SOME RELATIONS BETWEEN MARKOV ALGORITHMS AND FORMAL LANGUAGES

G. LEONI (1) - R. SPRUNGOI (2)

ABSTRACT - Markov algorithms have received very little attention in the studies about formal languages, so the purpose of the present paper is twofold: i) to characterize languages in terms of Markov algorithms, and ii) to produce automatically Markov algorithms accepting or parsing languages generated by given grammars.

We use a particular, although universal, subclass of Markov algorithms, which we call «pointer Markov algorithms»; we obtain a characterization of: i) regular, ii) deterministic context-free, and iii) type 0 languages, which is quite «natural» in terms of these algorithms. Furthermore, we show that, given a right linear or a strong $LL(k)$ grammar, it is possible to produce automatically a pointer Markov algorithm parsing the language generated by the grammar. These constructions are particularly interesting because pointer Markov algorithms can be compiled conveniently into machine code programs.

0. Introduction,

The concept of a Normal Markov Algorithm (NMA) [1] is well known in mathematical logic; in the computer field it was used in the early studies on the semantics of programming languages (see [2] for a review), but it was soon abandoned; other applications are occasional (see, e. g., [3]). However, in our opinion, NMA’s can be a useful tool in theoretical as well as in practical applications.

We think that there are two main reasons why MNA’s did not meet a general agreement:

1) their formulation, which, for example, makes it difficult to specialize subclasses of NMA’s performing particular tasks;

— Received July 19, 1976.
(1) Istituto di Scienze dell’Informazione - Università di Pisa.
(2) Istituto di Elaborazione dell’Informazione del C.N.R - Pisa.
b) their execution, which is intrinsically slow, so that they cannot be considered for actual realizations.

This latter point has been treated in [4], where it is shown how the execution time of an NMA can be reduced by a factor proportional to the length of the argument (a string of characters, from now on: the object string). A similar and stronger result is in [5], where the use of « pointers » in NMA’s is discussed.

The concept of a Pointer Markov Algorithm (PMA) was introduced formerly in [6], as a more convenient, although equivalent, formulation of NMA’s; for example, it allows us to avoid concluding rules. An analogous concept, but not so easy to be handled, can be found in [7].

The rules of a PMA have the form « \( w_1 \phi w_2 \rightarrow y_1 \theta y_2 \) », where \( w_1, w_2, y_1, y_2 \) are (possibly empty) strings of characters on the alphabet of the algorithm, and \( \phi, \theta \) are special characters, called « pointers »; their role is explained in section 1; for the moment, we use this form of the rules to present our results; in fact, PMA’s are used in this paper to characterize and parse formal languages.

As for the characterization of languages, is concerned we show that there are three natural classes of languages related to PMA’s:

1) regular languages, accepted by PMA’s, in every rule of which \( w_1 \) and \( y_1 \) are empty (dually, \( w_2 \) and \( y_2 \) are empty);

2) deterministic (context-free) languages, accepted by PMA’s, in every rule of which \( y_2 \) is not longer than \( w_2 \) (dually, \( y_1 \) is not longer than \( w_1 \));

3) type 0 languages, accepted by the most general type of PMA’s. This last result follows immediately from the equivalence between PMA’s and NMA’s, so that we shall be concerned mainly with 1) and 2), to which are dedicated sections 3 and 5, respectively.

For what concerns the parsing of languages we show how:

1) given any right linear grammar it is possible to produce automatically a PMA (with a minimal number of rules) accepting the regular language generated by the grammar;

2) given any strong \( LL(k) \) grammar it is possible to produce automatically a PMA parsing the deterministic language generated by the grammar. Sections 4. and 6. are dedicated respectively to these topics.

The importance of these constructions arises from the fact that PMA’s can be compiled into machine code, so that, in the present case, we can produce fast routines to recognize regular languages and parse languages generated by strong \( LL(k) \) grammars. The characteristics of our implementation of PMA’s can be found in [8], [10].