Acta Informatica 26, 279–284 (1988)

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 card S) and the largest element of S according to the chosen ordering (denoted by $\max S$). 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 $\max S$. 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 card S. If T' has overlooked at least one configuration belonging to S, the number of the configurations counted will be inevitably lower than 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 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 $\max S$ and 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 $\max S_0$ and card $S_0 = 1$. When T' knows $\max S_i$ and card S_i, it determines $\max S_{i+1}$ and card S_{i+1} as follows: First T' sets the counter of the configurations included into S_{i+1} to the value 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 $\max S_i$ and card S_i. The procedure verifying that a configuration belongs to S_i differs slightly, since T' simulates at most i steps of T. For each configuration from S_i the machine T' enumerates all configurations that can be reached by T in one step of computation. Thus T' gets all configurations from S_{i+1} so it can easily determine $\max S_{i+1}$. However, the situation concerning card S_{i+1} is more complex.

Let k_0 be a configuration from S_i and k a configuration that can be reached by T from k_0 in a single step. Although k definitely belongs to S_{i+1}, T' cannot simply include it in the total count, since k might also belong to S_i or it might be reachable in a single step from some configuration preceding k_0 in the chosen ordering. In both cases k has already been included in the total count, so T' must not count it again. For this reason T' enumerates in another track of its work tape the set S_i again, and compares each of its members with the configuration k. At the same time it forms from every configuration belonging to S_i and preceding k_0 in the chosen ordering all configurations that can be reached in a single step of T and compares them with k. If identity does not occur in any case, T' includes k into the total count. In this way the machine T' secures that every configuration is counted exactly once.

The above described process is carried out as long as $S_{i+1} \neq S_i$. For some $j \leq \operatorname{card} S$, $S_{j+1} = S_j$ must hold. The set S_j then contains all configurations reachable by T while working on then input word w; i.e. $S_j = S$. The values $\max S_j$ and card S_j evaluated by T' are therefore $\max S$ and card S.

Notice, that if $L(n) \geq \log n$, the above described simulation can be realized in $O(L(n))$ space.

This informal description of T' is presented in more detail in the Appendix.

The above result can be restated in the following form:

Corollary 1. *The family of languages recognized by $L(n)$ tape-bounded nondeterministic Turing machines, where $L(n) \geq \log n$, is closed under complement.*

Since the family of context-sensitive languages is equal to the family of languages recognized by $L(n)$ tape-bounded nondeterministic Turing machines for $L(n) = n$ (see [1], [4]), the following special case is obtained:

Corollary 2. *The family of context-sensitive languages is closed under complement.*

Acknowledgement. The author of this paper would like to express his gratitude to Branislav Rovan for his kind help and advice during the preparation of this paper.

References

1. Hopcroft, J.E., Ullman, J.D.: Formal languages and their relation to automata. Reading: Addison-Wesley 1969
2. Immerman, N.: Nondeterministic space is closed under complementation. Proceedings of the 3rd Annual Conference Structure in Complexity Theory, 14–17 June 1988, Washington D.C. (also TH Report YALEU/DCS/TR 552, July 1987)
3. Immerman, N.: Descriptive and computational complexity. (unpublished manuscript, 1987)
4. Kuroda, S.Y.: Classes of languages and linear-bounded automata. Inf. Control 7, 207–233 (1964)
5. Szelepcsényi, R.: Context-sensitive languages are closed under complementation, TR Komensky University, April 1987 (in Slovak)

6. Szelepcsényi, R.: The method of Forcing for nondeterministic automata. Bull. Eur. Assoc. Theor. Comp. Sci. **33**, 96–100 (1987)

Received November 10, 1987/July 6, 1988

Appendix

A more detailed informal description of the work of T' is presented here using common programming language notation.

```
begin
index:=0; max:=init; card:=1;
max1:=init; card1:=1;
repeat
    newconf:=false;
    checksum:=0;
    for conf0:=first to max do
        begin
        sim:=init; countstep:=0; step:=guessstep;
        while (countstep<index) and (step>0) do
            begin
            execute(sim,step); countstep:=countstep+1;
            step:=guessstep;
            end;
        if conf0=sim then
            begin
            checksum:=checksum+1;
            for step:=1 to m do
                begin
                conf:=conf0; execute(conf,step);
                occurs:=false;
                checksum1:=0
                for conf1:=first to max do
                    begin
                    sim:=init; countstep:=0; step1:=guessstep;
                    while (step1>0)and(countstep<index) do
                        begin
                        execute(sim,step1); countstep:=countstep+1;
                        step1:=guessstep
                        end;
                    if conf1=sim then
                        begin
                        checksum1:=checksum1+1;
                        if conf=conf1 then occurs:=true;
                        if conf1<conf0 then
                            for step1:=1 to m do
                                begin
                                conf2:=conf1; execute(conf2,step1);
                                if conf=conf2 then occurs:=true
                                end
                        end
                    end;
                if checksum1<card then reject;
                if not occurs then
```

```
                    begin
                    newconf:=true;
                    if conf>max1 then max1:=conf; card1:=card1+1
                    end
                end
            end
        end;
    if checksum<card then reject;
    index:=index+1; max:=max1; card:=card1
until not newconf;
occurs:=false;
checksum:=0;
for conf0:=first to max do
    begin
    sim:=init; step:=guessstep;
    while step>0 do
        begin
        execute(sim,step);
        step:=guessstep
    end;
    if conf0=sim then
        begin
        checksum:=checksum+1;
        if accepting(conf0) then occurs:=true
        end
    end;
if checksum<card then reject;
if occurs then reject else accept
end.
```

The meaning of the symbols used is following:

1. Constants:
   - **first** – denotes the first configuration of T in the chosen ordering
   - **init** – denotes the initial configuration of T
   - **m** – denotes the maximum number of applicable steps
2. Variables:
   - **max, max1, conf0, conf, conf1, conf2, sim**
     – represent one of the configurations from the range **first**... max S
   - **index, card, card1, checksum, checksum1, counstep**
     – represent integer numbers from the range 0... card S
   - **step, step1**
     – represent integer numbers from the range 0... **m**
   - **newconf, occurs**
     – boolean variables
3. Functions:
   - **guessstep**
     – a *nondeterministic* function which provides a number from 1 to **m** of a selectable step which is to be executed during the simulation (or 0 to signal the end of the simulation)
   - **accepting(conf)**
     – boolean function which gives the value **true** if **conf** is an accepting configuration
4. Procedures:
   - **execute(conf,i)**
     – executes the **i**-th applicable step on the configuration **conf**
   - **reject** – rejection of the input word

accept – accepting of the input word

A brief comment to the program:

In lines 1–53 the values max S and card S are evaluated. The evaluation of max S_{i+1} and card S_{i+1} from the values max S_i and card S_i is carried out in lines 5–51. The variable **conf0** runs through all configurations. In lines 8–13 a number of steps of T is *nondeterministically* simulated to test whether **conf0** belongs to S_i. If **conf0** belongs to S_i, all configurations reachable from **conf0** in one step of computation are enumerated in **conf** in line 19. Before the inclusion into S_{i+1} every configuration is tested in lines 20–42 whether it has already been included into S_{i+1}. The variable **conf1** again runs through all configurations. The simulation of T is carried out in lines 24–29. In line 33 **conf** is compared with every configuration from S_i. If **conf1** is in the chosen ordering before **conf0**, then in lines 35–39 **conf** is compared with all configurations that are reachable in one step of computation. If **conf** has not been included into S_{i+1}, the variables **newconf**, **max1** and **card1** are updated in lines 44–47.

In lines 54–71 the set S is searched for an accepting configuration. The simulation for testing whether **conf0** belongs to S is carried out in lines 58–63. Every configuration is tested in line 64. If no accepting configuration has been found, the input word is accepted (line 71).