that the difference between the output words of A_1 and A_2 stays bounded, and checks at each moment that one of the output words is a prefix of the other.

We end this section by a brief consideration of the case of the a-transducer (nondeterministic finite state transducer with λ -input and accepting states). Let A-TR denote the class of all a-transducer mappings.

THEOREM 7. $EQ(A-TR) = FP(A-TR)$, and this class is equal to the class of languages accepted by one-way nondeterministic two-head finite state automata.

PROOF. The first equality follows from Lemma 2 and the fact that A-TR is closed under composition and inverse (and contains all identities).

It is easy to see that the fixed point language of an a-transducer A can be recognized by a two-head automaton which simulates A with one head and keeps track of the output (on the input word, if possible) with the other head.

Now let M be a one-way nondeterministic two-head automaton. We define two a-transducers A_1 and A_2 as follows, using a technique from [21]: A_1 simulates the first head (and the finite control) of M by choosing and executing any instruction of M which is consistent with these, disregarding the behavior of the second head. The output of A_1 is the string of chosen instructions. A_2 is defined similarly, simulating the second head of M . Clearly a word w is accepted by M if and only if there is a sequence of instructions (of M) which is "executable" by both A_1 and A_2 , i.e., $A_1(w) \cap A_2(w) \neq \emptyset$, i.e., $w \in Eq(A_1, A_2)$. Hence $Eq(A_1, A_2)$ is the language accepted by M . \square

4. Representations of Recursively Enumerable Languages

Representing the class RE of recursively enumerable languages through "simple" operations on a "small" subclass of RE is a traditional topic within formal language theory. Such representation theorems can be used to show that certain problems are undecidable for languages in the subclass [23, 24], to show that the subclass is not closed under the operations [16], or to show the existence of simple AFL generators of RE [3, 15]. Since each RE language is the homomorphic image of the set of computations, suitably coded as strings, of a Turing machine (accepting or generating the language), these results are usually obtained by representing the class of "computation languages" of Turing machines.

Example 8. It is well known [30] and easy to see that each computation language of a Turing machine can be recognized by a (deterministic) one-way two-head finite state automaton. Hence, by Theorem 7, each RE language is the homomorphic image of the fixed point language of an a-transducer.

In this section we show the slightly stronger result that each RE language is the homomorphic image of the fixed point language of a dgsm mapping. We then demonstrate in this and the next section how (versions of) several other (known or unknown) representation theorems easily follow from this basic result together with some obvious properties of dgsm's.

Note that the idea behind the proof of the next result is a simple variation on the classical construction used to prove the undecidability of the Post correspondence problem.

THEOREM 8. For each recursively enumerable language K over an alphabet Σ there exists a dgsm mapping α such that $K = Pres_{\Sigma}(Fp(\alpha))$.

PROOF. We prove this theorem by demonstrating how the fixed point language of a dgsm simulates the set of computations of a deterministic Turing machine. For this purpose the formalism for Turing machines as in [5] will be especially useful. In particular, we assume that a Turing machine has a finite, but extendible, tape, as well as instructions $(q, a, q', \text{erase, move left})$ applicable only at the right end of its tape and instructions $(q, a, q', \text{erase, move right})$ applicable only at the left end of its tape; these instructions

cause the tape to contract (shrink). Moreover, it accepts by producing the empty tape (not to be confused with the blank tape!) in its (unique) final state.

Given a deterministic Turing machine A accepting the language K over Σ , we define a *computation string* of A to be a string of the form $\$w_1\epsilon\bar{w}_2\epsilon\bar{w}_3 \dots \epsilon\bar{w}_n\$$ such that

- (i) $n \geq 2$;
- (ii) ϵ and $\$$ are two new symbols not in the alphabet of A ;
- (iii) w_1, w_2, \dots, w_n are configurations of A , i.e., words uqv where u, v are words over the alphabet of A and q is a state of A (and as usual \bar{w}_i is the barred version of w_i);
- (iv) $w_1 = q_{in}w$ is an initial configuration of A , where q_{in} is the initial state of A and w is an input word over Σ ;
- (v) $w_n = q_f$ is the final configuration of A , where q_f is the final state of A .

We now define a *successful computation string* of A to be a computation string $\$w_1\epsilon\bar{w}_2\epsilon\bar{w}_3 \dots \epsilon\bar{w}_n\$$ such that w_{i+1} is the successor of w_i (due to the execution of one instruction of A) for $1 \leq i \leq n-1$. The *successful computation language* of A , denoted as $Scomp(A)$, is the set of all successful computation strings of A .

Clearly $K = Pres_{\Sigma}(Scomp(A))$. Hence to prove the theorem it suffices to show that there exists a dgsm B such that $Fp(B) = Scomp(A)$. To this aim we construct B as follows (note the resemblance to Example 6). Using its finite control, B rejects all words which are not computation strings of A (note that the set of computation strings is regular). The translation of a computation string $\$w_1\epsilon\bar{w}_2\epsilon \dots \epsilon\bar{w}_n\$$ is done by B as follows:

- (i) It erases $\$w_1$, paying attention to the "local situation" in w_1 , that is, the state of A and the symbol read by A , and storing in its finite control the instruction J_1 of A corresponding to this situation in A .
- (ii) On the basis of the information in (i) it rewrites $\epsilon\bar{w}_2$ into the word $\$w'_2$ where w'_2 is the (unique) predecessor of w_2 (if it exists) by instruction J_1 ; otherwise the string is rejected (note that w'_2 can be obtained from w_2 by a local change only; note also that w'_2 is unique because the change involves the unique state occurring in w_2). In performing this step B stores in its finite control the instruction J_2 corresponding to configuration w_2 in A .
- (iii) Then, for $3 \leq i \leq n$, it rewrites $\epsilon\bar{w}_i$ into $\epsilon\bar{w}'_i$, where w'_i is the predecessor of w_i (if it exists) by instruction J_{i-1} ; otherwise it rejects the string. In performing this step B stores the instruction J_i corresponding to configuration w_i in A .
- (iv) Finally, it rewrites $\$$ into $\epsilon\bar{w}_n$, where $w_n = q_f$.

As an example, if $J_{i-1} = (q, a, q', \text{print } b, \text{move right})$, then B rewrites $bq'd$ by qad (for $d \neq \epsilon$) and $bq'\epsilon$ by $q\epsilon$ (if a is the blank symbol) in $\epsilon w_i \epsilon$, where bars should of course be added. As another example, if $J_{i-1} = (q, a, q', \text{erase}, \text{move right})$, then B rewrites $\epsilon q'$ by $\epsilon q a$ in $\epsilon w_i \epsilon$.

Thus B translates a computation string $x = \$w_1\epsilon\bar{w}_2\epsilon\bar{w}_3 \dots \epsilon\bar{w}_n\$$ into $\$w'_2\epsilon\bar{w}'_3 \dots \epsilon\bar{w}'_n\epsilon\bar{w}_n\$$. Hence x is a fixed point of B if and only if $w_i = w'_{i+1}$ for $1 \leq i \leq n-1$ (i.e., w_i is the predecessor of w_{i+1} according to the instruction corresponding to w_i ; i.e., w_{i+1} is the successor of w_i). In other words, $x \in Fp(B)$ if and only if x is a successful computation string. So $Fp(B) = Scomp(A)$ and $K = Pres_{\Sigma}(Scomp(A))$. \square

We would like to make the following remarks concerning the above proof. When the dgsm B simulates the Turing machine A , it translates prefixes (of the input) of the form $\$w_1\epsilon\bar{w}_2 \dots \epsilon\bar{w}_{i+1}$ into prefixes (of the output) of the form $\$w_1\epsilon\bar{w}_2 \dots \epsilon\bar{w}_i$. Since, in general, the length of w_{i+1} can get arbitrarily long, we obtain in this way examples of dgsm's which when translating strings into themselves get into situations in which the length of the input already translated can become arbitrarily larger than the length of the output already obtained. This is a remarkable and very essential property of dgsm's. As a matter of fact, we show in [12] that if this property does not hold, the fixed point language of a dgsm is regular.

We will now investigate several representations of the fixed point languages of dgsm mappings. These results will then be used to provide various representations of the RE languages. One should notice that all representation results mentioned in this paper are effective.

Our first result in this line is based on the well-known representation of a-transducer mappings by a pair of homomorphisms and a regular language (see, e.g., [11, 29]).

LEMMA 4. *Let α be a dgsm mapping on Σ^* . There exist a coding β , a homomorphism γ , and a regular language M such that $Fp(\alpha) = Pres_{\Sigma}(Eq(\beta, \gamma) \cap M)$.*

PROOF. Let $A = (Q, \Sigma, \Delta, \delta, q_{in}, F)$ be a dgsm such that $Tr(A) = \alpha$. Let $\Theta = \Sigma \cup \{(q, a) \mid q \in Q \text{ and } a \in \Sigma\}$ and $\beta, \gamma \in HOM(\Theta, \Sigma)$ be defined by $\beta(q, a) = a = \beta(a)$, and $\gamma(a) = \lambda$, $\gamma(q, a) = d(\delta_o(q, a))$ where d is the homomorphism which doubles each letter b , i.e., $d(b) = bb$. Let M be the regular language over Θ consisting of all words of the form $(q_0, a_0)a_0(q_1, a_1)a_1 \cdots (q_i, a_i)a_i(q_{i+1}, a_{i+1})a_{i+1} \cdots (q_n, a_n)a_n$ such that $q_0 = q_{in}$, $\delta_s(q_n, a_n) \in F$ and $\delta_s(q_i, a_i) = q_{i+1}$ for $0 \leq i \leq n-1$.

Since obviously $\{(d(y), d(z)) \mid (y, z) \in \alpha\} = \{(\beta(x), \gamma(x)) \mid x \in M\}$, we get $Fp(\alpha) = Pres_{\Sigma}(Eq(\beta, \gamma) \cap M)$ and the result holds. \square

By a slight modification of the proof of the above lemma we get a representation of the fixed point languages of dgsm mappings by the fixed point languages of finite substitutions.

LEMMA 5. *Let α be a dgsm mapping on Σ^* . There exist a finite substitution β and a regular language M such that $Fp(\alpha) = Pres_{\Sigma}(Fp(\beta) \cap M)$.*

PROOF. Let $A = (Q, \Sigma, \Delta, \delta, q_{in}, F)$ be a dgsm such that $Tr(A) = \alpha$. Let $\Theta = \Sigma \cup \{(q, a) \mid q \in Q \text{ and } a \in \Sigma\}$ and let $\beta \in FINSUB$ be defined by $\beta(a) = \lambda$ and $\beta(q, a) = \{(q_0, a_0)a_0(q_1, a_1)a_1 \cdots (q_n, a_n)a_n \mid \delta_o(q, a) = a_0a_1 \cdots a_n\}$. Let M be the same regular language as in the proof of Lemma 4. Then obviously $Fp(\alpha) = Pres_{\Sigma}(Fp(\beta) \cap M)$. \square

Finally we will use linear (context-free) languages to represent fixed point languages of dgsm mappings. The proof is based on a well-known simulation of dgsm mappings by linear grammars [18]. For the notion of a simple deterministic (context-free) language we refer the reader to [27].

LEMMA 6. *Let α be a dgsm mapping on Σ^* . There exists a simple deterministic linear language $M \subseteq \Sigma^* \# \bar{\Sigma}^*$ such that $Fp(\alpha) = Pres_{\Sigma}(\{x \# \bar{x}^R \mid x \in \Sigma\} \cap M)$, where $\# \notin \Sigma$.*

PROOF. Let $A = (Q, \Sigma, \Delta, \delta, q_{in}, F)$ be a dgsm such that $Tr(A) = \alpha$, and let $G = (Q, \Sigma \cup \bar{\Delta} \cup \{\#, \bar{\#}\}, P, q_{in})$ be the context-free grammar defined by

- (i) for every q in F , $q \rightarrow \# \bar{\#}$ is in P , and
- (ii) for every q in Q and $a \in \Sigma$, if $\delta(q, a) = (q', w)$, then $q \rightarrow aq' \bar{w}^R$ is in P .

Note that G is linear and, moreover, because A is a deterministic gsm, G is simple deterministic. Since $L(G) = \{x \# \bar{y}^R \mid (x, y) \in Tr(A)\}$, if we set $M = L(G)$, then indeed $Fp(\alpha) = Pres_{\Sigma}(\{x \# \bar{x}^R \mid x \in \Sigma^*\} \cap M)$ and the lemma holds. \square

We note that $\bar{\#}$ is not needed but is added for symmetry.

Theorem 8 together with these representation results for fixed point languages of dgsm's (Lemmas 4–6) now yield various representations of RE languages (note that weak identities are closed under composition).

The first result concerns equality languages of homomorphisms, i.e., sets of solutions of instances of the Post correspondence problem; see also [6, 12, 31].

THEOREM 9. *For each recursively enumerable language K there exist a coding β , a homomorphism γ , a weak identity α , and a regular language M such that $K = \alpha(Eq(\beta, \gamma) \cap M)$.*

PROOF. Direct from Theorem 8 and Lemma 4. \square

It is easy to see that Lemma 4 also holds for a-transducers if one allows β to be an arbitrary homomorphism. Therefore the corresponding weaker version of Theorem 9 already follows easily from Theorem 7 and Example 8. Other versions of this result are shown in [6, 12, 31] and in the next section; see the discussion at the end of the next section for a comparison.

The next result concerns fixed point languages of finite substitutions.

THEOREM 10. *For each recursively enumerable language K there exist a finite substitution β , a weak identity α , and a regular language M such that $K = \alpha(Fp(\beta) \cap M)$.*

PROOF. Direct from Theorem 8 and Lemma 5. \square

It is instructive to compare the above result with Theorem 5; it certainly sheds some light on the question of how much stronger is the language generating power of finite substitutions than that of homomorphisms.

The last result concerns the well-known representation of RE languages by the intersection of two context-free languages, originally appearing in [16, 23] and the less well-known [22].

THEOREM 11. *For each recursively enumerable language K there exist an alphabet Δ , a simple deterministic linear language $M \subseteq \Delta^+ \bar{\Delta}^+$, and a weak identity α such that $K = \alpha(\{w\bar{w}^R \mid w \in \Delta^+\} \cap M)$.*

PROOF. Direct from Theorem 8 and Lemma 6 (let $\Delta = \Sigma \cup \{\#\}$). \square

Note that $\{w\bar{w}^R \mid w \in \Delta^+\}$ is also a simple deterministic linear language.

Let us compare Theorem 11 with several similar results in the literature (without claiming completeness!) (see also [32]). The general format of such results is that each RE language is of the form $\alpha(M_1 \cap M_2)$, where M_1 and M_2 are context-free languages and α is a weak identity. Two goals can be distinguished in the competition to obtain the simplest result of this kind: (1) M_1 and M_2 are taken from a small subclass of CF, and (2) one of M_1 and M_2 is of a "fixed form" (such as a Dyck language).

With respect to (1), Theorem 11 strengthens the results from [16, 23] and [3], where M_1 and M_2 are both deterministic and both linear, respectively. A result strengthening that of [16, 23] but not of [3] is the one of [24], also discussed in [15], where M_1 and M_2 are taken from the smallest family containing REG and $\{a^n b^n \mid n \geq 1\}$ and closed under inverse dgsms mappings and marked Kleene +.

With respect to (2), note that (in Theorem 11) since $M \subseteq \Delta^+ \bar{\Delta}^+$, the language $\{w\bar{w}^R \mid w \in \Delta^+\}$ may be replaced without trouble by the Dyck language over Δ . Thus Theorem 7 strengthens the result of [22], where M_1 is a Dyck language (see also [33] where a weaker version is stated), and similarly a result of [34], where M_1 is a "Dyck-like" language and M_2 is deterministic.

Strongly related representation results are those using pushdown transductions. In [17] it is shown that each RE language is of the form $\alpha(M)$, where M is a context-free language and α is a nondeterministic pushdown transduction; this result is clearly an easy consequence of that of [16, 23]. It is easy to see from Theorem 11 that each RE language is the image of $\{w\bar{w}^R \mid w \in \{0, 1\}^+\}$ under a deterministic one-turn pushdown transducer. Similarly one obtains from Theorem 11 the result that each RE language is of the form $\alpha(\beta^{-1}(\{w\bar{w}^R \mid w \in \{0, 1\}^+\}))$, where α is a weak identity and β a deterministic one-turn pushdown transduction (cf. [34]). Note finally that each RE language is accepted by a nondeterministic automaton with two one-turn pushdown tapes [3] and (by Theorem 11) is the range of a deterministic transducer with two one-turn pushdown tapes (see also Section 7).

We end this section with a few additional remarks on Theorems 8–11. First, we hope to have made it clear that these results are strongly interrelated. As another example, suppose one has a proof of a weaker version of Theorem 9 where β is only known to be a λ -free

homomorphism. From this one can then easily derive Theorems 9 and 10 as follows: (i) $\text{Eq}(\beta, \gamma) = \text{Fp}(\beta^{-1}\gamma) = \text{Fp}(\alpha)$ for some (nondeterministic) gsm mapping; (ii) Lemmas 4 and 5 also hold for gsm mappings, by a slight modification of the proofs; (iii) the set-theoretic equality $f(A) \cap B = f(A \cap f^{-1}(B))$ can be used to “get M inside.” In the same way one can prove Theorem 11 without the simple determinism (which seems to be a specific application of Theorem 8).

Second, Theorems 8–11 implicitly involve an alphabet V which is the domain of the weak identity $\alpha = \text{Pres}_{V,\Sigma}$, where Σ is the alphabet of K . In Theorem 8 the usual trick of coding all symbols in $V - \Sigma$ over $\{0, 1\}$ can be used to show that one can always take $V = \Sigma \cup \{0, 1\}$: the dgsm can easily read and produce coded symbols. This implies that $\Delta = \Sigma \cup \{0, 1, \#\}$ in Theorem 11 (with $V = \Delta \cup \bar{\Delta}$), i.e., V only depends on the alphabet of K . It is not clear whether a similar statement holds for Theorems 9 and 10.

Third, Theorems 8–10 imply that the class RE can be generated (in the sense of AFL theory) by each of the classes $\text{FP}(\text{DGSM})$, $\text{EQ}(\text{HOM})$, and $\text{FP}(\text{FINSUB})$. That means that $\text{RE} = \{\alpha(K) \mid \alpha \text{ is an a-transduction and } K \in \text{FP}(\text{DGSM})\}$ and similarly for $\text{EQ}(\text{HOM})$ and $\text{FP}(\text{FINSUB})$. Since RE is a full principal semi-AFL [12], there exists a language $K_0 \in \text{RE}$ which is a full generator of RE; i.e., $\text{RE} = \{\alpha(K_0) \mid \alpha \text{ is an a-transduction}\}$. Consequently there is also a full generator of RE in $\text{FP}(\text{DGSM})$; i.e., there exists $K_1 \in \text{FP}(\text{DGSM})$ such that $\text{RE} = \{\alpha(K_1) \mid \alpha \text{ is an a-transduction}\}$. Similarly, full generators of RE can be found in $\text{EQ}(\text{HOM})$ and $\text{FP}(\text{FINSUB})$. In the next section we show the existence in $\text{EQ}(\text{HOM})$ of a very simple generator of RE.

5. Complete Twin Shuffles

In this section we demonstrate the existence of a class of languages of a fixed form (the complete twin shuffles) which can be used to characterize each of the following classes of languages: $\text{EQ}(\text{HOM})$, $\text{EQ}(\text{DGSM})$, and RE. These languages play the same role for the class RE as the Dyck languages play for the class of context-free languages. By the usual “coding into two letters” argument this will imply the existence of a single language characterizing the classes $\text{EQ}(\text{HOM})$, $\text{EQ}(\text{DGSM})$, and RE: They are the smallest classes which contain this particular language and are closed under inverse homomorphisms, inverse dgsm mappings, and dgsm mappings, respectively. Thus this particular language (complete twin shuffle over $\{0, 1\}$) is the hardest language (in the sense of [20]) for $\text{EQ}(\text{HOM})$ and is a full AFL generator of RE.

It is well known from AFL theory [15] that simple generators of RE (or any other AFL) can be obtained from simple representation theorems of the kind of Theorem 11: Roughly speaking, it suffices to take a generator of the involved subclass of RE and shuffle it with itself (cf. [3, Cor. 4]). As usual, for strings x and y we denote by $\text{shuffle}(x, y)$ the finite language $\{x_1 y_1 x_2 y_2 \dots x_n y_n \mid x = x_1 x_2 \dots x_n \text{ and } y = y_1 y_2 \dots y_n\}$. For languages L_1 and L_2 , $\text{shuffle}(L_1, L_2) = \bigcup \{\text{shuffle}(x, y) \mid x \in L_1, y \in L_2\}$. To shuffle a language L with itself, one considers $\text{shuffle}(L, \bar{L})$. We will show that an alternative way of shuffling a language with itself gives rise to a very simple generator of RE (by an application of Theorem 8): Each word of the language is shuffled only with (a barred version of) itself.

Definition 1. The complete twin shuffle over an alphabet Σ is the language $L_\Sigma = \bigcup \{\text{shuffle}(w, \bar{w}) \mid w \in \Sigma^*\}$.

Note that $L_\Sigma = \{w \in (\Sigma \cup \bar{\Sigma})^* \mid \overline{\text{Pres}_\Sigma(w)} = \text{Pres}_{\bar{\Sigma}}(w)\}$. Note that the simplest L_Σ (with Σ a singleton) is isomorphic to the context-free equality language $\{x \in \{a, b\}^* \mid \#_a x = \#_b x\}$ (see Example 2). In fact, each L_Σ is a (very simple) equality language.

LEMMA 7. For every Σ , $L_\Sigma \in \text{EQ}(\text{HOM}) - \text{FP}(\text{DGSM})$.

PROOF. Let α, β in $\text{HOM}(\Sigma \cup \bar{\Sigma}, \bar{\Sigma})$ be defined as follows: For every a in $\Sigma \cup \bar{\Sigma}$,

$$\alpha(a) = \begin{cases} \bar{a} & \text{if } a \in \Sigma, \\ \lambda & \text{if } a \in \bar{\Sigma}, \end{cases} \quad \text{and} \quad \beta(a) = \begin{cases} \lambda & \text{if } a \in \Sigma, \\ a & \text{if } a \in \bar{\Sigma}. \end{cases}$$

Then obviously $\text{Eq}(\alpha, \beta) = L_\Sigma$. Hence $L_\Sigma \in \text{EQ}(\text{HOM})$. The fact that $L_\Sigma \notin \text{FP}(\text{DGSM})$ follows as a straightforward application of Lemma 3. \square

Next we show how to use $L_{(0,1)}$ to represent an arbitrary language in $\text{EQ}(\text{HOM})$ and an arbitrary language in $\text{EQ}(\text{DGSM})$; this will provide the basis for a characterization of $\text{EQ}(\text{HOM})$ and $\text{EQ}(\text{DGSM})$.

LEMMA 8

- (i) Let $K \in \text{EQ}(\text{HOM})$. There exists a homomorphism γ such that $K = \gamma^{-1}(L_{(0,1)})$.
- (ii) Let $K \in \text{EQ}(\text{DGSM})$. There exists a dgsm mapping γ such that $K = \gamma^{-1}(L_{(0,1)})$.

PROOF

(i) Let $K = \text{Eq}(\alpha, \beta)$ with α, β in $\text{HOM}(\Sigma, \Delta)$. Let $\Delta = \{a_1, \dots, a_n\}$. Let ϕ in $\text{HOM}(\Delta, \{0, 1\})$ be defined by $\phi(a_i) = 0^i 1$ for $1 \leq i \leq n$ and let γ in $\text{HOM}(\Sigma, \{0, 1\})$ be defined by $\gamma(b) = \phi(\alpha(b))\phi(\beta(b))$ for every b in Σ . Clearly, for every w in Σ , $\alpha(w) = \beta(w)$ if and only if $\gamma(w) \in L_{(0,1)}$. Consequently $K = \gamma^{-1}(L_{(0,1)})$, and the result holds.

(ii) Let $K = \text{Eq}(\alpha, \beta)$ with $\alpha = \text{Tr}(A)$ and $\beta = \text{Tr}(B)$ where $A = (Q, \Sigma, \Delta, \delta, q_{\text{in}}, F)$ and $B = (Q', \Sigma, \Delta, \delta', q'_{\text{in}}, F')$ are dgsm's. Let $D = (Q \times Q', \Sigma, \{0, 1, \bar{0}, \bar{1}\}, \rho, (q_{\text{in}}, q'_{\text{in}}), F \times F')$ be the dgsm with ρ defined as follows: $\rho((p, q), a) = ((p_1, q_1), \phi(x)\phi(y))$ where $\delta(p, a) = (p_1, x)$, $\delta'(q, a) = (q_1, y)$, and ϕ is defined as in (i) above. Then clearly $K = \gamma^{-1}(L_{(0,1)})$, where $\gamma = \text{Tr}(D)$. \square

The above result yields the following theorem characterizing $\text{EQ}(\text{HOM})$ and $\text{EQ}(\text{DGSM})$.

THEOREM 12

- (i) $\text{EQ}(\text{HOM}) = \{\gamma^{-1}(L_{(0,1)}) \mid \gamma \in \text{HOM}\}$, and it is the smallest class containing $L_{(0,1)}$ and closed under inverse homomorphisms.
- (ii) $\text{EQ}(\text{DGSM}) = \{\gamma^{-1}(L_{(0,1)}) \mid \gamma \in \text{DGSM}\}$, and it is the smallest class containing $L_{(0,1)}$ and closed under inverse dgsm mappings.

PROOF. We start by observing that if X is a class of mappings which is closed under composition, then $\text{EQ}(X)$ is closed under inverses of mappings from X . This follows because $\gamma^{-1}(\text{Eq}(\alpha, \beta)) = \text{Eq}(\alpha\gamma, \beta\gamma)$. In particular it means that $\text{EQ}(\text{HOM})$ is closed under inverse homomorphisms and $\text{EQ}(\text{DGSM})$ is closed under inverse dgsm mappings. Now (i) follows from Lemma 7 and Lemma 8(i), and (ii) follows from Lemma 7 and Lemma 8(ii). \square

Next we show how to represent fixed point languages of dgsm mappings by using languages L_Σ .

LEMMA 9. Let α be a dgsm mapping on Σ^* . There exists a regular language M such that $\text{Fp}(\alpha) = \text{Pres}_\Sigma(L_\Sigma \cap M)$.

PROOF. Let $A = (Q, \Sigma, \Delta, \delta, q_{\text{in}}, F)$ be a dgsm such that $\text{Tr}(A) = \alpha$. Let $B = (Q, \Sigma, \Sigma \cup \bar{\Delta}, \delta', q_{\text{in}}, F)$ be the dgsm where δ' is defined by $\delta'(q, a) = (p, w)$ if and only if $\delta(q, a) = (p, a\bar{w})$. Let $\text{Tr}(B) = \beta$. Since $M = \beta(\Sigma^*)$ is a regular language and obviously $\text{Fp}(\alpha) = \text{Pres}_\Sigma(L_\Sigma \cap M)$, the result holds. \square

Now we proceed to demonstrate how complete twin shuffles can be used to represent recursively enumerable languages.

THEOREM 13. Let K be a recursively enumerable language.

- (i) There exist an alphabet Σ , a weak identity α , and a regular language M such that $K = \alpha(L_\Sigma \cap M)$.
- (ii) There exist a weak identity α , a homomorphism β , and a regular language M such that $K = \alpha(\beta^{-1}(L_{(0,1)}) \cap M)$.
- (iii) There exists a dgsm mapping γ such that $K = \gamma(L_{(0,1)})$.

PROOF

(i) This follows directly from Theorem 8 and Lemma 9.

(ii) By (i), $K = \alpha(L_\Sigma \cap M)$. Let $\Sigma = \{a_1, \dots, a_n\}$ and let β in $\text{HOM}(\Sigma \cup \bar{\Sigma}, \{0, 1, \bar{0}, \bar{1}\})$ be defined by $\beta(a_i) = 0^i 1$, $\beta(\bar{a}_i) = \bar{0}^i \bar{1}$ for $1 \leq i \leq n$. Then clearly $L_\Sigma = \beta^{-1}(L_{(0,1)})$, and consequently $K = \alpha(\beta^{-1}(L_{(0,1)}) \cap M)$.

(iii) Let α and β be as above. Then we get $K = \alpha(\beta^{-1}(L_{(0,1)}) \cap M) = \alpha\beta^{-1}(L_{(0,1)} \cap \beta(M))$. Since $\beta(M)$ is regular and β^{-1} is obviously a dgsm mapping, there exists a dgsm mapping γ such that $K = \gamma(L_{(0,1)})$. \square

We end this section with several comments concerning Theorem 13.

It is easy to see that the alphabet Σ in Theorem 13(i) may be taken equal to $\Delta \cup \{0, 1\}$, where Δ is the alphabet of K (cf. the second remark at the end of Section 4).

Note that Lemma 9 also holds for a-transducers. Therefore Theorem 13(i) would also follow from Example 8.

It is interesting to note that Theorem 13(i) is an analog of the Chomsky-Schutzenberger theorem for context-free languages, which says that every context-free language K is of the form $K = \alpha(D_\Sigma \cap M)$, where Σ is an alphabet, D_Σ is the Dyck language over Σ , M is a regular language, and α is a weak identity. But then if we define β similarly to the way it was defined in the proof of Theorem 13(ii) so that it codes elements of Σ into bytes over $\{0, 1\}$, then indeed we get $K = \alpha(\beta^{-1}(D_{(0,1)}) \cap M)$ and so $K = \gamma(D_{(0,1)})$ for a dgsm γ . Hence L_Σ (or $L_{(0,1)}$) plays the same role for recursively enumerable languages as D_Σ (or $D_{(0,1)}$) for context-free languages.

Exactly the same theorem holds with L'_Σ instead of L_Σ , where L'_Σ is the shuffle of $P_\Sigma = \{w\bar{w}^R \mid w \in \Sigma^*\}$ with itself (where \bar{w} means w with a bar under each symbol), i.e., $L'_\Sigma = \text{shuffle}(P_\Sigma, \bar{P}_\Sigma)$. In fact it is easy to see that $L'_\Sigma = \text{Pres}_{\Sigma \cup \bar{\Sigma}}(L'_\Sigma \cap M)$, where M is the regular language $(\Sigma \cup \bar{\Sigma})^* \cdot \{\bar{a}\bar{a} \mid a \in \Sigma\}^*$; from this the result can be shown using Theorem 13. Note that L'_Σ is the intersection of two deterministic context-free languages (cf. Theorem 11); L'_Σ is not in $\text{EQ}(\text{HOM})$ by Theorem 2(i). Similar shuffle results have been shown in [3, 14, 15].

Actually, in Theorem 2 of [14] it is shown that representations of the form $K = \alpha(K_\Sigma \cap M)$, where K_Σ is a "fixed" language, are valid for each full principal AFL. Roughly, K_Σ from [14] is $\text{shuffle}(\Sigma^*, \text{shuffle}(\bar{\Sigma}^*, P))$ where P is a specific generator of the AFL C under consideration; P is obtained from the machine ("network") representation of C (cf. [15, 19] and Section 7).

It was shown in [31] that $\text{RE} = \{\alpha(K) \mid \alpha \in \text{DGSM and } K \in \text{EQ}(\text{HOM})\}$. Since, by Lemma 7, $L_{(0,1)} \in \text{EQ}(\text{HOM})$, Theorem 13(iii) is a stronger version of this result.

Another interesting consequence of Theorem 13(iii) is that if X is a class of languages properly included in RE and closed under dgsm mappings, then $L_{(0,1)} \notin X$ and even $L_\Sigma \notin X$ for every Σ with $\#\Sigma \geq 2$ (because $L_{(0,1)}$ is a homomorphic image of such an L_Σ). In this way we get, for example, the following application of Theorem 13(iii): If $\#\Sigma \geq 2$, then L_Σ is a (context-sensitive) language which is neither in the class of indexed languages (see [1]) nor in the class of tree-transformation languages.

Theorem 9, Theorem 13(i), and Theorem 6 from [12] provide three versions of the fact, obtained independently in [31], that for each RE language K there exist homomorphisms β and γ , a weak identity α , and a regular language M such that $K = \alpha(\text{Eq}(\beta, \gamma) \cap M)$. In Theorem 9 we have demonstrated that β can be taken as a coding, and in Theorem 13(i) we have shown that $\text{Eq}(\beta, \gamma)$ can be replaced by an equality language of fixed form (L_Σ for some Σ), whereas in [12, 31] it is shown that M can be taken in a fixed form ($\Sigma^* \Delta^*$ for some alphabets Σ and Δ). It has recently been shown in [6] that one can even omit M in case K is a star event; the main result in [6] is that each RE language K is of the form $\alpha(\text{Pref}(\text{Eq}(\beta, \gamma)))$, where $\text{Pref}(L) = \{x \in L \mid x \neq \lambda \text{ and no proper prefix } y \text{ of } x, \text{ except } \lambda, \text{ is in } L\}$.

6. Some Decision Problems

In this section we consider some basic decision problems relevant to the material that we have presented so far. We consider those results that are implied by representation theorems for recursively enumerable sets that were given before.

Our first result answers three of the most natural decision problems concerning fixed point languages of dgsm's.

THEOREM 14

- (i) *It is decidable whether or not $Fp(A) = \Sigma^*$ for an arbitrary dgsm A .*
- (ii) *It is undecidable whether or not $Fp(A) = \emptyset$ for an arbitrary dgsm mapping A .*
- (iii) *It is undecidable whether or not $Fp(A)$ is finite for an arbitrary dgsm mapping A .*

PROOF

(i) Let $A = (Q, \Sigma, \Delta, \delta, q_{in}, F)$ be a dgsm and let Q_r be the set of all states reachable from q_{in} . It is easy to see that $Fp(A) = \Sigma^*$ if and only if $Q_r \subseteq F$ and $\delta_o(q, a) = a$ for all $a \in \Sigma$ and all $q \in Q_r$.

(ii) This follows from Theorem 8: A language $\beta(Fp(\alpha))$, where β is a homomorphism, is empty if and only if $Fp(\alpha)$ is empty. Therefore the decidability of the emptiness problem for the fixed point languages of dgsm's would imply the decidability of the emptiness problem for recursively enumerable languages, but this problem is undecidable.

(iii) This follows similarly from (the proof of) Theorem 8. Notice that β , as constructed in that proof, is an injective mapping on $Fp(\alpha)$ (due to the determinism of the simulated Turing machines). \square

Theorem 14(i) can be strengthened quite considerably. It was shown in [9] that it is decidable whether or not $K \subseteq Eq(A, B)$ for an arbitrary context-free language K and arbitrary dgsm's A and B . It was recently shown in [7] that Theorem 14(i) is true for deterministic pushdown transducers and that it is even decidable for a nondeterministic pushdown transducer whether it realizes a subset of the identity.

A stronger version of Theorem 14(ii) was proved in [12]: The emptiness problem is even undecidable for the fixed point languages of D2L mappings.

Finally the reader should realize that the undecidability of the finiteness problem for $EQ(HOM)$ is known under the name of Post correspondence problem.

Next we turn to similar problems concerning complete twin shuffles. We show that it is undecidable for an arbitrary regular language M whether or not it contains a string from $L_{\{0,1\}}$.

THEOREM 15. *It is undecidable whether or not*

- (i) $M \cap L_{\{0,1\}} = \emptyset$,
- (ii) $M \cap L_{\{0,1\}}$ *is finite,*

where M is an arbitrary regular language.

PROOF. It follows from the proof of Theorem 13(i) (and so from the proofs of Theorem 8 and Lemma 9) that each recursively enumerable language K is (effectively) of the form $K = \alpha(L_\Sigma \cap M)$, where α is an injective mapping on $L_\Sigma \cap M$. Since both the emptiness and the finiteness problems are undecidable for recursively enumerable languages, this implies that both emptiness and finiteness are not decidable for languages of the form $L_\Sigma \cap M$, where M is a regular language. But then if we code elements of Σ into "bytes" from $\{0, 1\}$ (see our proof of Theorem 13(ii)), we also get that both emptiness and finiteness are undecidable for languages of the form $M \cap L_{\{0,1\}}$, where M is a regular language. \square

Obviously Theorem 15 is true for any other L_Σ providing that $\#\Sigma \geq 2$. This yields the following result.

THEOREM 16. *Let Σ be an alphabet such that $\#\Sigma \geq 2$. Let Γ be a family of grammars*

such that there exists an algorithm which, given an arbitrary finite automaton A and an arbitrary grammar G in Γ , decides whether or not $L(G) \cap L(A) = \emptyset$. Then $L_\Sigma \notin \{L(G) \mid G \in \Gamma\}$.

The usefulness of the above result stems from the fact that it allows one to locate "concrete" languages in between the class of context-sensitive languages and various subclasses of it. Thus, for example, we have the following result.

COROLLARY 1. *For every Σ with $\#\Sigma \geq 2$, L_Σ is a context-sensitive language which is not an IO macro language.*

Note that the class of IO macro languages (see [13]) is not closed under dgsm mappings, so that the above result cannot be obtained from Theorem 13(iii) (see the discussion following Theorem 13 at the end of the last section).

In the same way as Theorem 15 was proved on the basis of Theorem 13(i), one can, using Theorem 11, prove the following result.

THEOREM 17. *It is undecidable whether or not*

- (i) $M \cap \{w\bar{w}^R \mid w \in \{0, 1\}^*\} = \emptyset$,
- (ii) $M \cap \{w\bar{w}^R \mid w \in \{0, 1\}^*\}$ is finite,

where M is an arbitrary simple deterministic linear language.

7. Equality Machines

Guided by the results of the previous sections we present in this section an automaton (called the *equality machine*) which nondeterministically accepts all recursively enumerable languages and deterministically accepts all languages in the class EQ(DGSM).

The equality machine has the usual structure of a one-way acceptor (see Figure 2). It has a one-way input tape, a finite control, and a memory. The memory part consists of two write-only tapes (i.e., two tapes of the kind mostly used as output tapes); thus the machine has no tests on its memory. Initially the memory tapes are empty. The interesting feature of the equality machine, making it different from other machines, is that it "accepts by equality" (and final state): At the end of its computation the machine accepts the input only if the contents of the two memory tapes are the same.

Thus the equality machine may be regarded as a very special case of the (one-way nondeterministic) machine with two one-turn pushdown stores (equality may be checked by popping the symbols on each tape simultaneously); it was shown in [3] using their version of Theorem 11 that each RE language can be accepted by such a machine. Similarly, the equality machine is a special case of the recently introduced (one-way nondeterministic) machine with two "reset" tapes [4] (equality can be checked by resetting each head to the beginning of its tape and moving them simultaneously to the right); each RE language can be accepted by such a machine [4].

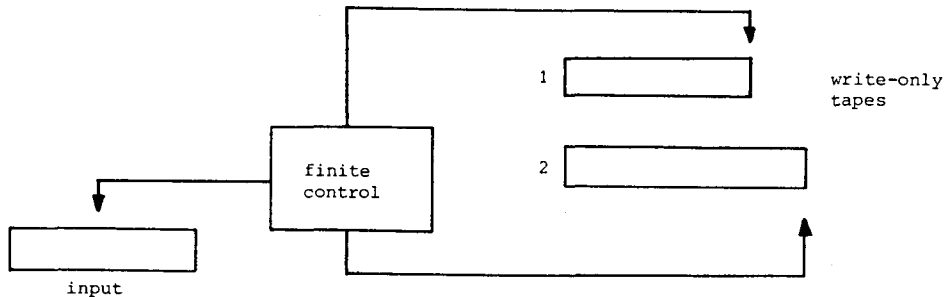


FIGURE 2

The equality machine can be formalized as follows.

Definition 2. A (one-way, nondeterministic) *equality machine* M is a structure $(Q, \Sigma, \Delta, \delta, q_{in}, F)$ where Q is the finite set of states, $q_{in} \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, Σ is the input alphabet, Δ is the memory alphabet, and δ is a mapping from $Q \times \Sigma^*$ into the finite subsets of $Q \times (\Delta \times \{1, 2\})^*$.

The interpretation of $(q, u) \in \delta(p, w)$ is that M , in state p , may move its input pointer over the string w , go into state q , and execute the sequence of instructions u on its memory. An instruction $\langle a, 1 \rangle$ is executed by writing the symbol a at the right end of its memory tape 1, and similarly for $\langle a, 2 \rangle$.

Definition 3. Let $M = (Q, \Sigma, \Delta, \delta, q_{in}, F)$ be an equality machine.

(1) For $u \in (\Delta \times \{1, 2\})^*$ we define $\bar{u}: \Delta^* \times \Delta^* \rightarrow \Delta^* \times \Delta^*$ as follows: For $v_1, v_2 \in \Delta^*$ and $a \in \Delta$, $\langle \bar{a}, 1 \rangle(v_1, v_2) = (v_1 a, v_2)$ and $\langle \bar{a}, 2 \rangle(v_1, v_2) = (v_1, v_2 a)$; for $u = u_1 u_2 \dots u_n$ ($u_i \in \Delta \times \{1, 2\}$, $n \geq 1$), $\bar{u}(v_1, v_2) = \bar{u}_n(\dots \bar{u}_2(\bar{u}_1(v_1, v_2)) \dots)$; for $u = \lambda$, $\bar{u}(v_1, v_2) = (v_1, v_2)$.

(2) A configuration of M is of the form $(q, w, (v_1, v_2))$ with $q \in Q$, $w \in \Sigma^*$, and $v_1, v_2 \in \Delta^*$. If $(q, u) \in \delta(p, w)$, then we write $(p, ww', (v_1, v_2)) \vdash (q, w', \bar{u}(v_1, v_2))$. The relation \vdash^* is defined as usual.

(3) The *language accepted* by M is $L(M) = \{w \in \Sigma^* \mid (q_{in}, w, (\lambda, \lambda)) \vdash^* (q, \lambda, (v, v)) \text{ for some } q \in F \text{ and } v \in \Delta^*\}$.

Note that in the definition of δ (in Definition 2) we could have taken $(\Delta \times \{1\})^*(\Delta \times \{2\})^*$ instead of $(\Delta \times \{1, 2\})^*$.

Definition 4. An equality machine $M = (Q, \Sigma, \Delta, \delta, q_{in}, F)$ is *deterministic* if δ is a mapping from $Q \times \Sigma$ into $Q \times (\Delta \times \{1, 2\})^*$. M is *one-state* if $Q = \{q_{in}\} = F$.

It is usual to allow a deterministic machine also to read λ -input; however, since there are no tests on the memory, this makes no difference for the equality machine. In a one-state deterministic machine the input can be viewed as a “code” of a sequence of instructions (δ is a mapping from Σ into $(\Delta \times \{1, 2\})^*$); this code is interpreted directly by the machine by executing the corresponding instructions. Thus one-state deterministic machines are a natural subclass of the class of deterministic machines.

The classes of languages accepted by nondeterministic, deterministic, and one-state deterministic equality machines will be denoted by $\mathcal{L}_N(\text{EM})$, $\mathcal{L}_D(\text{EM})$, and $\mathcal{L}_{OD}(\text{EM})$, respectively.

We now show the power of the equality machine and its restrictions.

THEOREM 18

- (i) $\mathcal{L}_N(\text{EM}) = RE$,
- (ii) $\mathcal{L}_D(\text{EM}) = EQ(\text{DGSM})$;
- (iii) $\mathcal{L}_{OD}(\text{EM}) = EQ(\text{HOM})$.

PROOF

(i) This follows from Theorem 8. Since $\mathcal{L}_N(\text{EM})$ is obviously closed under homomorphism, it remains to show that the equality machine can accept $\text{Fp}(\alpha)$ for all dgsm mappings α . But clearly, for every input string w , the equality machine can copy w into the first memory tape and simultaneously, simulating the dgsm A with $\text{Tr}(A) = \alpha$, write $\alpha(w)$ on the second memory tape. Thus w is accepted by the equality machine if and only if $\alpha(w) = w$.

(ii) To see that $\text{EQ}(\text{DGSM}) \subseteq \mathcal{L}_D(\text{EM})$ it suffices to notice that a deterministic equality machine can simulate two dgsm's simultaneously and write their respective output strings on the memory tapes. For the reverse inclusion let M be a deterministic equality machine. A dgsm A_1 can be obtained from M by viewing the first memory tape of M as the output tape of A_1 , and disregarding the second memory tape of M . Similarly a dgsm A_2 can be

constructed which simulates M with respect to its second memory tape. It should be clear that the equality language of A_1 and A_2 is precisely the language accepted by M . Note that this would not work in the nondeterministic case.

(iii) The constructions used in the proof of (ii) preserve the one-state property (of both the equality machine and the dgsm's involved). Since a one-state dgsm is a homomorphism, the result follows. \square

Note that Theorem 18(i) can also be proved easily by the fact that the equality machine is very suitable to simulate tag-systems [28] (see also [12]).

We also note that regarding the equality machine as a machine with two one-turn pushdown stores shows that Theorem 18(i) is just a stronger version of Theorem 2 of [3]: Every RE language can be accepted by such a machine. A similar remark holds for the "reset" machine of [4].

We now discuss how the results of Section 4 (Theorems 12 and 13) can be understood from Theorem 18 and some elementary facts from machine theory (AFA theory [15]; see also [19]).

Let $M = (Q, \Sigma, \Delta, \delta, q_{in}, F)$ be an equality machine. Let I_Δ denote $\Delta \times \{1, 2\}$, i.e., the set of instructions of M . Now δ is a mapping from $Q \times \Sigma^*$ into the finite subsets of $Q \times I_\Delta^*$. By viewing I_Δ as an alphabet, M may be viewed as an a-transducer A_M from Σ^* into I_Δ^* . Let $\alpha_M = Tr(A_M)$. The relationship between $L(M)$ and α_M is established as follows. Let L^Δ denote the language over I_Δ of all sequences of instructions leading from the initial memory state to a final memory state, i.e., $L^\Delta = \{u \in I_\Delta^* \mid \bar{u}(\lambda, \lambda) = (v, v) \text{ for some } v \in \Delta^*\}$. Then $L(M) = \alpha_M^{-1}(L^\Delta)$; see [15] for a formal proof, or better [19], because in [15] acceptance by equality has to be simulated by an additional instruction.

But it is easy to see (after renaming $\langle c, 1 \rangle$ as c and $\langle c, 2 \rangle$ as \bar{c} for every $c \in \Delta$) that $L^\Delta = L_\Delta$ as defined in Section 5 (Definition 1). In words, the complete twin shuffle L_Δ is precisely the language of all instruction sequences which lead from the initial memory state to some final memory state of the equality machine with memory alphabet Δ .

Since one may assume that each equality machine has memory alphabet $\Delta = \{0, 1\}$, it follows from the above considerations that $\mathcal{L}_N(EM) = \{\alpha^{-1}(L_{(0,1)}) \mid \alpha \text{ is an a-transducer}\}$. Moreover, α^{-1} may be replaced by α , because a-transducers are closed under inverse. By Theorem 18(i) and well-known decomposition properties of a-transducers, this implies the result of Theorem 13(ii).

In the deterministic case δ is a mapping from $Q \times \Sigma$ into $Q \times I_\Delta^*$, and consequently α_M is a dgsm mapping. Hence $\mathcal{L}_D(EM) = \{\alpha^{-1}(L_{(0,1)}) \mid \alpha \text{ is a dgsm mapping}\}$. By Theorem 18(ii) this corresponds to Theorem 12(ii). Finally, in the one-state deterministic case δ is a mapping from Σ into I_Δ^* , and so α_M is a homomorphism. Thus $\mathcal{L}_D(EM) = \{\alpha^{-1}(L_{(0,1)}) \mid \alpha \text{ is a homomorphism}\}$, which is Theorem 12(i) (by Theorem 18(iii)).

It is also possible to explain Theorem 13(iii) by the same methods, as follows: Consider a (nondeterministic) equality machine working backward in time, i.e., it starts with its memory pointers at the end of equal memory tapes, goes through its computation steps in the reverse order, moving its memory pointers to the left, and halts with empty memory tapes. It should be clear (by identifying the equal tapes) that a backward equality machine is the same as a one-way two-head finite state transducer (and vice versa): The single memory tape is the input tape of the transducer, whereas the original input string is produced as output; the two memory pointers both move on the same tape. Since it is well known [30] that each recursively enumerable language can be obtained as the range of a deterministic one-way two-head transducer (it checks the successful computation strings of a Turing machine by comparing two consecutive configurations with its two heads), it easily follows that the mapping δ of the corresponding equality machine is from $Q \times \Sigma^*$ to the finite subsets of $Q \times I_\Delta$ and such that δ^{-1} is a mapping from $Q \times I_\Delta$ into $Q \times \Sigma^*$ (δ^{-1} is the transition function of the one-way two-head transducer). Consequently, for each equality machine M we may assume that α_M is an inverse dgsm mapping. Hence, as before, $\mathcal{L}_N(EM) = \{\alpha^{-1}(L_{(0,1)}) \mid \alpha \text{ is an inverse dgsm mapping}\} = \{\alpha(L_{(0,1)}) \mid \alpha \in \text{DGSM}\}$, and we have obtained Theorem 13(iii). It should now be clear that an easy proof of Theorem

13(iii) can also be obtained directly from the simulation of Turing machines by one-way two-head transducers.

Note that the above remarks together with Theorem 7 also imply that if the memory tapes of the equality machine are viewed as output tapes (and the output is the one word on both tapes), then the class of ranges of such an "equality transducer" is EQ(A-TR).

We conclude this section by considering the two-way case. Similarly to the one-way case, a two-way deterministic equality machine is given by a two-way dgsm mapping from Σ^* into I_1^* (for the notion of two-way dgsm see [2]; the input is surrounded by endmarkers). Hence, denoting $\mathcal{L}_{2D}(\text{EM})$ the corresponding class of languages, we obtain as before that $\mathcal{L}_{2D}(\text{EM}) = \{\alpha^{-1}(L_{(0,1)}) \mid \alpha \text{ is a two-way dgsm mapping}\}$ which is the smallest class containing $L_{(0,1)}$ and closed under two-way dgsm mappings (these mappings are closed under composition). It is also straightforward to show that $\mathcal{L}_{2D}(\text{EM}) = \text{EQ}(2\text{DGSM})$, where 2DGSM denotes the class of two-way dgsm mappings. These results generalize Theorem 12 and Theorem 18.

We note that $\text{EQ}(\text{DGSM}) \subsetneq \text{EQ}(2\text{DGSM})$ (cf. Examples 5 and 6) and that $\text{EQ}(2\text{DGSM}) \subseteq \text{DSPACE}(\log n)$ (cf. [12] and the comment following Theorem 6). From the above characterization of $\text{EQ}(2\text{DGSM})$ it can be shown that $\text{EQ}(2\text{DGSM})$ is included in (one-turn)-2DPDA, the class of languages accepted by two-way deterministic pushdown automata for which the pushdown store turns once only. In fact, it is obvious that $L_{(0,1)}$ is in this class; moreover, every class of two-way deterministic automata of a given storage type is closed under inverse 2dgsm mappings [2].

8. Discussion

In this paper we have considered the fixed point languages and the equality languages of homomorphisms and dgsm mappings. We have investigated some of their basic properties, like, e.g., the relationships between these classes of languages and their position in the Chomsky hierarchy. However, we have focused our attention on the problem of representing recursively enumerable languages by languages from the above mentioned classes. In particular, we have found equality languages of a special, very simple form that play for the class RE the same role as Dyck languages play for the class of context-free languages. These special languages (complete twin shuffles) were shown also to have a very clear interpretation in the framework of equality machines.

The results of this paper add to the research of [6, 7, 9, 12, 31]. However, it is rather clear that all these papers together form only a beginning of research in this (rather promising) direction. For example, there is no reason to restrict attention to homomorphisms and dgsm mappings only. Formal language theory is full of various kinds of mappings on free monoids. The thorough investigation of these mappings is very essential for understanding various aspects of formal language theory. But clearly considering the nature of similarity of mappings and of their fixed point languages constitutes perhaps the most basic step (from the mathematical point of view) in their systematic investigation. As we have also seen, research in this direction is quite instructive from the point of view of various decision problems.

For example, one could start by considering several two-way and nondeterministic variants of dgsm mappings. Comparing these new classes of (equality and fixed point) languages with the corresponding ones for dgsm mappings could shed some light on the nature of deterministic and one-way restrictions in machines.

ACKNOWLEDGMENTS. The authors are indebted to A. Ehrenfeucht for useful discussions on the topic of this paper. They also thank P. R. J. Asveld and in particular one of the referees for many useful remarks on a previous version of this paper.

REFERENCES

1. AHO, A.V. Indexed grammars—an extension of context-free grammars. *J. ACM* 15, 4 (Oct. 1968), 647–671.

2. AHO, A.V., AND ULLMAN, J.D. A characterization of two-way deterministic classes of languages. *J. Comput. Syst. Sci.* 4 (1970), 523-538.
3. BAKER, B.S., AND BOOK, R.V. Reversal-bounded multipushdown machines. *J. Comput. Syst. Sci.* 8 (1974), 315-332.
4. BOOK, R.V., GREIBACH, S.A., AND WRATHALL, C. Comparisons and reset machines. Proc 5th Int. Colloq. on Automata, Languages, and Programming, *Lecture Notes in Computer Science* 62, Springer-Verlag, Berlin, 1978, pp. 113-124.
5. CLARK, K.L., AND COWELL, D.F. *Programs, Machines and Computation*. McGraw-Hill, London, 1976.
6. CULIK, K. II. A purely homomorphic characterization of recursively enumerable sets. *J. ACM* 26, 2 (April 1979), 345-350.
7. CULIK, K. II. Some decidability results about regular and pushdown translations. *Inform. Proc. Letters* 8 (1979), 5-8.
8. CULIK, K. II, AND FRIS, I. The decidability of the equivalence problem for DOL systems. *Inform. and Control* 35 (1977), 20-39.
9. CULIK, K. II, AND SALOMAA, A. On the decidability of homomorphism equivalence for languages. *J. Comput. Syst. Sci.* 17 (1978), 163-175.
10. EHRENFEUCHT, A., AND ROZENBERG, G. Elementary homomorphisms and a solution of the DOL sequence equivalence problem. *Theoret. Comput. Sci.* 7 (1978), 169-184.
11. EILENBERG, S. *Automata, Languages, and Machines, Vol. A*. Academic Press, New York, 1974.
12. ENGELFRIET, J., AND ROZENBERG, G. Equality languages and fixed point languages. *Inform. and Control* 43 (1979), 20-49.
13. FISCHER, M.J. Grammars with macro-like productions. Ph.D. Th., Harvard U., Boston, Mass., 1968.
14. FISHER, G.A., AND RANEY, G.N. On the representation of formal languages using automata on networks. IEEE Conf. Rec. 10th Ann. Symp. Switching and Automata Theory, Waterloo, Ontario, Canada, 1969, 157-165.
15. GINSBURG, S. *Algebraic and Automata-Theoretic Properties of Formal Languages*. North-Holland/American Elsevier, Amsterdam/New York, 1975.
16. GINSBURG, S., GREIBACH, S.A., AND HARRISON, M.A. Stack automata and compiling. *J. ACM* 14, 1 (Jan. 1967), 172-201.
17. GINSBURG, S., AND ROSE, G.F. Preservation of languages by transducers. *Inform. and Control* 9 (1966), 153-176. See also: GINSBURG, S., AND ROSE, G.F. A note on preservation of languages by transducers. *Inform. and Control* 12 (1968), 549-552.
18. GINSBURG, S., AND SPANIER, E.H. Finite-turn pushdown automata. *SIAM J. Control* 4 (1966), 429-453.
19. GOLDSTINE, J. Automata with data storage. Proc. Conf. Theoret. Comput. Sci., University of Waterloo, Waterloo, Ontario, 1977, pp. 239-246.
20. GREIBACH, S.A. The hardest context-free language. *SIAM J. Comput.* 2 (1973), 304-310.
21. GREIBACH, S.A. Visits, crosses, and reversals for nondeterministic off-line machines. *Inform. and Control* 36 (1978), 174-216.
22. GRIFFITHS, T.V. Some remarks on derivations in general rewriting systems. *Inform. and Control* 12 (1968), 27-54.
23. HARTMANIS, J. Context-free languages and Turing machine computations. In *Mathematical Aspects of Computer Science, Vol. 19: Proc. Symp. Applied Mathematics*, J.T. Schwartz, Ed., American Mathematical Society, Providence, R.I., 1967, pp. 42-51.
24. HARTMANIS, J., AND HOPCROFT, J.E. What makes some language theory problems undecidable? *J. Comput. Syst. Sci.* 4 (1970), 368-376.
25. HERMAN, G.T., AND WALKER, A. Context-free languages in biological systems. *Int. J. Comput. Math.* 4 (1975), 369-391.
26. HERMAN, G.T., AND WALKER, A. On the stability of some biological schemes with cellular interactions. *Theoret. Comput. Sci.* 2 (1976), 115-130.
27. KORENJAK, A.J., AND HOPCROFT, J.E. Simple deterministic languages. IEEE 7th Ann. Symp. Switching and Automata Theory, Berkeley, Calif., 1966, pp. 36-46.
28. MINSKY, M. *Computation: Finite and Infinite Machines*. Prentice-Hall, Englewood Cliffs, New Jersey, 1967.
29. NIVAT, M. Transductions des langages de Chomsky. *Ann. Inst. Fourier* 18, 1 (1968), 339-456, Grenoble.
30. ROZENBERG, A.L. On multi-head finite automata. *IBM J. Res. Devel.* 10 (1966), 388-394.
31. SALOMAA, A. Equality sets for homomorphisms of free monoids. To appear in *Acta Cybernetica*.
32. SALOMAA, A. *Formal Languages*. Academic Press, New York, 1973.
33. SAVITCH, W.J. How to make arbitrary grammars look like context-free grammars. *SIAM J. Comput.* 2 (1973), 174-182.
34. VITÁNYI, P.M.B., AND SAVITCH, W.J. On inverse deterministic pushdown transductions. *J. Comput. Syst. Sci.* 16 (1978), 423-444.

RECEIVED JUNE 1978; REVISED JUNE 1979; ACCEPTED AUGUST 1979