

Games on Infinite Trees and Automata with Dead-Ends
A new Proof for the Decidability of the Monadic Second Order
Theory of Two Successors

An.A.Muchnik

Institute of New Technologies

Kirovogradskaja 11, Moscow 113587, Russia.

e-mail: amuchnik @ globlab.msk.su.

A central problem of mathematical logic and the theory of algorithms is the problem of the decidability of logical theories, that is the problem of constructing an algorithm which distinguishes which formulae of a given language belong to the theory (are true in a given semantics, provable in a given deductive system etc.).

The study of decision problems has shown that a large number of naturally arising theories are undecidable, the desired algorithm does not exist. At the same time, from the point of view of applications, either inside or outside mathematics, the most important cases are exactly those for which such an algorithm can be constructed.

One of the basic results concerning the decidability of logical theories is the Theorem of the decidability of the monadic second order theory of several successors. This Theorem, proven by M.O.Rabin in 1969 [1], yields as simple corollaries

many of the results on decidability known to that time. Since then, it has been used many times in applications, namely in proving the decidability of non-classical logics, such as the logic of programs.

However, in his lecture at the International Congress of Mathematics held in Nice in 1970, M.O.Rabin formulated the problem of simplifying his proof. The first problem of his lecture [2] is:

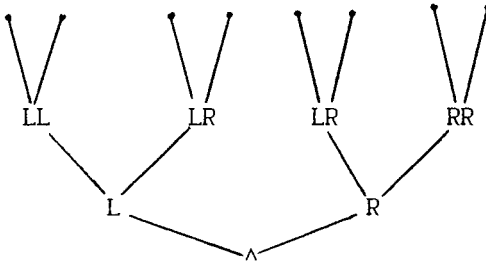
"1. Find a simpler proof for Theorem 2(ii), possibly avoiding the transfinite induction used in [2]." (Theorem 2(ii) is the key statement of Rabin's proof, [2] is paper [1] of our references). Indeed, Rabin's proof, beyond the technical complications, has a more fundamental inconvenience. In order to obtain a perfectly constructive result, it uses a very strong instrument - the principle of transfinite induction.

In 1978, the author of this paper gave a new and simpler proof of Rabin Theorem without using transfinite induction. This proof was presented in the course on decidable theories given in 1978/79 at the Faculty of Mechanics and Mathematics of Moscow State University by A.L.Semenov and it was the subject of the author's graduation paper. The present version differs only in small details from the version given there.

Another proof, close to ours, was presented in May, 1982 at the Symposium on the Theory of Computing [3]. Some time ago, D.Muller and P.Schupp proposed the use of alternating finite automata to prove the Rabin Theorem.

§1. THE MONADIC THEORY OF TWO SUCCESSORS AND FINITE AUTOMATA ON TREES.

Let us give the definition of the monadic theory of two successors, denoted by S2S, which is formed by all formulae of a certain language true in a certain interpretation. The language contains both individual and set (= monadic predicate) variables; its atomic formulae are of the form $x \in Q$, $L(x,y)$, $R(x,y)$, where x,y are individual variables and Q is a set variable. Both individual and set variables can be bounded by quantifiers in the formulae of the language. To describe the interpretation we consider the binary tree of finite words in the alphabet $\{L,R\}$:



Words in this alphabet will be called also vertices of the tree. Values of individual variables are vertices of the tree, values of set variables are arbitrary sets of vertices. The interpretation of atomic formulae is the following: $x \in Q$ states that vertex x is a member of Q , $R(x,y)$ is interpreted as the fact that word y is obtained by adding letter R to the end of x . $L(x,y)$ is understood in a similar way. Theory S2S is thereby fully described.

Theorem of Rabin. *Theory S2S is decidable.*

The proof proceeds in the same way as Rabin's, converting the problem of the decidability of S2S into that of demonstrating some properties of finite automata. What is new is the method of demonstrating the most complicated of these properties. The object of this paper is to present this method. To make our exposition self-contained, we shall recall all necessary definitions concerning finite automata.

We shall use the variant of the theory S2S in which formulae contain only set variables and atomic formulae are of the form $P \in Q$, $\text{Vert}(P)$, $R(P,Q)$, $L(P,Q)$. Their interpretation is as follows. $\text{Vert}(P)$ indicates that P is a one-element set; $P \in Q$ indicates that P is one-element and its unique element belongs to Q ; $R(P,Q)$ and $L(P,Q)$ mean that P and Q are one-element, $P=\{x\}$, $Q=\{y\}$ and x,y satisfy $R(x,y)$ ($L(x,y)$), respectively). It is obvious that this variant of S2S is equivalent to the preceding one from the point of view of decidability.

Let Σ be an alphabet. A Σ -tree is a binary tree whose vertices are labeled by the letters of Σ , i.e. a total mapping from the set of all vertices to Σ . We shall define below the notion of an automaton on Σ -trees and the notion of acceptance of a Σ -tree by an automaton. In this way, to each automaton corresponds a set of all Σ -tree accepted by the automaton. Such sets of Σ -trees will be called *recognizable*.

Let us associate with any set of vertices of a binary tree a $\{0,1\}$ -tree assigning 1 to the vertices of this subset and 0 elsewhere. Likewise, with each n -tuple $\langle P_1, \dots, P_n \rangle$ of sets of

vertices, we associate a $\{0,1\}^n$ -tree. A set of n -tuples of the form $\langle P_1, \dots, P_n \rangle$ will be called *recognizable* if its associated set of $\{0,1\}^n$ -trees is recognizable.

Theorem 1. *Let $A(P_1, \dots, P_n)$ be a formula of theory S2S, all parameters of which are among P_1, \dots, P_n . Then the set of all $\{0,1\}^n$ -trees corresponding to those tuples of values of P_1, \dots, P_n which make $A(P_1, \dots, P_n)$ true is recognizable. The corresponding automaton can be effectively constructed whenever $A(P_1, \dots, P_n)$ is given.*

The proof of this Theorem proceeds by induction on the structure of formula A . The main difficulty here is to construct for a given automaton \mathcal{U} another automaton \mathcal{Q} which accepts precisely those trees which are not accepted by \mathcal{U} . A new method of doing this construction forms the basis part of this paper (§3).

To prove the decidability of S2S, it is sufficient to have, in addition to this Theorem, an algorithm deciding whether or not the set of trees accepted by the automaton is empty. Such an algorithm is given in [1]. In the present paper we give the more simple method for constructing of such algorithm (§2).

Let us now proceed to some definitions and remarks concerning finite automata.

Automata on ω -words

We begin with the important notion of an automaton on ω -words, even though in this paper it plays only an auxiliary

role.

If Σ is an alphabet, an ω -word over Σ is an infinite sequence of letters of Σ . An automaton \mathcal{U} on ω -words is given by:

1) a finite set S of states; the elements of 2^S (subsets of S) will be called *macrostates*;

2) a table of transitions, i.e. a subset $P \subset S \times \Sigma \times S$; for $\langle s, a, s' \rangle \in P$, we say that if the automaton \mathcal{U} is in state s when reading letter a then it can move to state s' ;

3) a subset $S_0 \subset S$ of initial states;

4) a subset $F \subset 2^S$ of final macrostates.

An automaton is called *deterministic* if it has exactly one initial state and its table of transitions is the graph of an everywhere defined function from $S \times \Sigma$ to S (being in any state and reading any letter of Σ , the automaton can move to exactly one state).

Let \mathcal{U} be an automaton on ω -words over Σ and let A be an ω -word over Σ . A *run of \mathcal{U} on A* is an infinite sequence of states which may occur when \mathcal{U} reads A . There may be many runs as well as none. But it is clear that for a deterministic automaton, there is always exactly one run on the given ω -word. More precisely, a run is such a sequence (ω -word) $s_0 s_1 \dots$ of states of \mathcal{U} that s_0 is an initial state (i.e. $s_0 \in S_0$) and, for each i , if \mathcal{U} is in state s_i and reads a_i , then it can move to state s_{i+1} . We can associate with any ω -word its *limit* - the set of all symbols occurring in it an infinite number of times. A run whose limit is a final macrostate is called *accepting*. We say that \mathcal{U} *accepts an ω -word A* if there is an accepting run of \mathcal{U} on A . Otherwise, we say that \mathcal{U} *rejects A* . In such a way, to every

automaton there corresponds a set of ω -words, namely the set of all ω -words accepted by this automaton. Such a set of ω -words is called *recognizable*. It is well-known ([4]) that any recognizable set can be accepted by a deterministic automaton.

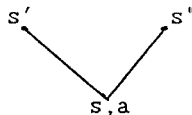
Automata on trees

We recall that a binary tree the vertices of which are labelled by the letters of an alphabet Σ is called *tree over Σ* (or Σ -tree). An automaton on Σ -trees is given by

1) a finite set S of *states* with a distinguished initial state $s_0 \in S$; (subsets of S will be called *macrostates* as before);

2) a *table of transitions* - i.e. a subset of the set $S \times \Sigma \times S \times S$.

Element $\langle s, a, s', s'' \rangle$ of this set will be represented by the following scheme:



A scheme belonging to the table of transitions of \mathcal{U} is called a *transition* of \mathcal{U} and we say that, if it happens that at a certain vertex, \mathcal{U} is in state s and reads a , then it may be at the vertices immediately succeeding this vertex in states s', s'' ;

3) a list of *final macrostates* (each of them being, of course, a subset of S).

Given an automaton on a Σ -tree, there are many *possible*

runs of the automaton on the tree. Each run is an assignment of states to the vertices of the binary tree according to the table of transitions and such that the initial state is assigned to the root of the tree. More precisely, a run of \mathcal{U} on Σ -tree A is an S -tree H satisfying the following condition: the initial state is at the root of H , and if x is an arbitrary vertex of the binary tree which in A is labelled by a and in H is labelled by s , and if the vertices xL , xR of run H are respectively labelled by s' and s'' , then $\langle s, a, s's'' \rangle$ is an element of the table of transitions of \mathcal{U} . Now any run has many limit macrostates (which is not the case for the automata on ω -words): there is one for each path. More precisely, let $x_0 x_1 \dots$ be an infinite path in the binary tree (i.e. a sequence of vertices in which $x_0 = \lambda$ and x_{i+1} is one of the two immediate successors of x_i). If a run H and a path are given, then the letters which label the vertices lying along the path form an ω -word. The limit of this ω -word (i.e. the set of states occurring in it infinitely many times) is called the *limit macrostate of the run corresponding to the given path*. A run is called *accepting* if all limit macrostates (corresponding to all paths) are final.

We say that an automaton *accepts* a tree if there exists an accepting run, i.e. a run for which all the limit macrostates corresponding to all paths are final. Otherwise, when for each run there exists a path whose limit macrostate is not final, we say that the automaton *rejects* the input tree. A set of trees is called *recognizable* if there exists an automaton accepting all the trees of this set and no others. As we said above, our aim is to prove (a) that the problem of emptiness of automate set of

trees is solvable and (b) that the complement of any recognizable set α is recognizable and that the corresponding automaton can be constructed effectively from the automaton accepting α . The remaining part of the present paper is devoted to these proofs.

§2. THE PROOF OF SOLVABILITY OF EMPTINESS OF AUTOMATE SET OF TREES

In this paragraph we shall construct an algorithm, which given an automaton on trees decides whether the set of trees recognized by this automata is empty, and which, moreover, finds the regular tree in this set, if the set isn't empty (the definition of a regular tree is given in the following two lines).

Let us give the definition of a regular Σ -tree. In this definition the notion of a finite deterministic transducer is used. A finite transducer on words is a finite synchronous automaton with output. It has two alphabets: the input one and the output one. After reading current letter of the input alphabet the transducer outputs one letter from the output alphabet. The output letter depends on the input letter and on the current state of the automaton. The transducer outputs also one letter before reading the input word (we'll call this letter *initial*). Thus a finite transducer can be defined by the input alphabet Σ , output alphabet Δ , the set of states S , the initial state S_0 , the initial letter δ_0 and the set of transitions.

Formally, the set of transition is a everywhere defined function from $S \times \Sigma$ into $S \times \Delta$.

Definition. A Σ -tree is regular if there is a finite transducer with the input alphabet $\{L,R\}$ and output alphabet Σ that after reading any word u over $\{L,R\}$ outputs the mark of the vertex u in this tree. (Thus the mark of the root will be the initial letter.)

Any transducer satisfying this definition will be called the *transducer generating the tree*.

Theorem 2. 1) There is an algorithm that given an automaton on Σ -trees decides whether the set recognized by the automaton is empty. 2) Any nonempty recognizable set of Σ -trees has a regular Σ -tree and if this set is recognized by an automaton with n states then there is a transducer with $n!$ states generating a Σ -tree in this set.

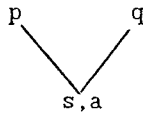
Proof. a) Firstly, let us proof the existence of a transducer generating a Σ -tree in recognizable nonempty set of Σ -trees. This will be proved by induction, the parameter of induction is the number of states of the automaton on trees that recognizes the set. In order to make the induction step we must prove a more general assertion. Let us state it. Let us introduce the notion of a Σ -tree with dead ends. Let D be a finite set disjoint with Σ . The elements of D will be called *dead-ends*.

Definition. A Σ -tree with dead-ends from D is any subtree of the whole binary tree each vertex of which has either 0 sons (is a leaf) or 2 sons, the leaves of which are marked by symbols from D and the internal vertices of which are marked by letters

from Σ .

Evidently every Σ -tree is also a Σ -tree with dead-ends from D (for any D).

Let us give the definition of a Σ -automaton with dead-ends from D . The only difference with a Σ -automaton is that a Σ -automaton with dead-ends from D can have transitions of the form



where p or q or both are dead-ends from D . That is the set of transitions is a subset of the set $S \times \Sigma \times (SUD) \times (SUD)$. The sets S and D must be disjoint. A possible run of a Σ -automaton with dead-ends from D on a Σ -tree T with dead-ends from D is any tree H that can be obtained from T by assignment to each internal vertex of D a state in such a way that the root is marked by initial state and the obtained marking is consistent with the set of transitions. That is, for every internal vertex x of H if a is a mark of x from Σ , s is the assigned mark from S and p and q are respectively the marks from SUD of the vertices xL and xR then the tuple $\langle s, a, p, q \rangle$ belongs to the set of transitions. An ordinary automaton on Σ -trees is a particular case of a Σ -automaton with dead-ends from D (for any D). A possible run is called *accepting* if the limit macrostates of all its infinite paths are final.

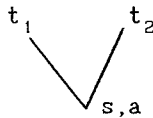
Now let us extend the notion of a regular tree to the trees with dead-ends. A transducer with input alphabet $\{L, R\}$ and output alphabet SUD generates a Σ -tree T with dead-ends from D

if after reading any word v over $\{L,R\}$ it outputs the mark of the vertex v in T .

We will prove by induction the following assertion. *If a Σ -automaton \mathcal{U} with dead-ends from D accepts some Σ -tree with dead-ends from D then \mathcal{U} accepts some Σ -tree with dead-ends from D generated by a transducer with at most $n!$ states.* The parameter of induction is the number of states of \mathcal{U} (other parameters can be arbitrary). Let us call for brevity the automata accepting nonempty sets the *nonempty automata*.

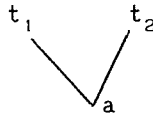
Base of induction. Let \mathcal{U} be a nonempty Σ -automaton with dead-ends from D having only one state. As \mathcal{U} accepts some Σ -tree with dead-ends the set of transitions isn't empty. Let us denote the single state by s . Consider two cases.

1). There is a transition of the form



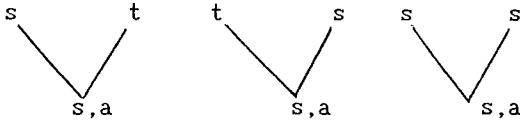
where t_1, t_2 are dead-ends.

Then \mathcal{U} accepts the tree with dead-ends

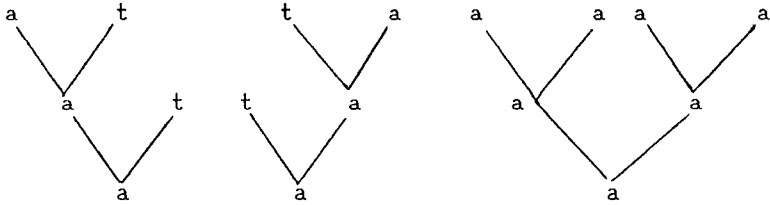


generated by a transducer with one state.

2). There is no transition with two dead-ends on the top. In this case every accepting run has infinite path therefore the macrostate $\{s\}$ is final. There is a transition of one of the following forms



where t is a dead-end. Correspondingly \mathcal{U} accept one of the following trees



All these trees are generated by a transducer with 1 state.

Induction step. Let a nonempty Σ -automaton \mathcal{U} with dead-ends from D have $n \geq 2$ states. We will regard the states as the residues modulo n . Thus for every state k we can speak about the state $k+1$. Without loss of generality we can assume that \mathcal{U} has single initial state (as the recognized by \mathcal{U} set is the union of the sets recognized by the automata obtained from \mathcal{U} by leaving only one initial state). Let the initial state be 0 . Let us define $2n$ new automata $\mathcal{B}_0, \dots, \mathcal{B}_{n-1}, \mathcal{C}_0, \dots, \mathcal{C}_{n-1}$, having $n-1$ states.

\mathcal{B}_k is obtained from \mathcal{U} by the following transformation. The set of states of \mathcal{B}_k is $\{0, \dots, n-1\} \setminus \{k\}$ and the set of dead-ends is D . All transitions in which the state k occurs are deleted and all macrostates having k are deleted.

\mathcal{C}_k is obtained from \mathcal{U} by the following transformation. The initial state of \mathcal{C}_k is k . The set of states is $\{0, \dots, n-1\} \setminus \{k+1\}$ and the set of dead-ends is $D \cup \{k+1\}$. All the transitions having $(k+1)$ on the bottom are deleted and all the macrostates having

$(k+1)$ are deleted.

Let us consider two cases

Case_1. For some k the automaton \mathfrak{B}_k is nonempty. Let us define a regular Σ -tree with dead-ends from D accepted by \mathfrak{U} . Let \mathfrak{U}' be the automaton obtained from \mathfrak{U} by deleting k from the set of states, adding k to the set of dead-ends, deleting all transitions having k on the bottom and deleting all macrostates having k . Let us prove that \mathfrak{U}' is nonempty. Pick some Σ -tree T with dead-ends from D accepted by \mathfrak{U} . Let us fix an accepting run H of \mathfrak{U} on T . Let T' consist of all vertices u of T such that no ancestor of u is marked by k in H (u itself can be marked by k). Evidently T' is a Σ -tree with dead-ends from $D \cup \{k\}$. Evidently it is accepted by \mathfrak{U}' .

By induction hypothesis there is a Σ -tree T_1 with dead-ends from $D \cup \{k\}$ generated by a transducer with $(n-1)!$ states and accepted by \mathfrak{U}' . Again by induction hypothesis there is a Σ -tree T_2 with dead-ends from D accepted by \mathfrak{B}_k and generated by a transducer with $(n-1)!$ states. Let us "glue" to all leaves of T_1 marked by k the Σ -tree T_2 and erase the mark k in that leaves (now all leaves that were marked by k are marked by the root mark of T_2). We have obtained a Σ -tree T_3 with dead-ends from D .

Firstly, T_3 is accepted by \mathfrak{U} : the accepting run of \mathfrak{U} is obtained by the same gluing of an accepting run of \mathfrak{U}' on T_1 and an accepting run of \mathfrak{B}_k on T_2 . Indeed, every infinite path in the obtained run either belongs to the run of \mathfrak{U}' (and hence has final limit macrostate) or from some place is in the run of \mathfrak{B}_k (and hence also has final limit macrostate).

Secondly T_3 is generated by the transducer obtained by

gluing the transducers generating T_1 and T_2 : in those states in which the first transducer outputs the dead-end k the new transducer outputs the initial letter of the second transducer and switch it on. The number of states of the new transducer is $(n-1)!+(n-1)! \leq n!$ (as $n \geq 2$).

Case 2. For all k the automaton \mathfrak{B}_k is empty. Let us prove that in this case the *whole* macrostate $\{0,1,\dots,n-1\}$ is final and for all k the automaton \mathfrak{C}_k is nonempty. By condition \mathcal{U} accepts some Σ -tree T with dead-ends from D . Let us fix some accepting run H of \mathcal{U} on T . Let us prove that H has an infinite path with final macrostate $\{0,1,\dots,n-1\}$. As \mathfrak{B}_0 is empty there is a vertex v_0 in H marked by 1. As \mathfrak{B}_1 is empty there is a vertex v_1 in H that is marked by 2 and follows vertex v_0 (that is $v_1 = v_0 u$ for some word u over $\{L,R\}$). And so on. We have found an infinite path in H which passes through all states $0,1,\dots,n-1$ infinitely many times. As H is an accepting run, the macrostate $\{0,1,\dots,n-1\}$ is final.

Let us prove that for all k the automaton \mathfrak{C}_k is nonempty. Let us take arbitrary k . Let us fix again a run H of \mathcal{U} on a Σ -tree T with dead-ends from D . We have already proved that some vertex v of H is marked by k . Consider the Σ -tree $H_1 = \{u|v u \in H\}$, in which the vertex u is marked as the vertex $v u$ in H . Consider the subtree H_2 of H_1 consisting of all vertices u such that no their ancestor is marked by $(k+1)$ (u itself can be marked by $k+1$; in this case we delete the Σ -mark of u). Evidently we have obtained an accepting run of \mathfrak{C}_k on some Σ -tree with dead-ends from $D \cup \{k+1\}$.

Now let us construct a regular Σ -tree with dead-ends from D

accepted by \mathcal{U} . By induction hypothesis for each k there is a Σ -tree T_k with dead-ends from $D \cup \{k+1\}$ accepted by \mathcal{G}_k and generated by a transducer with $(n-1)!$ states. The Σ -tree T with dead-ends from D is constructed as follows. Let us take T_0 . Let us "glue" to all leaves of T_0 marked by 1 the tree T_1 (and delete the mark 1 of those leaves). Let us "glue" the tree T_2 to all leaves marked by 2 of the obtained tree. And so on. Let us denote the tree obtained by repeating this procedure ω times by T . Obviously T is a Σ -tree with dead-ends from D . We claim that \mathcal{U} accepts T and T is regular.

Let us prove that \mathcal{U} accepts T . To this end let us fix for every k an accepting run H_k of \mathcal{G}_k on T_k . Let us perform with H_0, \dots, H_{n-1} the same "gluing" as with T_0, \dots, T_{n-1} . We'll obtain a run H of \mathcal{U} on T . Every infinite path in H either from some place belongs to some run of \mathcal{G}_k for some k (therefore has final limit macrostate) or passes infinitely many times through all states (therefore its limit macrostate is $\{0, 1, \dots, n-1\}$, which is final).

The tree T is regular because it can be generated by "gluing" the transducers generating T_0, \dots, T_{n-1} . The number of its state is equal to $n \cdot (n-1)! = n!$.

b). Let us now prove that given a Σ -automaton \mathcal{U} with dead-ends from D we can decide whether it is empty and construct the transducer generating the accepted tree with dead-ends. Our algorithm is recursive (in recursive calls the number of states decreases). Let us be given \mathcal{U} . Consider two cases.

Case 1. \mathcal{U} has single state s . If the set of transitions is empty then \mathcal{U} accepts empty set. Otherwise we look if there is a

transition with two dead-ends on the top. If there is such a transition then \mathcal{U} is nonempty. Otherwise \mathcal{U} is empty if and only if the macrostate $\{s\}$ is final. Obviously we can construct the transducer with single state generating an accepted tree if it exists.

Case 2. \mathcal{U} has $n \geq 2$ states. Making the recursive calls we check whether there is k such that \mathcal{B}_k is nonempty. If there is such k then \mathcal{U} is empty if and only if \mathcal{U}' is empty (\mathcal{U}' is defined in the item a)). Making the recursive call we can decide whether \mathcal{U}' is empty.

Otherwise (if for all k , \mathcal{B}_k is empty) \mathcal{U} is nonempty if and only if the macrostate $\{0, 1, \dots, n-1\}$ is final and for all k , \mathcal{C}_k is nonempty (this is in fact proved in the proof of case 2 of item a)). Making recursive calls we can decide whether this is true. The transducer is constructed as it was described.

The proof is finished.

It turns out that the bound $n!$ in theorem 2 cannot be improved.

Theorem 3. *There is a constant c such that for all n there is a nonempty set of $\{0, 1\}$ -trees accepted by an automaton with cn states and having no $\{0, 1\}$ -tree generated by a transducer with less than $n!$ states.*

Proof. Let us introduce the notion of an n -ary Σ -tree ($n \in \mathbb{N}$). Recall that binary Σ -tree is a mapping from the set of all words over $\{L, R\}$ into Σ . Likewise a n -ary Σ -tree is a mapping from the set of all words over the alphabet $\{1, \dots, n\}$ into Σ . If x_0, x_1, x_2, \dots is a path in n -ary Σ -tree and $x_i = x_{i-1} a_i$, $a_i \in \{1, \dots, n\}$ then a_i will be called the *direction of the path*

on i -th level. The notions of an automaton on Σ -trees and of a transducer can be naturally generalized to n -ary Σ -trees.

Firstly we will construct for every natural n a nonempty set of n -ary $\{1,2,\dots,n\}$ -trees accepted by an automaton with cn states and having no tree generated by a transducer with less than $n!$ states. Then this set of trees will be transformed into the set of binary $\{0,1\}$ -trees with the same properties. This will complete the proof.

Let us fix $n \in \mathbb{N}$ and define the set M of n -ary Σ -trees where $\Sigma = \{1,2,\dots,n\}$. Let T be a n -ary Σ -tree. Define in what case T belongs to M . Let us take an arbitrary path in T . Let $a_1, a_2, \dots, a_k, \dots$ be its directions and $p_0, p_1, p_2, \dots, p_k, \dots$ be its marks (that is p_k is the mark of the vertex $a_1 \dots a_k$ in T). We'll call p_k the *mark of the path on k -th level*. Let N be the set of all $a \in \{1, \dots, n\}$ that occur infinitely many times in the sequence $a_1, a_3, a_5, \dots, a_{2k+1}, \dots$. Briefly speaking, N is the limit set of the directions on odd levels. Let P be the limit set of the *marks* on *even* levels. Let us call the path *correct* if the maximal element of P is equal to the number of elements of N (that is $\max P = |N|$). By definition T belongs to M if all paths in T are correct.

Let us prove that M isn't empty. To this end we define a transducer with $2n!$ states generating a tree in M . Reading the sequence $a_1, a_2, \dots, a_k, \dots$ of directions the transducer computes the so called "sequence of last occurrences of the directions on odd levels". Let us define this sequence. This is a permutation of the set $\{1,2,\dots,n\}$. In the beginning it is equal to $(1,2,\dots,n)$ and after reading the current direction a_{2k+1} on an

odd level the current permutation Π_{2k} is transformed as follows: a_{2k+1} is moved in the beginning of Π_{2k} (the previous occurrence of a_{2k+1} is deleted). Let us define the output of the transducer. On the odd steps (i.e. after reading $a_1 \dots a_{2k+1}$) it outputs 1. Let us define the output symbol on an even step, say after reading $a_1 \dots a_{2k+1}$. Let l_{2k+1} be the length of the prefix of Π_{2k} that was changed on the step $2k+1$ (i.e. l_{2k+1} is equal to the number of the occurrence of a_{2k+1} in Π_{2k}). The transducer outputs this l_{2k+1} on the step $2k+2$. The initial letter of the transducer is 1.

Let us prove that the tree generated by the defined transducer belongs to M. Let us fix an arbitrary path in this tree. Let N be the limit set of the directions on odd levels of this path. Then for all sufficiently large k the prefix of Π_k of the length |N| is a permutation of elements of N therefore the number output by the transducer on the k-th level will be less than or equal to |N|. On the other hand, as every direction from N occurs infinitely many times on odd levels, the |N|-th element of the sequence infinitely many times is moved in the beginning therefore the transducer will infinitely many times output |N|.

Let us prove that M is accepted by an automaton with $2n$ states. The states of this automaton are of two types: the first type consists of one state for every direction and the second type consists of one state for every letter from Σ (altogether $2n$ states). On the even levels (i.e. after reading $a_1 \dots a_{2k}$) the state of automaton is equal to the mark of the current vertex (i.e. the mark of $a_1 \dots a_{2k}$). On the odd levels (i.e. after reading $a_1 \dots a_{2k+1}$) the state of automaton is equal to the last

direction (i.e. a_{2k+1}). The initial state is arbitrary state of the second type. The final macrostates are those states in which the maximum of the states of the second type is equal to the number of states of the first type.

Let us prove that there is no transducer with less than $n!$ states generating a Σ -tree in M . Assume that \mathcal{B} is a transducer with less than $n!$ states generating a tree in M .

Let $p=(v_1, \dots, v_n)$ be a permutation of the set $\{1, \dots, n\}$ of directions. Consider the word

$$u_p = (\dots((v_1^{n!} v_2)^{n!} \dots v_n)^{n!}$$

where w^k denotes $w w \dots w$, k times. We'll call the word

$$w_i = ((\dots((v_1^{n!} v_2)^{n!} \dots v_{i-1})^{n!} v_i$$

the *block of level i* . Thus the block of level i consists of $n!$ blocks of level $(i-1)$ and of the letter v_i and u_p consists of $n!$ blocks of level n . Define for any word u the word \bar{u} to be the word obtained by inserting 1 before each letter of u (for example $\overline{010}=101110$). Let \mathcal{B} work on the input word \bar{u}_p . Consider the states of \mathcal{B} before reading the occurrences of \bar{w}_i in \bar{u}_p . As the number of occurrences is greater than the number of states there are two occurrences of \bar{w}_n in \bar{u}_p and a state s_n such that \mathcal{B} is in the state s_n before reading these two occurrences. Let us call these two occurrences of \bar{w}_n in \bar{u}_p *marked*. Let now \mathcal{B} start in the state s_n and work on the input word \bar{w}_n . Again there are two occurrences of \bar{w}_{n-1} in \bar{w}_n and a state s_{n-1} such that \mathcal{B} is in the state s_{n-1} before reading these two occurrences. Let us call these two occurrences of \bar{w}_{n-1} in \bar{w}_n *marked*. Let us call the marked occurrences of \bar{w}_{n-1} in the marked occurrences of \bar{w}_n in \bar{u}_p also marked. Thus we have four marked occurrences of \bar{w}_{n-1} in \bar{u}_p .

Repeating this procedure n times we get a state s_1 and 2^n marked occurrences of \bar{w}_1 in \bar{u}_p such that if \mathcal{B} reads \bar{u}_p then before reading each marked occurrence of \bar{w}_1 in \bar{u}_p it is in the state s_1 . As s_1 depends on p we'll write $s_1(p)$. As the number of permutations of the set $\{1, \dots, n\}$ is greater than the number of states there are two different permutations p and p' such that $s_1(p) = s_1(p')$. Let $p = (v_1, \dots, v_n)$, $p' = (v_1', \dots, v_n')$. Let $i \leq n$ is defined by the equalities $v_1 = v_1', \dots, v_{i-1} = v_{i-1}'$, $v_i \neq v_i'$.

Let us prove that the word u_p has a subword y over $\{v_1, \dots, v_i\}$ having occurrences of all the letters v_1, \dots, v_i and such that if \mathcal{B} works on input word \bar{u}_p then \mathcal{B} is in the state $s_1(p)$ before and after reading \bar{y} . Indeed, let us pick a marked occurrence of \bar{w}_{i+1} in \bar{u}_p . It has two marked occurrences of \bar{w}_i (we'll call them left and right). Let us pick in each of these two occurrences a marked occurrence of \bar{w}_1 . Take the part of \bar{u}_p from the occurrence of \bar{w}_1 in the left occurrence of \bar{w}_i in \bar{u}_p (including \bar{w}_1) up to the occurrence of \bar{w}_1 in the right occurrence of \bar{w}_i in \bar{u}_p (excluding \bar{w}_1). This part is equal to \bar{y} for some word y over $\{v_1, \dots, v_i\}$. The word y satisfies the required conditions. Let us denote the word obtained from u_p , by the same procedure by y' .

Now let us pick a word x such that after reading \bar{x} , \mathcal{B} is in the state $s_1(p)$. Consider three ω -words over Σ

$\bar{x} \bar{y} \bar{y} \bar{y} \bar{y} \dots$

$\bar{x} \bar{y}' \bar{y}' \bar{y}' \bar{y}' \dots$

$\bar{x} \bar{y} \bar{y}' \bar{y} \bar{y}' \bar{y} \bar{y}' \dots$

Any of these three ω -words defines an infinite path in n -ary tree.

Evidently the limit sets of the directions on odd levels of these paths are respectively equal to

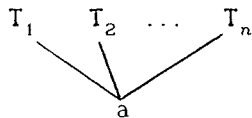
- $\{v_1, \dots, v_{i-1}, v_i\}$
- $\{v_1, \dots, v_{i-1}, v_i'\}$
- $\{v_1, \dots, v_{i-1}, v_i, v_i'\}$.

On the other hand let P and P' are the sets of letters output by \mathcal{B} on even levels when it starts in the state $s_1(p)$ and reads respectively \bar{y} and \bar{y}' . Then the limit sets of the letters output by \mathcal{B} on even levels on these ω -words are respectively equal to

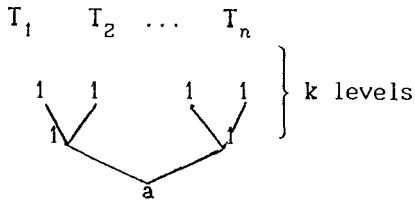
- P
- P'
- POP'

Thus we have $\max P = l$, $\max P' = l$, $\max(POP') = l + 1$. Contradiction.

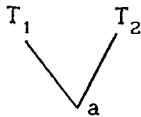
We have constructed the nonempty set M of n-ary $\{1, \dots, n\}$ -trees accepted by a deterministic automaton with cn states and having no tree generated by a transducer with $<n!$ states. Let us construct the set of binary $\{0, 1\}$ -trees with these properties. Clearly without loss of generality we may assume that $n = 2^k$ for some k. For every n-ary $\{1, \dots, n\}$ -tree T let us define its "counterpart" \bar{T} , which is a binary $\{0, 1\}$ -tree. Let us do this in two steps. On the first step let us define a binary $\{1, \dots, n\}$ -tree \bar{T} as follows. Each "fork" in T the form



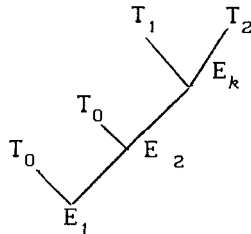
$(T_1, T_2, \dots, T_n \text{ are trees})$ is replaced by the "fork"



The obtained binary tree is \bar{T} . Then we replace in \bar{T} every "fork" of the form



(T_1, T_2 are trees) with the "fork"



where $E_1 \dots E_k$ is the binary code of the letter a (under some fixed coding of the elements of $\{1, \dots, n\}$ with binary words of length k) and T_0 is the binary tree marked with only zeros. The obtained tree is \bar{T} .

Define $\bar{M} = \{\bar{T} \mid T \in M\}$. One can easily prove that \bar{M} is accepted by a deterministic automaton with cn states. The proof of the lower bound $n!$ can be easily transformed to the proof of the same bound for \bar{M} .

The theorem is proved.

§3. PROOF OF THE RECOGNIZABILITY OF COMPLEMENTATION

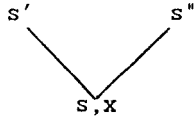
Strategies

Let \mathcal{U} be an automaton on Σ -trees. Our aim is to construct an automaton \mathcal{L} accepting precisely those trees which are rejected by \mathcal{U} . This will be done in two steps. First, we introduce the notion of "rejecting strategy" for a given automaton on a given tree. Such a strategy will exist if and only if the automaton rejects the tree. In the second step we shall construct an automaton \mathcal{L} having an accepting run if and only if there exists a rejecting strategy for \mathcal{U} . This will be the required automaton.

A rejecting strategy is a strategy for finding a path with a non-final limit on any run of the given automaton. We look for such a path step by step, starting at the root of the tree and proceeding in the following way. Let a run X be given. In the first step, we consider the transition at the root and we choose one of the two possible directions - left or right. Suppose that the left is chosen. In the next step, we consider the transition we have in the left vertex of the first level of run X (vertex L) and, once again, we choose a direction. Thus, at each step, a direction is chosen (by respecting the transition observed in this step) and a move to the next vertex in the chosen direction is effected. In this way, applying a strategy to run X , we get an infinite path in the run. A rejecting strategy applied to any run must give a path with a non-final limit.

Let us describe the notion of a strategy more formally.

Suppose that automaton \mathcal{U} and input tree T are given. Consider all transitions possible at the root of T , i.e. all transitions of the form



where s is the initial state of \mathcal{U} and x is the letter labelling the root of T . Let us divide them in some way into "left transitions" and "right transitions". If a run X starts with a left (right) transition, then the strategy will seek in this run a path with a non-final limit going in the first step to the left (right, respectively). Having made this division, we have defined the *states possibly occurring at L during the search*. Let us call them *states possible at L*, or in short *L-possible states*. In other words, they are the states which are top left in the transitions that have been chosen as left ones. We similarly define *R-possible states* or *states possible at R*. (Note that the sets of L-possible and R-possible states can be empty). Furthermore, we divide the transitions of \mathcal{U} having at the bottom an L-possible state and the letter labelling L in the input tree (*the L-possible transitions*), into the left and right ones. This division determines how the strategy will seek a path with a non-final limit in the second step. Afterwards, we define which states are possible at the vertices LL and LR . Similarly, having divided the R-possible transitions into left and right ones, we define the states possible at the vertices RL and RR , and so on.

To give a *strategy for automaton \mathcal{U} on input tree T* means to decide for each vertex which of the transitions possible at this vertex should be considered as left ones and which as right ones. Note that the set of states possible at a given vertex (and, hence, the set of transitions possible at this vertex) depends on the decision made in the preceding step.

So, let a strategy for the given automaton \mathcal{U} and input tree T be given. The strategy seeks, as described, a path in each run of \mathcal{U} . The strategy is called *rejecting* if all the paths it defines, for all runs of \mathcal{U} , have non-final (for \mathcal{U}) limits, and moreover, each probable path of the given strategy has a non-final limit. By a *probable path of the given strategy* (the automaton and the input trees are fixed) we mean a path which can be obtained as follows. First of all, choose one of the transitions possible at the root. Then look at what the strategy proposes - to consider this transition as left or as right. Let it be left. Take the state which is top left in this transition. It is one of the L-possible states. Then, choose an L-possible transition starting in this state. Again consult the strategy. Suppose this transition is right. Take the state which is top right in this transition. This state is LR-possible. And so on. If this process does not terminate because of the absence of possible transitions starting in the given state, we obtain a path in the tree and the states lying along it. These paths, obtained in this way, are called probable paths of the given strategy. Every path obtained by applying the strategy to a run of automaton \mathcal{U} on tree T is probable. The inverse implication is, however, not generally true because a probable path requires

a selection of transitions only "lying along" the path and we do not know whether this selection can be extended to a run on the whole tree.

So far, we have introduced the notion of a strategy (for a given automaton on a given input tree) and we have distinguished the rejecting ones. The existence of a rejecting strategy for \mathcal{U} on T is sufficient for \mathcal{U} to reject T . To make this condition necessary, we have to change the notion of strategy by accepting the existence of a "memory of finite size" in it. The above given notion of strategy - a preliminary one - becomes a particular case of the more general notion of strategy that we shall present in the next section.

Modification of the notion of strategy
- strategy with memory.

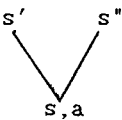
A strategy with memory differs from the previous one when it is deciding how to continue to seek a path with a non-final limit in a given run. The new strategy considers not only the transition at the vertex it is working on but also the history of the search reflected in the "inner state". Actually, applying the strategy to two different runs, it is possible to reach the same vertex in both cases, with the same transitions at this vertex. The existence of a "memory" allows it to go, in one case to the left, and in the second, to the right.

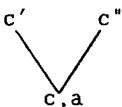
The new notion of strategy will be defined as follows. Suppose that each state s of \mathcal{U} divides into one or more different *copies* $s_1, s_2 \dots$ (more formally, we are given a set C

of all copies of all states and a surjective mapping φ of C onto S).

Let for every pair $\langle s, s' \rangle$ and every copy c of the state s a copy c' of the state s' be given. c' is called the *result of the application of transition $s \mapsto s'$ to the copy c* . (More precisely, we are given a function $\varphi: S \times C \rightarrow C$ s.t. $\varphi(\langle s', c \rangle) = s'$ for every $s' \in S$ and every $c \in C$. We speak about a mapping from $S \times C$ rather than about a mapping from $S \times S \times C$ because of $s = \varphi(c$.) Finally, let one of the copies of the initial state be fixed and called the *initial copy*. In this situation we say that a *strategy set for automaton \mathcal{U} is given*. We are going to define the notion of a *rejecting strategy for \mathcal{U} on tree T based on the strategy set M* . If M contains exactly one copy of each state, then the new definition coincides with the old one.

Previously, the chosen strategy was fixing in each vertex a set of possible states, now, instead of this, we shall have a set of possible copies. (Note that it may happen that one copy of a state will be possible and another copy of the same state will not).

Let  be a transition of \mathcal{U} and let c be a copy of

the state s . Consider the scheme , where c', c'' are

obtained as results of the application of transitions $s \mapsto s'$ and $s \mapsto s''$ to c . (Note that c' and c'' are copies of s' and s'' , respectively.) All transitions of this form (obtained by applying all transitions of \mathcal{U} to all copies c) will be called *copy-transitions*. The copy-transitions, with a copy possible in

some vertex at the bottom are called *copy-transitions possible* at this vertex. The strategy divides copy-transitions possible at a vertex into the left and the right ones. (Two copy-transitions obtained from the same "ordinary" transition need not to be of the same direction). This division defines the sets of copies possible in the successor vertices: for example, the set of copies possible in the vertex xL is the set of all copies which occur at the left top of copy-transitions that are possible in x and related by the strategy to the left ones. The initial vertex has exactly one possible copy - the initial copy (of the initial state). The probable paths of the strategy are defined as before with the following modification: each path is related to a sequence of copies (and not to a sequence of states, as previously). However, when defining the notion of a rejecting strategy, we shall be interested only in states (remaining indifferent to which copy of the given state occurs). That is, a *strategy* is called *rejecting* if for each probable path the corresponding sequence of states has a non-final limit.

Now, let a rejecting strategy for \mathcal{U} on tree T based on the strategy-set M be given, and moreover, let a run on T be given. How can we find in this run, using the rejecting strategy, a path with a non-final limit? This demands selecting a copy in each transition, starting at the bottom and moving upwards. We start at the initial copy of the initial state. Then we look at the copy-transition which lies in the root. Suppose it to have been designated as the left. Hence, we have to move to the left into the vertex L . The copy-transition which lies there is a possible copy-transition and it was, again, designated as left

or right. Depending on that, we have to move either to the vertex LL or to the vertex LR. And so on. The path we thus get will be one of the probable paths and, therefore, its limit will be non-final. We have now proved the implication (2) \mapsto (1) of the following statement.

Theorem 4. *For every automaton \mathcal{U} there exists such a finite strategy-set M that for any tree T the conditions*

(1) " \mathcal{U} rejects T "

and

(2) "there exists a rejecting strategy for \mathcal{U} on T based on M "

are equivalent.

Our first aim is to give the proof of this theorem by showing that for a suitable strategy-set M , (1) implies (2). Then, the only remaining thing to establish will be that the existence of a rejecting strategy for \mathcal{U} on T based on M is equivalent to the acceptance of the tree T by another automaton.

Beginning of the Proof of Theorem 4:

Introduction of Dead-ends in Automata and Strategies

To prove Theorem 4, we shall have to generalize it by introducing dead-ends to automata on trees. Let Δ be a finite set of dead-ends. Instead of Σ -trees, we shall consider $\Sigma \times P(\Delta)$ -trees and not the Σ -trees with dead-ends from Δ as in §2, i.e. trees the vertices of which are labelled by both a letter from Σ and a set of dead-ends. The dead-ends of this set will be called dead-ends *allowed in the given vertex*. The definition of

automata with dead-ends was given in §2.

Let us now give the definition of a run of an automaton with dead-ends on a $\Sigma \times P(\Delta)$ -tree. The run of automaton \mathcal{U} on a $\Sigma \times P(\Delta)$ -tree is a subtree of the complete binary tree, in which each vertex has either 0 or 2 successors. The vertices with 0 successors are labelled by dead-ends, those with 2 successors by states. Moreover, each transition occurring here has to be a transition of the automaton \mathcal{U} . End of the definition of a run.

A run is called *accepting* if, first, all its dead-ends are allowed (that is, belong to the $P(\Delta)$ -label of the corresponding vertex) and second, the limits of all infinite paths are final macro-states. We say that a $\Sigma \times P(\Delta)$ -tree is accepted by \mathcal{U} (the automaton with dead-ends) if there is an accepting run; otherwise, we say that the tree is *rejected* by \mathcal{U} . Let us now explain what is a *rejecting strategy* for a given automaton on a given $\Sigma \times P(\Delta)$ -tree based on a given strategy-set M . It seeks, starting in the root, either an infinite path with a non-final limit or a finite path ending in a non-allowed dead-end. When considering the possible copy-transitions, it relates them either to the left ones or to the right ones. What is new is the fact that in some of these transitions there may be one or two dead-ends at the top. (Dead-ends do not have any copies). Suppose for example that in a possible copy-transition, there is, top left, a dead-end and top right a state (or, more exactly, a copy of a state). If this transition is related to the right ones, then we get a possible copy at the right successor vertex; if it is related to the left ones, we get a dead-end in the left successor vertex. In this way we get, in

each vertex, in addition to the set of possible copies, a set of possible dead-ends. For example if a vertex is the left successor (of the preceding vertex) then this set is the set of all dead ends lying at the left hand side of the top of the copy transitions possible at the preceding vertex and related to the left ones. Probable paths will be of two kinds now: infinite ones, defined as before, and finite ones ending in a possible dead-end. We are ready now to formulate the promised generalization of Theorem 4.

Theorem 4'. *Let automaton \mathcal{U} in alphabet Σ and a set Δ of dead-ends be given. Then there exists such a finite strategy set M that for any $\Sigma \times P(\Delta)$ -tree T , the following statements are equivalent:*

(1) " \mathcal{U} rejects T "

(2) "*there exists a rejecting strategy for \mathcal{U} on T based on the strategy set M ".*

The implication (2) \rightarrow (1) can be proved as before.

The implication (1) \rightarrow (2) is going to be proved by induction on the number of states of \mathcal{U} (with an arbitrary number of dead-ends).

The case of the automaton with one inner state.

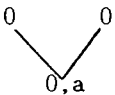
We want to prove Theorem 4' for the automaton with one inner state (denoted in the following by 0) and an arbitrary number of dead-ends. As a strategy set, for this automaton, we

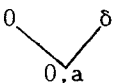
can use the set containing the unique copy of the unique state. We have to prove that one of the two following possibilities is always true: either there is an accepting run or there is a rejecting strategy. Let us consider two cases. First case: macrostate $\{0\}$ is not final. In this case, the limit of any infinite path is not final and any accepting run must be finite. Moreover, the definition of a rejecting strategy requires the limits of all probable paths to be non-final and this condition is here automatically fulfilled. In this case, we shall construct a rejecting strategy assuming that there is no accepting run. In the second case, when macrostate $\{0\}$ is final, the limit of each path is final; and when constructing an accepting run, we have only to verify the condition concerning dead-ends. In this latest case, we shall construct an accepting run assuming that there is no rejecting strategy.

First case: $\{0\}$ is not final.

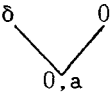
Let no accepting run exist. This means that no transition used at the root can be the beginning of such a run. More precisely, the following Lemma is true.

Lemma 1. Let tree T with the root labelled by $a \in \Sigma$ be rejected by automaton \mathcal{U} .

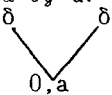
(a) If  is a transition of automaton \mathcal{U} then the subtree with root in L or the subtree with root in R is rejected by \mathcal{U}

(b) If  is a transition of automaton \mathcal{U}

then the dead-end δ is not allowed (in vertex R) or the subtree with root in L is rejected by \mathcal{U} .

(c) If  is a transition of \mathcal{U} then the dead-end δ

is not allowed (in vertex L) or the subtree with root in R is rejected by \mathcal{U} .

(d) If  is a transition of the automaton then at least

one of the two dead-ends is not allowed (in the corresponding vertex).

Proof. This Lemma is almost obvious: if its statement were not true, then it would be possible to construct an accepting run of \mathcal{U} , by "gluing" the described parts (in (a), e.g., "gluing" accepting runs on L-subtrees and R-subtrees).

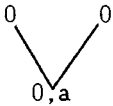
We are now ready to describe a rejecting strategy of \mathcal{U} on T (recall: \mathcal{U} rejects T, $\{0\}$ is a non-terminal macrostate). On the first step, we have to divide all transitions having at the bottom the same letter as the tree T has at its root, into left and right ones. This will be done as follows: the transition is considered as a left one if it has top left either a non-allowed dead-end or the macrostate $\{0\}$ and, in this latest case, the L-subtree is rejected by \mathcal{U} . If neither of those conditions is satisfied, then the transition is considered as a right one. Following Lemma 1, it then has top right either a non-allowed dead-end or the state $\{0\}$ and, in the latter case, the R-subtree is rejected by \mathcal{U} . The previous considerations yield that all dead-ends possible at L and R are not allowed; and, moreover, it

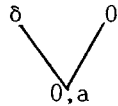
is clear that if the state 0 is possible at one of the vertices L and R, then the subtree with the root at this vertex will be rejected by \mathcal{U} . This gives us the opportunity to divide similarly all transitions possible at L and R into left and right ones. Continuing thus, we finally get a rejecting strategy: the condition for dead-ends is satisfied and, as mentioned before, there is no need to verify the condition for infinite paths. The first case is complete.

Second case: $\{0\}$ is final.

We shall now proceed in the opposite direction: supposing that there is no rejecting strategy, we shall construct an accepting run. So, let no rejecting strategy for \mathcal{U} on the tree T exist. This means that in the first step, it was not possible to assign certain transitions either to the right or to the left in such a way that we could continue with the construction of the strategy. To put it more precisely, we arrive at the following.

Lemma 2. *Let us suppose that there is no rejecting strategy for automaton \mathcal{U} on the tree T, the root of which is labeled by letter a. Then there exists a transition of \mathcal{U} for which one of the following statements holds:*

(a) the transition is of the form  and there is no rejecting strategy for \mathcal{U} on the L-subtree and on the R-subtree;

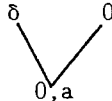
(b) the transition is of the form , there is no rejecting strategy on the R-subtree and δ is an allowed (in L) dead-end;

(c) the transition is of the form $\begin{array}{c} 0 \qquad \delta \\ \diagdown \quad \diagup \\ 0, a \end{array}$, there is no rejecting strategy for \mathcal{U} on the L-subtree and δ is an allowed (in R) dead-end;

(d) the transition is of the form $\begin{array}{c} \delta \qquad \delta' \\ \diagdown \quad \diagup \\ 0, a \end{array}$, where δ and δ' are allowed in L and R dead-ends respectively.

Proof. The Lemma is almost obvious: if such transitions did not exist then we would be able to designate each transition (at the root) as a left one or as a right one and, going on in the same way, to obtain a rejecting strategy. (For example, either the L-subtree or the R-subtree would have a rejecting strategy for any transition of the form $\begin{array}{c} 0 \qquad 0 \\ \diagdown \quad \diagup \\ 0, a \end{array}$; depending on the subtree, we would simply follow the (existing) rejecting strategy on this subtree).

We shall now construct an accepting run (of automaton \mathcal{U} with final macrostate $\{0\}$ which does not have any rejecting strategy on the tree T). Take the transition guaranteed by Lemma 2 and put it in the root of the run. Suppose for example that it is of the form



Then δ is an allowed dead-end and the R-subtree does not have any rejecting strategy. If we apply Lemma 2 to the R-subtree we obtain the next transition, and so on. The resulting run will be accepting: the condition for dead-ends is satisfied and the condition for paths is obvious because of $\{0\}$ being final.

The case of one-state automaton is proved.

Induction step

(description of the strategy set and two lemmas)

So, we are now to prove Theorem 4' for automaton \mathcal{U} with $n+1$ states, assuming it to be true for any automaton with n states (and an arbitrary number of dead-ends). Let us denote the states of \mathcal{U} by $0, 1, \dots, n$ and consider them as the elements of $N/(n+1)$ in order to speak conveniently about state $(i+1)$ as following state i . 0 is considered as the initial state.

For the purposes of proof, we shall introduce some auxiliary automata. \mathcal{U}_i will denote the automaton that has the same table of transitions and the same set of final macrostates as \mathcal{U} , but the initial state of \mathcal{U}_i is i . \mathcal{B}_i will denote the automaton derived from \mathcal{U} by considering state i as the initial state and $i+1$ as a dead-end. This means that the number of states decreases by one (because of the exclusion of $i+1$), and the number of dead-ends increases by one (because of the addition of $i+1$). Further, all transitions having $i+1$ at the bottom are excluded, and all final macrostates containing $i+1$ are excluded, while the transitions having $i+1$ on the top are kept (but $i+1$ is considered as a dead-end and not as a state). Automata \mathcal{B}_i have n states (\mathcal{U} has $n+1$ states). By the induction assumption, for any $i=0, \dots, n-1$, there exists a strategy set M_i which satisfies the following condition: tree T is rejected by automaton \mathcal{B}_i if and only if there exists a rejecting strategy for \mathcal{B}_i on T based on M_i . This strategy set contains a certain

number of copies of each state of \mathfrak{B}_i , hence, the copies of all states of \mathcal{U} except for $i+1$. Assume that sets M_i are pairwise disjoint. We shall now describe the strategy set M for \mathcal{U} . It contains all the copies contained in the sets M_i . (In this way, a copy of i belongs to M if and only if it belongs to one of the sets M_k . Note that M_k does not contain any copy of the state $k+1$). Let us describe what is the result of the action of the transition $i \mapsto j$ on copy c of state i . Let $c \in M_{k_0}$. If the result of the action of the transition $i \mapsto j$ on c is defined in M_{k_0} (i.e. if $j \neq k_0+1$; $i \neq k_0+1$ is always true because c belongs to M_{k_0}), then it will be considered as the result of the action of $i \mapsto j$ on c in M . Otherwise (i.e. when $j = k_0+1$), the result is defined as the initial copy of state j in set M_j . To finish the description of set M , it remains to designate its initial copy. It will be the initial copy of state 0 in M_0 .

The above constructed set M is the required set: automaton \mathcal{U} rejects T if and only if there exists a rejecting strategy for \mathcal{U} based on M . To prove this, we shall need the following two lemmas.

Lemma 3. Automaton \mathcal{U}_i accepts tree T if and only if automaton \mathcal{B}_i accepts tree T' , which is obtained by taking T and adding the dead-end $i+1$ into the vertices which are roots of subtrees accepted by \mathcal{U}_{i+1} .

Before formulating the second lemma, let us introduce a notation: M^i is the strategy set which differs from M in a

unique point - the initial copy is the initial copy of state i in M_i .

Lemma 4. *There exists a rejecting strategy based on M^i for automaton \mathcal{U}_i on tree T if and only if there exists a rejecting strategy (based on M_i) for automaton \mathcal{B}_i on tree T' , which is obtained by taking T and adding the dead-end $i+1$ to the vertices which are roots of subtrees on which there is no rejecting strategy for \mathcal{U}_{i+1} based on M^{i+1} .*

Proof of Lemma 3. This lemma has nothing to do with strategies, and therefore it is simple. If \mathcal{U}_i accepts T , then, cutting the accepting run at the point, where it goes through state $i+1$, we get an accepting run of automaton \mathcal{B}_i on T' : the cut parts guarantee that the dead-ends $i+1$ are allowed.

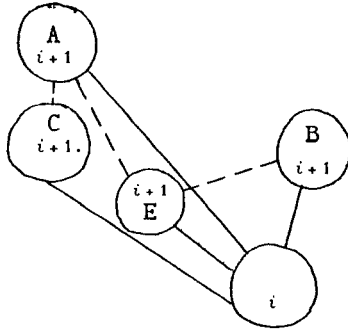
On the other hand, if \mathcal{B}_i has an accepting run on T' , then, "gluing" at the dead-ends $i+1$, which occur in this run, the accepting runs of automaton \mathcal{U}_{i+1} (they exist because the dead-ends $i+1$ are allowed), we get a run of \mathcal{U}_i on T .

Proof of Lemma 4. This lemma is more complicated because of its connection with strategies. To clarify its formulation and understand its analogy with Lemma 3, let us introduce the following terminology. Let us say that an automaton *quasi-accepts* a tree, if there is no rejecting strategy for this automaton on this tree (strategy based on the corresponding set described in the lemma). Then Lemma 4 can be reformulated as follows:

Automaton \mathcal{U}_i quasi-accepts tree T if and only if automaton \mathcal{B}_i quasi-accepts tree T' , which is obtained by taking T and inserting the dead-end $i+1$ into the vertices which are roots of subtrees quasi-accepted by \mathcal{U}_{i+1} .

Hence, suppose that there exists a rejecting strategy on tree T' based on M_i for automaton \mathcal{B}_i . All we need is to construct a rejecting strategy based on M^i for \mathcal{U}_i on T . As first step, we have to divide into left and right the same copy-transitions as in the strategy for \mathcal{B}_i (here, the definition of action of a transition on a copy in M_i is used). In the following step we select the same copies as in the strategy for \mathcal{B}_i except for one case: it is possible that the dead-end $i+1$ has been selected for the strategy for \mathcal{B}_i , we then select (the initial copy in M_{i+1} of) state $i+1$. But as the dead-end has been selected, this means that it is not allowed, in other words, there is a rejecting strategy for \mathcal{U}_{i+1} based on M_{i+1} on the subtree, the root of which is in the vertex labelled by this dead-end. Thus this strategy divides the copy-transitions having at the bottom the initial copy of $i+1$. Therefore, we can go on with the construction of the strategy for \mathcal{U}_i : the copy-transitions at the bottom of which there is a copy of a state different from $i+1$, will be divided into left and right copies as in the strategy for \mathcal{B}_i , and the copy-transitions, at the bottom of which there is the initial copy of $i+1$, will be divided by following the existing strategies for \mathcal{U}_{i+1} . In the following steps, we can proceed in a similar way and get a strategy for \mathcal{U}_i .

However, in the previous analysis, there is an important point we have not discussed. Indeed, the selected copies could



have been selected for different reasons: following a \mathfrak{B}_i -strategy and at the same time following an \mathfrak{U}_{i+1} -strategy, or even several \mathfrak{U}_{i+1} -strategies! The picture represents one of these situations: circles contain selected copies, full lines represent transitions due to \mathfrak{B}_i -strategy, dotted lines represent transitions due to \mathfrak{U}_{i+1} -strategy. In the left top circle, we see a copy A which has been selected for three reasons. One selection has been done according to \mathfrak{B}_i -strategy, the second according to \mathfrak{U}_{i+1} starting at C, and the third according to \mathfrak{U}_{i+1} -strategy starting at E. Copy B has been selected for two reasons: one selection has been made following \mathfrak{B}_i -strategy and another following \mathfrak{U}_{i+1} -strategy starting at E. What should be done in such cases? Which strategy should we follow? Answer: if possible, follow \mathfrak{U}_{i+1} -strategies and, among them, the strategy coming into effect as early as possible. This leads to the goal, namely, the rejecting strategy for \mathfrak{U}_i . Indeed, the condition for dead-ends is satisfied. Let us verify the condition for paths. Take an arbitrary one. Either it is out of range of

\mathcal{U}_{i+1} -strategy (then the limit is not final since \mathcal{B}_i -strategy is rejecting), or, if not, it never leaves this range (because we prefer \mathcal{U}_{i+1} -strategies). It may only happen that it comes into the range of another, earlier \mathcal{U}_{i+1} -strategy. But this is possible only a finite number of times. Hence, beginning at a certain point, the path lies entirely in the range of one \mathcal{U}_{i+1} -strategy and its limit is not final.

We have proved one implication of Lemma 4. The opposite implication is much simpler. Let us have a rejecting strategy for \mathcal{U}_i on T . Consider those selected copies of state $i+1$, which have been selected as the first ones (this means that on the paths they lie on there are no other copies of $i+1$ before them). These copies are initial in M_{i+1} (by the rules of application of transitions to the copies in M). Therefore, subtrees with roots in the vertices labelled by these copies have a rejecting strategy for \mathcal{U}_{i+1} (based on M^t). Hence, by considering these copies as dead-ends, we get a rejecting strategy for \mathcal{B}_i on T based on M_i .

The proofs of Lemma 3 and 4 are complete. In the following section, we shall use in fact only that part of Lemma 3 which has been proved as second and that part of Lemma 4 which has been proved as first. The converse implications were included just to make the statements complete.

Last part of the induction step

What is the outcome of the Lemmas proved in the previous paragraph? Knowing that tree T' of Lemma 3 coincides with tree T'' of Lemma 4, we can use the induction assumption for automaton

\mathfrak{B}_i and get what we need - the existence of an accepting run of \mathcal{U}_i on T would be equivalent to the fact that there is no rejecting strategy for \mathcal{U}_i on T . However, the assertion $T'=T$ states that the existence of an accepting run for \mathcal{U}_{i+1} is equivalent to the fact there being no rejecting strategy for \mathcal{U}_{i+1} , which has the same number of states as \mathcal{U}_i ! Nevertheless, the proved Lemmas are of some value.

We shall consider two cases: in the first, we shall assume that the set S of all states is final, in the second, that it is not. In the first case, we shall apply Lemma 4 and use the same ideas as when proving Lemma 3. In the second, Lemma 3 and ideas of the proof of Lemma 4 will be used.

First case. The set of all states is final. Let us prove that if there is no rejecting strategy on T for \mathcal{U}_0 , then there is an accepting run for \mathcal{U}_0 . As there is no rejecting strategy on T for \mathcal{U}_0 , there is an accepting run on T (constructed in Lemma 4) for \mathfrak{B}_0 (remember that for \mathfrak{B}_0 , by induction assumption, the existence of a run is equivalent to the non-existence of a strategy). For what reason is this not a run for \mathcal{U}_0 ? In some places, it comes to dead-ends 1 allowed in T . Subtrees with roots in these vertices do not have any rejecting strategy for \mathcal{U}_1 , hence, even without having an accepting run for \mathcal{U}_1 (which would permit us to use directly Lemma 3), they have at least a run for \mathfrak{B}_1 , which will be accepting if we add at certain places dead-and 2. But in those places it is possible to start the run of \mathfrak{B}_2 and so on. We get a run for \mathcal{U}_0 by "gluing" all these runs together. It will be accepting: any path either lies in a run of some of the \mathfrak{B}_i 's? (from a certain point) - (and has therefore a

final limit) - or it describes a circle, going infinitely many times from \mathfrak{B}_0 to \mathfrak{B}_1 , from \mathfrak{B}_1 to \mathfrak{B}_2, \dots , from \mathfrak{B}_n to \mathfrak{B}_0 , its limit being equal to the set of all states, which is final by assumption. So much for the first case.

Second case. The set of all states is not final. This case is more complicated; at this point we shall use the structure of the strategy set M we have constructed. Let us prove, supposing that there is no accepting run, the existence of a rejecting strategy. Hence, let us suppose there is no accepting run on tree T for automaton \mathfrak{U}_0 . By Lemma 3, as there is no accepting run for \mathfrak{B}_0 on T' , which was obtained from T by adding in some vertices dead-end 1 there is a rejecting strategy for \mathfrak{B}_0 . What does it lack in order to be an \mathfrak{U}_0 -strategy? It contains at several places possible dead-ends 1. When constructing \mathfrak{U}_0 -strategy, at the same places there appear possible copies of state 1 (initial copies in M_1). What should be done with them? The subtrees with root in the vertices labelled by these copies do not have any accepting run for \mathfrak{U}_1 . We do not know whether there is a rejecting strategy on these subtrees, (which would allow us to use Lemma 4). As there is no accepting run, we can once more apply Lemma 3 and state that by adding dead-end 2, we get trees without accepting runs for \mathfrak{B}_1 on it and hence (by the induction assumption) with a rejecting strategy for \mathfrak{B}_1 . Using these rejecting strategies, it is possible to go on with the construction of \mathfrak{U}_0 -strategy. Note that, until now, we have not met any problematical case (similar to those which were discussed in the proof of Lemma 4), because the \mathfrak{B}_0 -strategy deals with copies from M_0 and \mathfrak{B}_1 -strategy deals with copies from

M_1 . Going on with this procedure, we put strategies for $\mathfrak{B}_2, \dots, \mathfrak{B}_n$ into action. The strategy for \mathfrak{B}_{k+1} comes into action at the places in which the strategy for \mathfrak{B}_k has possible dead-ends $k+1$: in these places, there is no accepting run for \mathfrak{U}_{k+1} . At a certain moment, the circle is completed and it is necessary to use the \mathfrak{B}_0 -strategy, and so on. Now, the question can arise about the choice between the different strategies for \mathfrak{B}_i . The answer is - the one which is put into action earlier. We shall verify whether we indeed get a rejecting strategy for \mathfrak{U}_0 . The condition with dead-ends is obviously satisfied. Let us verify the condition for paths. Every probable path is of one of the two following kinds: either the path, starting from a certain point, goes through the copies of only one strategy set M_1 , or goes infinitely many times from the strategy sets M_j to the following ones, M_{j+1} . In the first case, the path, beginning at a certain place, follows the strategy for \mathfrak{B}_i ; to switch from one strategy to another is possible only if the second one started earlier. Therefore, such switches are possible only finitely many times and the path almost everywhere coincides with the probable path of the strategy for \mathfrak{B}_i and its limit is not final. In the second case, the path passing from M_j to M_{j+1} , goes through the initial copy of M_j and therefore its limit is the set of all states, which is (by assumption) not final.

So, the case when the set of all states is not final, is finished and the inductive proof of Theorem 4' is hence completed. Thus we have proved the above formulated Theorem 4.

Final stage of the proof of the complementation theorem

Corresponding to the announced plan, it remains to prove the following statement.

Theorem 5. *Let Σ be an alphabet, \mathcal{U} an automaton on Σ -trees (without any dead-end!), M a strategy set. Then the set of all Σ -trees on which there exists rejecting strategy for \mathcal{U} based on M is recognizable.*

To prove this statement, introduce the notion of a semiautomaton on Σ -trees. A semiautomaton is given by a set of inner states S , a table of transitions, a set of initial states (until now, everything listed has been the same as in the case of the usual automata on Σ -trees) and moreover, a certain automaton \mathcal{B} on ω -words in the alphabet S . A run is defined in the same manner as in the case of usual automata. It is called accepting if all paths, regarded as sequences of states (i.e. elements of S) are rejected by \mathcal{B} . The sets of trees accepted by semiautomata will be called (temporary) *semirecognizable*. It is clear that any recognizable set is semirecognizable. The following Lemma states that the converse implication is also true.

Lemma 5. *Any semirecognizable set is recognizable.*

Proof. It is known (see [4]) that every (nondeterministic) automaton on ω -words is equivalent to some deterministic one. Hence, the automaton \mathcal{B} , belonging to the given semiautomaton \mathcal{U} , can be considered as deterministic. Then, it is not difficult to construct an automaton which will be equivalent to the semiautomaton \mathcal{U} . Its set of states has to be the product of the

sets of states of \mathcal{U} and \mathcal{B} ; its run is in fact constructed from a run of \mathcal{U} and the runs of \mathcal{B} on ω -words (letters are the states of \mathcal{U}) which lie in the chosen run of along all paths.

Now it remains to prove that the set of all trees, on which there exists a rejecting strategy for \mathcal{U} based on M is semirecognizable. But this is quite clear: it is not difficult to formulate the notion of strategy itself in the form of an accepting run of a semiautomaton. The different possibilities of dividing copy-transitions to left and right ones correspond to the possibilities of continuing the run in different ways. It is possible to consider the states of the semiautomaton as pairs of disjoint sets of copy transitions (the members of the pair correspond to the sets of left and right copy-transitions). Automaton \mathcal{B} will look for the paths which do not satisfy the definition of a strategy (i.e. the paths with final limits) and hence, the acceptance by automaton \mathcal{B} of the sequences of states lying along all paths will indeed mean that the strategy is rejecting.

In this way, we have proved Theorem 4 and hence we have proved that the complement of a recognizable set is recognizable.

Acknowledgments. The author is sincerely grateful to J.Ryšlinkova, A.Shen, W.Thomas and N.Vereshchagin for the help in preparing this text.

References

- [1] Rabin, M.O. Decidability of second order theories and automata on infinite trees. - TAMS, 1969, 171, p.1-35.
- [2] Rabin, M.O. Decidability and definability in second order theories. - In: Actes du Congres des Math., Nice, 1970, 1, p.239-277.
- [3] Gurevich Y., Harrington L. Trees, automata and games. - In: Proceedings ACM Symp. on the Theory of Computing, San Francisco, 1982, May, p.60-65.
- [4] McNaughton R. Testing and generating infinite sequences by a finite automaton. - Information and control, 1966, 9, №5, p.521-530.

