The Method of Forced Enumeration for Nondeterministic Automata

Róbert Szelepcsényi*
Department of Theoretical Cybernetics, Faculty of Mathematics and Physics, Komensky University, Mlynská dolina F 1, 842 15 Bratislava, Czechoslovakia

Summary. Every family of languages, recognized by nondeterministic \( L(n) \) tape-bounded Turing machines, where \( L(n) \geq \log n \), is closed under complement. As a special case, the family of context-sensitive languages is closed under complement. This solves the open problem from [4].

Introduction

This paper presents a new method of simulating nondeterministic Turing machines. The method forces every accepting computation of a nondeterministic Turing machine to examine all reachable configurations of the simulated nondeterministic Turing machine working on an input word \( w \). It is important that the simulating machine does not use more space than the original Turing machine. Using the new method we have proved that every family of languages, recognized by nondeterministic \( L(n) \) tape-bounded Turing machines for \( L(n) \geq \log n \) is closed under complement. As a special case the closure of context-sensitive languages under complement follows. The method was first used in [5] to solve this open problem from [4], but it was found to work for more general families of languages (see [6]). This result has been obtained independently by Neil Immerman (see [2], [3]).

The model of Turing machine we are going to deal with is the nondeterministic Turing machine with one input tape and one work tape infinite in one direction. This paper considers following definition of nondeterministic tape-bounded Turing machines (see [1]).

Definition 1. A nondeterministic Turing machine is said to be \( L(n) \) tape-bounded when it uses at most \( L(n) \) squares of its work tape for every input word of length \( n \).

This definition slightly differs from the other definition used, which requires \( L(n) \) to be space constructible and the existence of an accepting computation which uses at most \( L(n) \) work tape squares. The definition used in this paper

* This research was supported by the grant SP2V I-1-5/08
defines slightly wider class of nondeterministic tape-bounded machines. For constructible \( L(n) \) the families of languages defined by these two kinds of tape-bounded machines are identical.

The main result follows:

**Theorem 1.** Let \( L \) be a language, recognized by an \( L(n) \) tape-bounded nondeterministic machine \( T \), where \( L(n) \geq \log n \). Then there exists an \( L(n) \) tape-bounded nondeterministic machine \( T' \), recognizing the complement \( L' \) of \( L \).

**Proof:** Informal construction of the machine \( T' \):

Let us presume, that \( T'' \) starts to work on an input word \( w \) of length \( n \). Let \( K \) denote the set of all configurations of \( T \). Let us choose an ordering of the set \( K \), in which configurations with short non-empty work tape sections are ordered before configurations with longer non-empty work tape sections.

Obviously not all configurations from the set \( K \) are reachable. Let \( S \) be the subset of all configurations reachable from \( w \). \( T \) accepts \( w \) if and only if \( S \) contains a configuration with an accepting state. The machine \( T' \) will determine if the input word \( w \) is in \( L \) by checking all configurations in \( S \) and will accept \( w \) if and only if it does not find any configuration with an accepting state of \( T \).

Let us suppose that \( T' \) knows the number of the elements of \( S \) (denoted by \( \text{card} \ S \)) and the largest element of \( S \) according to the chosen ordering (denoted by \( \text{max} \ S \)). Then \( T' \) enumerates the set \( S \) as follows: it starts to enumerate all elements from \( K \) in the chosen ordering beginning with the first configuration and ending with \( \text{max} \ S \). For every configuration \( k \) that occurs in this enumeration of \( K \), the machine \( T' \) nondeterministically simulates \( T \) on the input word \( w \). If \( T' \) reaches the configuration \( k \), then \( T' \) has a witness that the configuration \( k \) belongs to \( S \).

However, there is a problem: during this process machine \( T' \) might not select for all \( k \) in \( S \) a correct sequence of steps leading to \( k \). We shall "force" \( T' \) to simulate a computation reaching \( k \) if it exists, as follows: \( T' \) counts all configurations which it finds to be in \( S \) and after completing the enumeration compares this number with \( \text{card} \ S \). If \( T' \) has overlooked at least one configuration belonging to \( S \), the number of the configurations counted will be inevitably lower than \( \text{card} \ S \), since \( T' \) cannot count any configuration that is not in \( S \). In such cases \( T' \) rejects. If the number of all configurations counted is identical with \( \text{card} \ S \), then \( T' \) is guaranteed that \( S \) has been correctly enumerated. If no accepting configuration has been found in \( S \), then \( T' \) accepts.

It remains to show how \( T' \) determines \( \text{max} \ S \) and \( \text{card} \ S \) in space \( L(n) \).

Let \( S_i \) be the set of all configurations reachable by \( T \) working on the input word \( w \) in at most \( i \) steps of computation. Obviously \( S_0 \subseteq S_1 \subseteq \ldots \subseteq S \) and for some \( j \leq \text{card} \ S \), \( S_j = S \) holds.

The set \( S_0 \) only contains the initial configuration of \( T \) with the input word \( w \), which therefore equals \( \text{max} S_0 \) and \( \text{card} S_0 = 1 \). When \( T' \) knows \( \text{max} S_i \) and \( \text{card} S_i \), it determines \( \text{max} S_{i+1} \) and \( \text{card} S_{i+1} \) as follows: First \( T' \) sets the counter of the configurations included into \( S_{i+1} \) to the value \( \text{card} S_i \) thus counting all configurations from \( S_i \). Then \( T' \) enumerates the set \( S_i \) in the same way as described above – with the help of \( \text{max} S_i \) and \( \text{card} S_i \). The procedure verifying