5-1970

Studies in Computational Linguistics No. 1, The Recognition of Alphabets

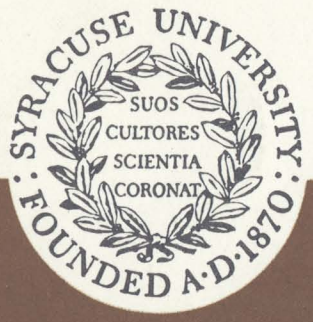
Edward F. Storm
Syracuse University

70-1

Studies in Computational Linguistics

No. 1

The Recognition of Alphabets



SYSTEMS AND INFORMATION SCIENCE
SYRACUSE UNIVERSITY

Studies in Computational Linguistics

No. 1

The Recognition of Alphabets

Edward F. Storm
Systems and Information Science
Syracuse University
250 Machinery Hall
Syracuse, New York 13210

May, 1970

ABSTRACT

Formal parsing rules for programming languages often have machinery for recognizing identifiers, numerical constants, and other substrings whose internal structure is only marginally relevant to the language structure as a whole. In this note alphabets are introduced in which identifiers, constants, etc. are regarded as single symbols. An alphabet is thus constructed out of a finite set of characters, is identified as a regular language and a simple recognition algorithm is described, giving the language designer considerable latitude in his choice of alphabet.

I. Introduction

A formal language is ordinarily determined by giving an alphabet and a set of well-formation rules which determine whether or not an object constructed out of instances of members of the alphabet belongs to the formal language. Individual elements in an alphabet are called characters, and the essential properties of an alphabet are these:

1. Each character must admit an arbitrary number of instances, or copies, as needed.
2. An instance of a character is always recognizable as such.
3. Instances of different characters are distinguishable.

Alphabets are often finite, but it is a simple matter to provide an infinite alphabet from a given finite one by including a set of well-formation rules for the construction of symbols in the infinite alphabet out of characters in the finite alphabet. We distinguish symbols from characters, so that symbols are elements of an infinite alphabet, and are constructed out of characters, which are elements of a finite alphabet. In this note we introduce a systematic process for defining symbols in terms of characters.

If the formal language under discussion is a programming language then there are good reasons to regard that its well-formed constructions are built up from an infinite alphabet.

For example, there is not much to be gained from a syntactic parse of the internal structure of an identifier. The same can be said about integers, floating point numbers, quoted literal strings and many others. It is convenient, then to given an infinite alphabet as the basis from which to build the well-formed constructions in a programming language. And if this infinite alphabet is to contain classes of things like identifiers, integers, reals, literal strings, connectives, brackets, etc., it will also be convenient to partition the alphabet explicitly into these classes.

In this note a systematic convention is described which gives the programming language designer considerable latitude in the specification of an alphabet of symbols, and which allows him to determine a partition of this alphabet into a finite number of classes. Syntactic considerations which are in fact purely alphabetic are thus eliminated from the processing of well-formed language constructions, resulting in a meaningful simplification of syntactic processes.

For those who may wish to savor the concept without digesting technical details we observe that a type-3 language may be described as the union of some of the equivalence classes of a congruence relation of finite index[2]. One may think of the set of accepting states of a finite-state recognizer as determining a partition of the set of accepted strings. The substitution property guarantees, for example, that if θ_1 and θ_2 are both identifiers, then the result of

concatenating the same letters to the right or left of each preserves identifierhood. If θ_1 and θ_2 are both unsigned integers, then concatenating these to the left of any exponent part will always yield an unsigned number.

In short it is convenient to identify partitioned alphabets with regular languages.

II. Some Examples of Alphabetic Definition

For simplicity suppose we make the following BNF definitions:

<letter>::=A|B|C

<digit>::=0|1

The Algol-60 report[1] contains the following definition of the syntax class of identifiers:

<identifier> ::=<letter>|<identifier><letter>|<identifier><digit>.

A finite-state recognizer for identifiers may be defined as follows:

$$\langle \{S_0, S_1\},\ M,\ S_0,\ \{S_1\} \rangle$$

where M, the transition function, is as follows

Current State	Symbol	Next State
S_0	A	S_1
S_0	B	S_1
S_0	C	S_1
S_1	A	S_1
S_1	B	S_1
S_1	C	S_1
S_1	0	S_1
S_1	1	S_1

The representation of the above transition table may be simplified as follows:

Current State	Symbol	Next State
S_0	<letter>	S_1
S_1	<letter>	S_1
S_1	<digit>	S_1

The requirement justifying this notational simplification is that the definiens for the metalinguistic variable appearing in the Symbol column must consist entirely of characters.

For a more complex example, consider the Algol-60 definition of number:

<unsigned integer>::=<digit>|<unsigned integer><digit>
<integer>::=<unsigned integer>|+<unsigned integer>|
-<unsigned integer>
<decimal fraction>::=.<unsigned integer>
<exponent part>::=@<integer>
<decimal number>::=<unsigned integer>|<decimal fraction>|
<unsigned integer><decimal fraction>
<unsigned number>::=<decimal number>|<exponent part>|
<decimal number><exponent part>
<number>::=<unsigned number>|+<unsigned number>|
-<unsigned number>

Table 1 shows the transition function for a finite-state acceptor for the number class of strings. Q_1, T_2 and S_3 are accepting states. Q_1 is the accepting state for an integer (signed or unsigned). T_2 is the accepting state for a decimal number (signed or unsigned), and S_3 is the accepting state for a number containing an exponent part. Clearly the transition function could be organized along alternative lines to realize a variety of different partitions of the class of numbers.

Current State	Symbol	Next State
S_0	<digit>	Q_1
Q_1	<digit>	Q_1
Q_1	.	T_1
S_0	.	T_1
T_1	<digit>	T_2
T_2	<digit>	T_2
S_0	@	S_1
S_1	+	S_2
S_1	-	S_2
S_1	<digit>	S_3
S_2	<digit>	S_3
S_3	<digit>	S_3
T_2	@	S_1
S_0	+	R_1
S_0	-	R_1
R_1	<digit>	Q_1
R_1	@	S_1
R_1	.	T_1

Table 1

Transition Function for Number Class

III. Algorithms for Recognizing Alphabets

In this section we describe an algorithm which incorporates a finite-state acceptor in a recognition process. The process accepts one character at a time from a source, and contains a buffer register which is used to assemble a symbol for output. The transition function is specified by a series of entries of the form:

$$(S, \sigma) \rightarrow (S', W)$$

where S is the current state and σ the current input character, S' is the designated next state, and W is a string to be concatenated to the buffer. The string W may be empty, and it will ordinarily consist of σ alone, although it may be a complex character string containing one or more occurrences of σ. The interpretation is that when the machine finds itself in state S, scanning the character σ, it concatenates the string W to the buffer and goes into state S'. If the pair (S, σ) does not appear in the transition function then the contents of the buffer is transferred to the output, the state S is put out as an accepting state, and the pair (S_0, σ) is constituted, where S_0 is the designated initial state.

The set of accepting states constitutes an implementation of the partition of the alphabet. Thus, a syntax analyzer which is being supplied by the finite state acceptor will

know whether the symbol just provided is an identifier, an integer, a literal string or some other uniquely classified symbol.

It is the designer's responsibility to guarantee that the transition function and set of accepting states have the desired effect.

References

(1) Naur, P. (ed.), "Revised Report on the Algorithmic Language Algol 60," Comm. ACM, Vol 6, 1963; pp. 1-17.

(2) Rabin, M.O. and Scott, D., "Finite Automata and Their Decision Problems," IBM Journal of Res. and Dev., Vol. 3, 1959; pp. 114-125.