Finite-Memory Automata
(Extended abstract)

Michael Kaminski and Nissim Francez

Department of Computer Science
Technion - Israel Institute of Technology
Haifa 32000, Israel

Abstract. A model of computation dealing with *infinite alphabets* is proposed. This model is based on replacing the equality test by *unification*. It appears to be a natural generalization of the classical Rabin-Scott finite-state automata and possesses many of their properties.

1. Introduction

In this paper we introduce a model of computation dealing with *infinite alphabets*, a natural generalization of the classical Rabin-Scott finite-state automata ([2]). In doing so, we are aiming towards a very restrictive model, capable of recognizing only the natural analog of *regular languages* over finite alphabets. In addition, we would like our model, and the class of languages recognizable by it, to enjoy as many of the properties of the family of finite automata and regular languages as possible. Thus, we are interested in preserving *closure* under Kleene operations and Boolean operations as well as in having decidable inclusion and emptyness properties and structural properties such as the *pumping lemma*. We succeed in doing so, except for closure under *complementation*, which is achievable only by passing to a bit stronger model.

Clearly, this can not be achieved by preserving finite-stateness in its strict meaning. On the other hand, allowing for arbitrary infinite-state automata is obviously too powerful, as it renders *every* language recognizable, by having a state corresponding to each letter in the infinite alphabet. Thus, we need a very restricted memory structure of the automaton, so that it will not be able "to take advantage" of its memory capabilities beyond what is needed for our purposes. We also want to preserve the *computability* of the recognizable languages, so the model should not be powerful enough to do *coding*, or even *counting*.

The basic idea is to equip the automaton with a finite set of *proper* states (like in the classic model), in which the "real computation" is done. In addition, the automaton is equipped with a finite set of registers, each capable of being *empty*, or storing a symbol from the infinite alphabet. We refer to these registers as "*windows*". The restricted power of the automaton is obtained by highly restricting its transition relation. Here, the novel idea is to replace the *equality test* (of the next input symbol to some element of the finite alphabet), which underlies the classic model, by some (extended form of) *unification*. What the automaton does with the next input symbol is the following: If no window contains the input symbol, then it is *copied* into a specified window (depending on the state). Otherwise, the automaton "remembers" that the input symbol has been previously read, in which case the usual equality test applies. As it turns out, relating the next input symbol to *many* windows simultaneously, as well as initializing some windows to some constant letters from the infinite alphabet, lead to an easier development of the theory without strengthening the model.

Thus, by restricting the manipulative power of the automaton to copying and comparing only, without the ability to apply *any* modification functions, all the automaton is capable of doing is to "remember" some bounded number of previously read symbols. Therefore, recognizable languages will have the usual characteristics of regular languages. Typically, it is able to detect the presence of a specific letter, or the appearance of simple patterns such as a repetition of the same letter, etc.

The idea of replacing equality test with unification originates from [4], where it was applied to an infinite alphabet of a very specific structure, where the letters had the form $r(x_i, x_j)$ and were interpreted as binary relation symbols. Words over this alphabet have a structure bearing strong

relationship to *Datalog*, a useful data-base query language ([5]). Other useful interpretations of infinite alphabets are not hard to imagine. Finite sequences of simple patterns occur in many contexts within computer science. *Actions* of concurrent processes, where concurrency and communication are restricted to very simple patterns, are another possible interpretation of infinite alphabets.

A beneficial by-product of our theory is an ability to represent in a more compact way some classical finite-state automata, in case the finite alphabet is so large to best treated as if it is infinite. Thus, *process identifiers* for example (in computer networks) will usually be natural numbers. Any practical network will have some bound on these numbers. However, such bounds may be determined in some complicated, architecture-dependent way, and at a higher level of abstraction it is best to ignore these bounds and consider arbitrary natural numbers.

An important facet of our theory is a certain *indistinguishability* view of the infinite alphabet embedded in the modus operandi of the automaton. Languages are only unique up to *automorphisms* of the alphabet. Thus, the *actual* letters occurring in the input are of no real significance. Only the initial and repetition patterns matter. This follows from the nature of unification. If a *new* letter (i.e., one not in any window) is copied and later successfully compared, any other new letter, having appeared in the same position, would cause the same transitions. Therefore, the usual notion of "*pumping*" (both upwards and downwards), has to be somewhat modified to accommodate this indistinguishability.

It is not hard to envisage extensions of our theory to pushdown finite-memory automata, or to ω-words over infinite alphabets. Here, we restricted the presentation to the (analog of the) regular case only. However, the results have non-trivial proofs and establish the basic techniques that might be applicable to such extensions.

The paper is organized as follows. In the next section we define the model of computation and state some of its basic properties. Section 3 contains a decidability result. Section 4 deals with closure properties, and in Section 5 we present a "syntactical" description of recognizable languages and a pumping lemma adopted to the case of unification.

Finally, the proofs of the results stated below are left to the full paper. These proofs are *constructive*.

Acknowledgement

The part of the second author was partially funded by the Fund for the Promotion of Research in the Technion.

2. Definitions and basic properties

In this section we define the model of computation and state some of its basic properties.

Let Σ be an infinite alphabet and let # be a symbol not belonging to Σ. An *assignment* is a word $w_1 w_2 \cdots w_r \in (\Sigma \cup \{\#\})^*$ such that if $w_i = w_j$ and $i \neq j$, then $w_i = \#$. I.e., an assignment is a word where each symbol from Σ appears at most one time. Accordingly to the informal description of the model presented in the introduction, assignments represent the contents of the registers of the automaton: the symbol in the ith register (window) is w_i. If $w_i = \#$, then the ith window is empty. In our model we assume that a symbol cannot simultaneously appear in two or more windows. This restriction does not weaken the model.

For a word $\boldsymbol{w} = w_1 w_2 \cdots w_N \in (\Sigma \cup \{\#\})^*$ we define the *contents* of $\boldsymbol{w}$, denoted $[\boldsymbol{w}]$, by $[\boldsymbol{w}] = \{ w_i : i = 1, 2, \ldots, N \} - \{\#\}$, i.e., $[\boldsymbol{w}]$ consists exactly of those symbols of Σ which appear in $\boldsymbol{w}$.

Definition 1. A *finite-memory automaton* is a system $\boldsymbol{A} = < S, p, \boldsymbol{u}, \rho, T, F >$, where

- S is a finite set of *states*.
- $p \in S$ is the *initial* state.
- $\boldsymbol{u} = w_1 w_2 \cdots w_r \in (\Sigma \cup \{\#\})^r$ is the *initial* assingment - registers' initialization.
- $\rho : S \to \{1, 2, \ldots, r\}$ is a function from S to $\{1, 2, \ldots, r\}$ called the *reassingment*. The intuitive meaning of ρ is as follows: If $\boldsymbol{A}$ is in state s and the input symbol appears in no windows, then $\boldsymbol{A}$ "forgets" the contents of the $\rho(s)$th window and copies the input symbol into that window.
- $T \subseteq S \times \{1, 2, \ldots, r\} \times S$ is the *transition* relation. The intuitive meaning of T is as follows: If $\boldsymbol{A}$ is in state s, the input symbol is equal to the contents of the ith window, and $(s, i, t) \in T$, then $\boldsymbol{A}$ may enter

state t.

- $F \subseteq S$ is the set of *final* sets.

Similarly to a finite automaton, $\boldsymbol{A}$ can be represented by its initial assingment and a directed graph whose nodes are the elements of S. There is an edge from a node with the first label s to a node with the first label t, if there exists i such that $(s, i, t) \in T$. Such edge is labeled i. Also for each $s \in S$, the node s is labeled $\rho(s)$. For graph representation of finite-memory automata, cf. Example 1 below.

An actual state of $\boldsymbol{A}$ is an element of S together with the contents of all of the windows. Thus $\boldsymbol{A}$ has *infinitely* many states which are pairs $(s, \boldsymbol{w})$, where $s \in S$ and $\boldsymbol{w}$ is an assignment of length r. They are called *configurations* of $\boldsymbol{A}$. The set of all the configurations of $\boldsymbol{A}$ is denoted by S^c. The pair $p^c = (p, \boldsymbol{u})$ is called the *initial* configuration, and the configurations with the first component in F are called *final* configurations. The set of final configurations is denoted by F^c.

The relation T induces the following relation T^c on $S^c \times \Sigma \times S^c$.

Let $\boldsymbol{w} = w_1 w_2 \cdots w_r$ and $\boldsymbol{v} = v_1 v_2 \cdots v_r$. Then $((s, \boldsymbol{w}), \sigma, (t, \boldsymbol{v})) \in T^c$ if and only if either of the two following conditions is satisfied.

If $\sigma = w_k \in [\boldsymbol{w}]$, then $\boldsymbol{v} = \boldsymbol{w}$ and $(s, k, t) \in T$.

If $\sigma \notin [\boldsymbol{w}]$, then for $j \neq \rho(s)$, $v_j = w_j$, $w_{\rho(s)} = \sigma$, and $(s, \rho(s), t) \in T$, $i = 0, 1, \ldots, n-1$.

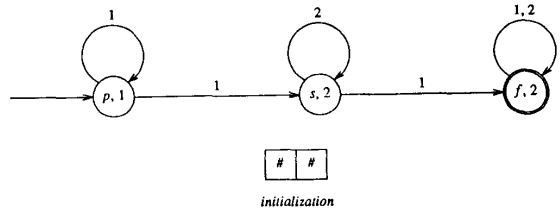
Let $\boldsymbol{\sigma} = \sigma_1 \sigma_2 \cdots \sigma_n$ be a word over Σ. A *run* of $\boldsymbol{A}$ on $\boldsymbol{\sigma}$ consists of a sequence of configurations $c_0, c_1, \ldots, c_n$ such that $c_0 = p^c$ and $(c_{i-1}, \sigma_i, c_i) \in T^c$, $i = 1, 2, \ldots, n$.

We say that $\boldsymbol{A}$ *accepts* $\boldsymbol{\sigma} \in \Sigma^*$, if there exists a run $c_0, c_1, \ldots, c_n$ of $\boldsymbol{A}$ on $\boldsymbol{\sigma}$ such that $c_n \in F^c$. The set of all the words acceptable by $\boldsymbol{A}$ is denoted by $L(\boldsymbol{A})$ and is referred to as a *quasi-regular* set.

Example 1. Consider a finite-memory automaton $\boldsymbol{A} = < \{p, s, f\}, p, \#\#, \rho, T, \{f\} >$, where

- $\rho(p) = 1$, $\rho(s) = \rho(f) = 2$; and
- $T = \{(p, 1, p), (p, 1, s), (s, 1, f), (s, 2, s), (f, 1, f), (f, 2, f)\}$.

This automaton has the following graph representation:



One can easily verify that $L(\boldsymbol{A}) = \{\sigma_1 \sigma_2 \cdots \sigma_n$: there exist $1 \leq i < j \leq n$ such that $\sigma_i = \sigma_j\}$, e.g., an accepting run of $\boldsymbol{A}$ on *abcbd* is $(p, \#\#), (p, a\#), (s, b\#), (s, bc), (f, bc), (f, bd)$. We shall see in the sequel that the complement of $L(\boldsymbol{A})$ to Σ^* is not quasi-regular.

Since the restriction of the set of configurations to a finite alphabet is finite, we immediately have the following result.

Proposition 1. *Let $\boldsymbol{A} = < S, p, \boldsymbol{u}, \rho, T, F >$ be a finite-memory automaton and let Δ be a finite subset of Σ. Then $L(\boldsymbol{A}) \cap \Delta^*$ is a regular language (over Δ).*

The following two propositions reflect the fact that a finite-memory automaton can only distinguish between "new" input symbols, i.e., ones appearing in the contents of the most recent assingment. It cannot, however, distinguish between different "new" symbols.

Proposition 2. (Closure under automorphisms) *Let $\boldsymbol{A} = < S, p, \boldsymbol{u}, \rho, T, F >$ be a finite-memory automaton. Then for each automorphism $\iota: \Sigma \to \Sigma$ that is an identity on $[\boldsymbol{u}]$ and each $\boldsymbol{\sigma} \in \Sigma^*$ we have that $\boldsymbol{\sigma} \in L(\boldsymbol{A})$ if and only if $\iota(\boldsymbol{\sigma}) \in L(\boldsymbol{A})$.*

Proposition 3. (Indistinguishability property of finite-memory automata) *Let $\boldsymbol{A} = < S, p, \boldsymbol{u}, \rho, T, F >$ be a finite-memory automaton, where the length of $\boldsymbol{u}$ is r. If $\boldsymbol{xy} \in L(\boldsymbol{A})$, then there exists a subset $\Sigma' \subseteq [\boldsymbol{x}]$ such that the cardinality of Σ' does not exceed r and the following holds. For any $\sigma \notin \Sigma'$ and any $\tau \notin [\boldsymbol{y}] \cup \Sigma'$, the word $\boldsymbol{x}(\boldsymbol{y}(\sigma/\tau))$ obtained from $\boldsymbol{xy}$ by the substitution of τ for each occurrence of σ in $\boldsymbol{y}$ belongs to $L(\boldsymbol{A})$.*

It immediately follows from Proposition 3 that for the finite-memory automaton $\boldsymbol{A}$ presented in Example 1 the complement of $L(\boldsymbol{A})$ to Σ^* is not quasi-regular. This proves the following result.

Proposition 4. *Quasi regular sets are not closed under complementation.*

In Section 4 we introduce a stronger model that achieves closure under complementation.

3. A decision problem

In this section we sketch the proof of the decidability of inclusion problem for finite-memory automata, cf. the corollary to Theorem 1. The idea of the proof is to associate with quasi-regular sets ordinary regular sets which locally inherit the quasi-regular sets' structure, and then to compare these associated regular sets. For this we need some definitions and preliminary results.

Let $\Gamma = \{\gamma_i : i = 1, 2, \ldots\}$ and $\Delta = \{\delta_i : i = 1, 2, \ldots\}$ be disjoint infinite sets each disjoint from $\Sigma \cup \{\#\}$. For a positive integer N the alphabets $\{\gamma_i\}_{i=1,\ldots,N}$ and $\Delta = \{\delta_i\}_{i=1,\ldots,N}$ are denoted by Γ_N and Δ_N, respectively. For a finite subset W of Σ we define a finite alphabet $\Phi_{N,W}$ by

$$\Phi_{N,W} = W \cup \Gamma_N \cup \Delta_N .$$

Let $\boldsymbol{A}$ be a finite-memory automaton whose initial assignment $\boldsymbol{u}$ is a word of length r. We define a finite alphabet $\Phi_{\boldsymbol{A}}$ by

$$\Phi_{\boldsymbol{A}} = \Phi_{r,[\boldsymbol{u}]} .$$

Definition 2. Let $\boldsymbol{A} = <S, p, \boldsymbol{u}, R, T, F>$ be a finite-memory automaton, $\sigma = \sigma_1\sigma_2 \cdots \sigma_n \in \Phi_{\boldsymbol{A}}^*$, and let $(s_0, w_0), (s_1, w_1), \ldots, (s_n, w_n)$ be an accepting run of $\boldsymbol{A}$ on σ. We say that $(s_0, w_0), (s_1, w_1), \ldots, (s_n, w_n)$ is a *characteristic* run if the following conditions are satisfied.

1. For each $i = 1, 2, \ldots, n-1$, if $\sigma_i = \gamma_m \in \Gamma$, then $\delta_m \notin [w_{i-1}]$ and if $\sigma_i = \delta_m \in \Delta$, then $\gamma_m \notin [w_{i-1}]$.
2. For every $i, j, k = 1, 2, \ldots, n-1$, $i < j < k$, if $\sigma_i = \sigma_k = \gamma_m(\delta_m) \in \Gamma(\Delta)$, and $\gamma_m(\delta_m) \notin [w_j]$, then there is an $l = i+1, i+2, \ldots, k-1$, such that $\sigma_l = \delta_m(\gamma_m)$.

Point 1 of the definition says that γ_m and δ_m cannot simultaneously appear in the memory of $\boldsymbol{A}$ and Point 2 of the definition says that γ_m and δ_m alternatingly appear in the memory of $\boldsymbol{A}$.

We say that $\sigma = \sigma_1\sigma_2 \cdots \sigma_n \in \Phi_{\boldsymbol{A}}^*$, is *strongly acceptable* by $\boldsymbol{A}$ if there exists is a characteristic run of $\boldsymbol{A}$ on σ. The set of all strongly acceptable words of $\Phi_{\boldsymbol{A}}^*$ is denoted by $L_s(\boldsymbol{A})$.

Lemma 1. *The language $L_s(\boldsymbol{A})$ is regular.*

The language $L_s(\boldsymbol{A})$ in some sense is a "free" generating language for $L(\boldsymbol{A})$. Namely, the relation between $L(\boldsymbol{A})$ and $L_s(\boldsymbol{A})$ is given by Lemma 2 below. In order to state the lemma we need the following definition.

Definition 3. Let $\sigma = \sigma_1\sigma_2 \cdots \sigma_n \in \Sigma^*$ and let $\phi = \phi_1\phi_2 \cdots \phi_n \in \Phi_{N,W}^*$. We say that ϕ *covers* σ, if the conditions below are satisfied.

1. If $\phi_i \in W$, then $\sigma_i = \phi_i$, $i = 1, 2, \ldots, n$.
2. If $\phi_i = \gamma_l(\delta_l)$, $i < j$, and there is no k such that $i < k < j$ and $\phi_k = \delta_l(\gamma_l)$, then $\sigma_i = \sigma_j$ if and only if $\phi_i = \phi_j$, $i, j = 1, 2, \ldots, n$, $l = 1, 2, \ldots, N$.

The covering relation can be thought as some kind of "local" isomorphism. Namely, all the γ_l's (δ_l's) between consecutive appearances of $\delta_l(\gamma_l)$ are mapped to the same element of Σ.

Lemma 2. *If $\phi \in L_s(\boldsymbol{A})$ and ϕ covers σ, then $\sigma \in L(\boldsymbol{A})$.*

Definition 4. Let $\phi = \phi_1\phi_2 \cdots \phi_n \in \Phi_{N_1,W_1}^*$ and $\psi = \psi_1\psi_2 \cdots \psi_n \in \Phi_{N_2,W_2}^*$. We say that ψ *reflects* ϕ, if the conditions below are satisfied.

1. If $\psi_i \in W_2$, then $\phi_i = \psi_i$, $i = 1, 2, \ldots, n$.
2. For $i, j = 1, 2, \ldots, n$ and $l = 1, 2, \ldots, r$ and $i < j$, if $\psi_i = \gamma_l(\delta_l)$, and there is no k such that $i < k < j$ and $\psi_k = \delta_l(\gamma_l)$, then the following holds.
   A. If $\phi_i = \gamma_m(\delta_m)$, then there is no k such that $i < k < j$ and $\phi_k = \delta_m(\gamma_m)$.
   B. $\psi_i = \psi_j$ if and only if $\phi_i = \phi_j$.

Let L_1 and L_2 be languages over Φ_{N_1,W_1} and Φ_{N_2,W_2}, respectively. We say that L_1 reflects L_2 if for each $\phi \in L_2$ there exists $\psi \in L_1$ such that ψ reflects ϕ.

Notice that reflection possesses local isomorphism's properties similar to those of covering. There is the following relationship between reflection and covering.

Lemma 3. *Let $\sigma \in \Sigma^*$, $\phi \in \Phi_{N_1,W_1}$, and $\psi \in \Phi_{N_2,W_2}$. If ψ reflects ϕ and ϕ covers σ, then ψ covers σ.*

Lemma 4. *It is decidable for finite-memory automata $\boldsymbol{A}$ and $\boldsymbol{B}$ whether $L_s(\boldsymbol{A})$ reflects $L_s(\boldsymbol{B})$.*

Theorem 1. *Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be finite-memory automata. Then $L(\boldsymbol{A}) \subseteq L(\boldsymbol{B})$ if and only if $L_s(\boldsymbol{B})$ reflects $L_s(\boldsymbol{A})$.*

By Theorem 1 and Lemma 4, we obtain the desired decision property.

Corollary. *The relation $L(\boldsymbol{A}) \subseteq L(\boldsymbol{B})$ is decidable.*

4. Closure properties of quasi-regular languages

In this section we present some closure properties of quasi-regular languages. The proofs are based on the standard construction adapted to a slightly modified version of finite-memory automata. This version allows to relate the next input symbol to *many* windows simultaneously, as well as initializing some windows to some constant letters from the infinite alphabet.

Theorem 2. *The quasi-regular sets are closed under union, intersection, concatenation and Kleene star.*

It can be shown that quasi-regular languages are not closed under homomorphisms, because in order to accept the (inverse) homomorphic image of a quasi-regular language, a finite-memory automaton may need to "remember" infinitely many identities over the input alphabet. Also, by Proposition 4, quasi-regular languages are not closed under complementation. However the closure under complementation is a very desired property. To circumvent this deficiency, we can extend the class of quasi-regular languages as follows.

Definition 5. A *system of finite-memory automata* S consists of a finite number of pairs of finite-memory automata

$$\boldsymbol{S} = [(\boldsymbol{A}_1, \boldsymbol{B}_1), (\boldsymbol{A}_2, \boldsymbol{B}_2), \ldots, (\boldsymbol{A}_N, \boldsymbol{B}_N)]$$

and defines the language

$$L(\boldsymbol{S}) = \bigcup_{i=1}^{N} (L(\boldsymbol{A}_i) \cap \overline{L(\boldsymbol{B}_i)}).$$

This definition is motivated by a similar construction for ω-regular languages in [3]. Let $L_1, L_2, \ldots, L_n$ be quasi-regular languages and let B be a boolean combination of these languages. Then there are index sets $I_k, J_k \subseteq \{1, 2, \ldots, n\}$ such that we can present B in the $\cup\cap$-normal form:

$$B = \bigcup_k ((\bigcap_{i \in I_k} L_i) \cap (\bigcap_{j \in J_k} \overline{L}_j)) = \bigcup_k ((\bigcap_{i \in I_k} L_i) \cap \overline{(\bigcup_{j \in J_k} L_j)})$$

Since quasi-regular languages are closed under union and intersection, we have $B = \bigcup_k (L_k^{\cap} \cap \overline{L_k^{\cup}})$, where $L_k^{\cap} = \bigcap_{i \in I_k} L_i$ and $L_k^{\cup} = \bigcup_{j \in J_k} L_j$ are quasi-regular languages. Thus we have the following result.

Proposition 5. *Languages definable by systems of finite-memory automata are closed under boolean operations.*

In order to complete the picture we need a decision property.

Proposition 6. *The emptiness problem for the languages definable by systems of finite-memory automata is decidable.*

Remark 1. It can be shown that Propositions 1 and 2 hold for the languages definable by a system of finite-memory automata as well.

5. A syntactic characterization of quasi-regularity

In this section we give a syntactic condition for a subset of Σ^* to be quasi-regular. This condition is motivated by Lemma 2 and is as follows.

Theorem 3. *A language L is quasi-regular if and only if there exist N, W and a regular language L^c over $\Phi_{N,W}$ such that L is the maximal subset of Σ^* covered by the elements of L^c.*

Example 2. Let

$$L = \{ \sigma_1, \sigma_2, \ldots, \sigma_n : n = 4k+2, k \geq 0, \sigma_1 = \sigma_3, \sigma_{n-2} = \sigma_n, \text{ and } \sigma_{2i} = \sigma_{2i+3} \}.$$

It can be easily verified that L is the maximal subset of Σ^* that is covered by a regular language $\gamma_1 \{\gamma_2\gamma_1\delta_1\gamma_2\delta_2\delta_1\gamma_1\delta_2\}^*\gamma_1 \subseteq \Phi^*_{\emptyset,2}$. Thus, by Theorem 3, L is quasi-regular.

Let $n = 4k+2$, $k > 0$, and let $\tau_0, \tau_1, \ldots, \tau_{2k}$ be distinct elements of Σ. Consider a word $\boldsymbol{w} = \sigma_1, \sigma_2, \ldots, \sigma_n$, where $\sigma_1 = \sigma_3 = \tau_0$, $\sigma_{n-2} = \sigma_n = \tau_{2k}$, and $\sigma_{2i} = \sigma_{2i+3} = \tau_i$ for $i = 1, 2, \ldots, 2k-1$. (The set of such words can be thought as a set of free generators for L.) One can show that $\boldsymbol{w}$ contains no non-empty pattern that may be pumped. Thus L does not satisfy the usual pumping lemma for regular languages. However, there is an "indistinguishable" version of pumping lemma for quasi-regular languages. This result, although being weaker than the pumping lemma for regular languages, reflects our unification approach.

Theorem 4. (Pumping lemma for quasi-regular languages) *For any quasi-regular language L there exists a constant N such that each word $\boldsymbol{\sigma} \in L$ of the length not smaller than N can be written in the form $\boldsymbol{\sigma} = \boldsymbol{xyz}$, where $\boldsymbol{y}$ is non-empty and the following holds. For each $n = 0, 1, \ldots$ there exist $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n$ each isomorphic to $\boldsymbol{y}$ and $\boldsymbol{z}'$ isomorphic to $\boldsymbol{z}$ such that the word $\boldsymbol{xy}_1\boldsymbol{y}_2 \cdots \boldsymbol{y}_n\boldsymbol{z}'$ belongs to L.*

Remark 2. In view of Proposition 3, the pattern $y_1y_2\cdots y_nz'$ whose existence is provided by Theorem 4 can be replaced by any "indistinguishable" word.

References

[1] J.E. Hopcroft and J.D. Ullman, *Introduction to automata theory, languages, and computation*, Addison Wesley, Reading, Massachusetts, 1979.

[2] M.O. Rabin and D. Scott, Finite automata and their decision problems, *IBM Journal of Research and Development* 3 (1959), 114-125.

[3] M.O. Rabin, *Automata on infinite objects and Church's problem*, Published for Conference Board of the Mathematical Sciences by the American Mathematical Society No. 13, Providence, R.I., 1972.

[4] Y. Shemesh and N. Francez, Finite state Datalog automata and relational languages, submitted to *Information and Computation*.

[5] J.D. Ullman, *Principles of database and knowledge-base systems*, Volume 1, Chapter 3, Computer Science Press Inc., Rockville, Maryland, 1988.