RAIRO
INFORMATIQUE THÉORIQUE

R. PINZANI
R. SPRUGNOLI

The separability of formal languages

RAIRO – Informatique théorique, tome 16, n° 1 (1982), p. 13-31.
<http://www.numdam.org/item?id=ITA_1982__16_1_13_0>

R.A.I.R.O. Informatique théorique/Theoretical Informatics
(vol. 16, n° 1, 1982, p. 13 à 31)

THE SEPARABILITY OF FORMAL LANGUAGES (*)

by R. PINZANI (1) and R. SPRUGNOLI (2)

Communicated by J. F. PERROT

Abstract. – *We introduce the concept of a separator, a concept weaker than a parser or a recognizer; however a separator can be used very conveniently when speed is important and a precise error recovery is not mandatory. We distinguish between internal and external separation criteria and show how a set of these criteria is forming a hierarchy of more and more complex separators.*

Résumé. – *On introduit le concept de séparateur, qui est plus faible que celui d'analyseur ou de reconnaisseur; un séparateur peut être très commode lorsqu'il s'agit d'aller vite et que l'on ne demande pas de diagnostics d'erreurs très précis. On distingue les critères de séparation internes et externes et on montre comment certains de ces critères forment une hiérarchie de séparateurs de plus en plus complexes.*

INTRODUCTION

Given a language L and its definition by means of a phrase structure grammar, a *parser* for L is a procedure which, for any word w, builds the syntactic tree for w if $w \in L$, and signals some error (or never stops) if $w \notin L$. A weaker concept than a parser is a *recognizer*, that is a procedure which answers "yes" if $w \in L$ and "no" otherwise (*see* [1]).

Most systems for syntax directed translation require parsers, but some systems may use recognizers especially when they perform transformations through string pattern matching (e. g. *see* [2, 3, 4, 5]). As a simple example, let us consider the FORTRAN statement:

$$DO\ 12I3 = 4.7; \tag{1}$$

(*) Received May 1980.
(1) G.N.I.M.-C.N.R., Istituto Matematico "U. Dini", Viale Morgagni 67/A, Firenze, Italy.
(2) I.E.I.-C.N.R., Via S. Maria 46, Pisa, Italy.

R.A.I.R.O. Informatique théorique/Theoretical Informatics, 0399-0540/1982/13/$ 5.00

a system of the first kind will build the subtree corresponding to this assignment statement; a system of the second kind will match the statement against some patterns, e.g.

$$DO \langle \text{label} \rangle \langle \text{identifier} \rangle = \langle \text{initial-value} \rangle , \langle \text{test-value} \rangle , \quad (2)$$

$$\langle \text{identifier} \rangle = \langle \text{expression} \rangle ; \quad (3)$$

when a successful match is found, some transformations will be performed, as the translation of the statement into machine code. For systems of this second kind, a concept weaker than a recognizer can be useful to improve its performance; intuitively, in the example above, without using any syntactic recognizer, we can remark that:

DO in (1) matches *DO* in (2);
12 in (1) is the only subword which can match ⟨ label ⟩ in (2);
*I*3 in (1) is the only subword which can match ⟨ identifier ⟩ in (2);
= in (1) matches "=" in (2);
4 in (1) is the only subword which can match ⟨ initial-value ⟩;
in (1) does not match "," in (2),

so that the entire matching process fails.

A *separator* for a language L is a procedure which finds out the only (or the longest, shortest) subword of a given word w, which starts at a predefined position in w and possibly belongs to L. Obviously, a parser or a recognizer are also separators; however, this last concept it useful if the procedure of separation for a language L (in a given pattern) is simpler and faster than the procedure of recognition of parsing. In general separators can be used only to show that a word does not match a given pattern, but there are cases in which separators can be used directly instead of recognizers; for example, sometimes programming languages have two compilers: a fast and inefficient compiler to set up programs, and a slow but efficient compiler to get optimized code of correct programs; this latter compiler can profitably use separators instead of recognizers, since programs are supposed to be syntactically correct. In the conclusions we shall give a list of applications in which the concept of a separator has been used very conveniently.

In this paper we present some topics about separators as they have been developed during the implementation of APS (Algorithmic Programming System); APS (*see* [5]) is a system based on string pattern matching, designed to define formally the operational semantics of programming languages and to solve problems of non-numerical nature. However, the results obtained are independent of any particular system; they have been applied also in other fields and rely only on the concept of string pattern matching.

1. PRELIMINARIES

The definitions, conventions and properties listed below will be used throughout the paper without any other reference.

A. Generalities

(1) $\underline{N}$ is the set of natural (non negative integer) numbers;

(2) $\underline{Z}$ is the set of integer numbers;

(3) an *alphabet* is any finite set A;

(4) any $a \in A$ is a *letter* or *character*;

(5) $(A^*, .)$ is the free monoid generated by A;

(6) the operation "." is called *concatenation* and is denoted by simple juxtaposition of its arguments;

(7) any $w \in A^*$ is a *word* or *string* on A;

(8) the identity in $(A^*, .)$ is the *empty word* denoted by $\underline{e}$;

(9) the number of elements in A is denoted by $\#(A)$.

B. Homomorphisms

(1) $(\underline{N}, +)$, $(\underline{Z}, +)$, $(A^*, .)$ are monoids;

(2) a (monoid) homomorphism is a function $f : M \to N$ (M, N two monoids with identities $\underline{e}_M$, $\underline{e}_N$) such that $f(xy) = f(x)f(y)$, $\forall x, y \in M$, and $f(\underline{e}_M) = \underline{e}_N$;

(3) the composition of two homomorphisms is a homomorphism;

(4) a homomorphism $f : A^* \to M$ is defined when the image $f(a)$, for every $a \in A$, is defined;

(5) the length $|w|$ of a word $w \in A^*$ is the image of w under the homomorphism of A^* onto $\underline{N}$ defined by $|a| = 1$, $\forall a \in A$;

(6) let $a \in A$; the homomorphism $f_a : A^* \to \{a\}^*$ is defined by $f_a(a) = a$, and $f_a(x) = \underline{e}$, $\forall x \in A$ and $x \neq a$;

(7) $\#_a(w) = |f_a(w)|$ is a homomorphism of A^* onto $\underline{N}$;

C. Languages

(1) a *language* on A is any subset $L \subseteq A^*$;

(2) if $\mathbb{L}$ is a set of languages on A, and B a possibly infinite set of symbols, a *naming* for $\mathbb{L}$ is any function $u : B \to \mathbb{L}$;

(3) if $K \in B$ and $u(K) = L \in \mathbb{L}$, then K is a *name* of L;

(4) in general, we shall not distinguish between a language and any one of its names;

(5) if $a \in A$, "a" will be considered conventionally as a name of the language $\{a\}$;

(6) $E = \{\underline{e}\}$;

(7) a language L is $\underline{e}$-free iff $\underline{e} \notin L$;

(8) a language L is *prefix-free* iff for every words $x, y \in L$, and every word $w \in A^*$, we have $xw \neq y$.

D. Operations (L, M any two languages)

(1) $L.M = LM = \{xy \mid x \in L \wedge y \in M\}$ (*product*);

(2) the product of languages is associative, not commutative, distributive with respect to union and intersection;

(3) product will have precedence over union and intersection;

(4) $\emptyset L = L\emptyset = \emptyset$;

(5) $EL = LE = L$;

(6) $L^0 = E$;

(7) $L^n = LL \ldots L$ (n times), that is $L^n = LL^{n-1}$;

(8) $L^* = \bigcup_{n=0}^{\infty} L^n$ (*star* operation);

(9) $L^+ = \bigcup_{n=1}^{\infty} L^n$

E. Word relations and languages (x, y, z any words on A)

(1) $y \underline{H} x$ iff $\exists z \in A^*$, $x = yz$ (y is a *head* or *prefix* of x);

(2) $y \underline{T} x$ iff $\exists z \in A^*$, $x = zy$ (y is a *tail* or *suffix* of x);

(3) $y \underline{W} x$ iff $\exists z_1, z_2 \in A^*$, $x = z_1 y z_2$ (y is a *subword* of x);

(4) $y H x$, $y T x$, $y W x$ iff $y \underline{H} x$, $y \underline{T} x$, $y \underline{W} x$, respectively, and $x \neq y$;

(5) $H(L; m) = \{w \in A^* \mid |w| = m \wedge (\exists x \in L, w \underline{H} x)\}$;

(6) $T(L; m) = \{w \in A^* \mid |w| = m \wedge (\exists x \in L, w \underline{T} x)\}$;

(7) $W(L; m) = \{w \in A^* \mid |w| = m \wedge (\exists x \in L, w \underline{W} x)\}$;

(8) $H(L) = H(L; 1)$ the leftmost characters in L;

(9) $T(L) = T(L; 1)$ the rightmost characters in L;

(10) $W(L) = W(L; 1)$ the characters in L;

(11) $H(L; 0) = T(L; 0) = W(L; 0) = E$;

(12) $W'(L; m) = W(L'; m)$ where $L' = \{w \in A^* \mid \exists x \in L, w H x\}$.

F. Special languages

(1) $T=\{a^n b^n c^n \mid n \geqq 1\}$;
(2) $D=ab \mid aDb \mid DD$ (context-free-like definition);
(3) $Q=ab \mid aDb$ (context-free-like definition);
(4) for every m, $n \in \underline{N}$, m, n not both zero, there are defined:
(5) $L(m; n)=(z^+ a_1 \ldots a_{m+n-1} z^+ a_2 \ldots a_{m+n})^* z^+ a_1 \ldots a_m$;
(6) $M(m; n)=a_{m+1} \ldots a_{m+n} z^+$;
(7) $N(m; 1)=z^+ a_2 \ldots a_{m+n} z^+ a_1 \ldots a_m$;
(8) $N(m; n)=a_{m+1} \ldots a_{m+n-1} z^+ a_2 \ldots a_{m+n} z^+ a_1 \ldots a_m$ $(n>1)$.

2. STRING PATTERN MATCHING

Let $\mathbb{L}$ be a set of languages over an alphabet A, and B a set of names for the languages in $\mathbb{L}$:

2.1. DEFINITION: A *subscripted language name* or *class symbol* is any element $(L, n) \in B'=B \times \underline{N}$; the couple (L, n) will be written L_n or simply L whenever $n=0$. Any element in $A \cup B'$ will be called a *symbol* and a (*string*) *pattern* is any finite sequence of symbols.

In the sequel, the term *class* will be used to mean either a language or a language name, according to convention C4.

2.2. DEFINITION: Let $v=v_1 v_2 \ldots v_k$ be a pattern (v_i a symbol, $\forall i=1, 2, \ldots, k$) and let w be a word on A; then w *matches* v iff there exists a decomposition $w=w_0 w_1 \ldots w_k w_{k+1}$ ($w_i \underline{W} w$, $\forall i=0, 1, \ldots, k+1$) such that:
(S1) if $v_i=v_j$ then $w_i=w_j$, $\forall i, j=1, 2, \ldots, k$;
(S2) if $v_i \in A$ then $w_i=v_i$, $\forall i=1, 2, \ldots, k$;
(S3) if $v_i=L_n$ then $w_i \in u(L)$, $\forall i=1, 2, \ldots, k$.

We remark that the matching is of some subwords $w_1, w_2, \ldots, w_k$ of w against the pattern v; however, to every pattern v it can be associated the languages of all the words w matching v in the restricted sense that the whole word w matches v; this is the *language defined* by the pattern (*see* [7]). The following theorem lists some simple consequences of the definition.

2.3. THEOREM: *Let v be a pattern and w a word on A; then:*
(*a*) *characters of A in the pattern match equal characters in the word;*
(*b*) *if two class symbols are equal both as names of languages and as subscripts, then they match equal subwords of w;*

(c) *if two class symbols are different either as names of languages and/or as subscripts, then they can match either equal or different subwords of w.*

Proof: Obvious from definition 2.1.

As an example, let "x" be a name of the language of the (finite representations of the) real numbers; then the pattern "$x+x$" matches the word "12+12", but not the word "11+3", while the two words are both matched by the pattern "x_1+x_2"; this pattern, however, can match a word, e. g. "12+12", in several ways, as "12+12", "2+12", "12+1", "2+1"; we can define an order in the set of all the possible decompositions of a word:

2.4. DEFINITION: Let $w=w_0 w_1 \ldots w_k w_{k+1}$ and $w'_0 w'_1 \ldots w'_k w'_{k+1}=w$ be two decompositions of the word w into $k+2$ subwords; the first decomposition *precedes* the second one iff:

(P1) there exists $0 \leqq i<k$ such that $w_j=w'_j$, $\forall j<i$, and $|w_i|<|w'_i|$ or

(P2) if $w_i=w'_i$, $\forall i=0, 1, \ldots, k-1$, then $|w_k|>|w'_k|$.

Furthermore, if $v=v_1 v_2 \ldots v_k$ is any pattern, the *canonical v-decomposition* of a word w is the first decomposition of w into $k+2$ subwords [in the order defined by (P1) and (P2)] satisfying the conditions (S1), (S2), (S3).

Continuing the example above, the canonical v-decomposition of the word "12+12" according to the pattern "x_1+x_2" is "12+12" (with "12" matching both x_1 and x_2); actually, we scan the word w from left to right looking for the longest subword of w matching the pattern at a minimum cost, that is taking the first k subwords as short as possible.

2.5. THEOREM: *There exist* $(n+k+1)!/n!(k+1)!$ *decompositions of a word w with* $|w|=n$, *into* $k+2$ *subwords; furthermore, if v is a pattern, the languages of which are all recursive, then there exists a recursive procedure which finds out the canonical v-decomposition of any word w.*

Proof: Obvious with classical arguments.

The problem of finding the canonical v-decomposition of a word w, according to some pattern v, has a combinatorial nature and the solution given by the second part of the theorem 2.5 has only a theoretical interest; in [6] we give a procedure which improves greatly this solution and the present paper is devoted to the theory underlying that procedure as well as to further improvements thereof.

Obviously, we can suppose that a pattern v is preceded by a symbol q not belonging to B, which is a name of the (regular) language A^*; thus, without any loss of generality, we can look for a matching of a prefix w' of a word w against the pattern qv. Because of that, if $v=v_0 v_1 \ldots v_k$ is a pattern and w is any word, the

problem of the (string) pattern matching can be stated in a recursive way: we look at some prefix w'_0 of w matching v_0; then, if we have $w = w'_0 w''$, we look at some prefix w'_1 of w'' matching v_1 and so on, trying to satisfy the conditions (S1), (S2), (S3) and (P1), (P2). At any moment in the process of pattern matching we have a word w and a pattern $v' = v_i v''$, where v'' is a suffix of the original pattern v; if $v_i \in A$, the first character in w must be v_i, by (S2); if $v_i = L_n$ and the couple L_n appeared in the prefix of v already considered, then, by (S1), the prefix of w must equal the word which matched L_n. The interesting case is $v_i = L_n (L \in B)$ and L_n not considered so far; our attention will be confined to this case, so we shall consider patterns of the form $v = L_n v'$ and look for prefixes of words w which possibly belong to the language $u(L)$.

2.6. DEFINITION: Let $v = L_n v'$ be a pattern; a *separation criterion* for L (relative to v) is any rule which will find out, given any word $w \in A^*$ and any prefix w' of w such that $w = w' w''$, that w'' cannot match v'. A *separator* is any procedure realizing a given separation criterion.

This definition reflects the negative nature of a separation criterion, that is its ability to tell that a pattern does not match a given string; as remarked before, a separator is a too weak procedure for deciding that a pattern does match a string. On the other hand, we can use separation criteria to find the shortest (or the longest or even the only) prefix of a word w which possibly belongs to L and allows a matching of the rest of the word with the rest of the pattern.

We shall distinguish between two classes of separation criteria: internal and external, according to the fact that they depend only on the structure of the language L or also on the rest of the pattern.

2.7. DEFINITION: A language L has an *internal* separation criterion (ISC) iff the following condition holds: let $v = L_n v''$ be any pattern and let w be any word matching v according to the canonical v-decomposition $w = w_1 w_2 w_3$ ($w_1 \in L$, w_2 matching v''); then if $w' = w_1 w''$ is any word matching the pattern $v' = L_k \hat{v}$, the canonical v'-decomposition of w' is $w' = w_1 w'_2 w'_3$. A separation criterion which is not internal will be called *external.*

The set $\underline{S}$ of separation criteria can be partially ordered:

2.8. DEFINITION: If C_1 and C_2 are two separation criteria, then C_1 is *weaker* then C_2 ($C_1 \leqq C_2$) iff whenever a language L can be separated by C_1 then it can be also separated by C_2; C_1 is *properly weaker* than C_2 ($C_1 < C_2$) iff $C_1 \leqq C_2$ and there exist a language L and a pattern v such that L is separable in v by C_2 but not by C_1.

3. INTERNAL SEPARATION

We can think of an ISC as a process which is going on until the end of the prefix of the word w corresponding to the language L has been found.

3.1. THEOREM: *Let L be an $\underline{e}$-free language; then L has an ISC if and only if L is prefix-free.*

Proof: If L is not prefix-free, there exist $x, y \in L$ such that $x = yz$, for some $z \in A^*$; if we consider the pattern $v = L$, the canonical v-decomposition of x is $w_1 w_2$, where $w_1 = x$ and $w_2 = \underline{e}$; on the other hand, if we consider the pattern $v' = L\ z$, in the canonical v'-decomposition of x, L is matched by y, which contradicts the definition 2.7. Now, let us suppose that there exists no ISC for L; by definition 2.7 there exist a pattern $v = L_n v''$ and a word w, the canonical v-decomposition of which is $w = w_1 w_2 w_3$ and $w_1 \in L$; moreover, there exist a pattern $v' = L_k \hat{v}$ and a word $w' = w_1 w''$, the canonical v'-decomposition of which is $w' = w'_1 w'_2 w'_3$, where $w'_1 \in L$ but $w'_1 \neq w_1$; thus, we must have $w_1 H w'_1$ or $w'_1 H w_1$, contradicting the hypothesis on L.

This is an important characterization of the languages having an ISC; as possible examples we have the languages Q, T and $L(m; 0)$, $\forall m > 0$; on the contrary, the languages $L(m; n)$ for $n > 0$ do not have any ISC.

The theorem 3.1 suggests also a separator for all the recursive prefix-free languages L; in fact, we can successively analyze all the prefixes of a word w; the first prefix belonging to L is the only subword matching L. However, from our point of view, this procedure is completely useless, since it makes use of some syntactic recognizer for the language L instead of giving an alternative to it. Thus, we have to follow other ways for our research. According to our experience, there are three important classes of languages with an ISC:

(*a*) fixed length languages;

(*b*) right closed languages;

(*c*) parenthetized languages.

Fixed length languages are finite languages, the elements of which all have the same length; important examples are character languages (subsets of A, as letters, digits, operators and so on) and some codified information languages (pieces of information occupying one or several fixed length fields in a record). A separator for these languages can be implemented by simply counting the characters in the word w.

For what concerns right closed languages, let us begin by considering the languages $L(m; 0)$; an ISC for them is given by:

3.2. THEOREM: *For the language* $L = L(m; 0)$, $m \geqq 1$, *we have:*

$$(\star) \qquad T(L; m) \cap W'(L; m) = \emptyset.$$

Proof: We remember that $W'(L; m)$ is the set of the subwords of the elements in L, which have length m and are not suffixes of words in L; thus, condition $(\star)$ means that a word of length m, which is the suffix of some word in L, can never be found inside any word in L. In our case we have:

$$T(L(m; 0); m) = \{a_1 a_2 \ldots a_m\}$$

and by definition $a_1 a_2 \ldots a_m$ is never contained in any word in L.

Many constructs in the definition of programming languages satisfy the condition $(\star)$; for example, statements ending with a semicolon or labels ending with a colon; thus we have:

3.3. DEFINITION: A language L for which there exists an $m \geqq 1$ such that condition $(\star)$ holds is called a *right closed language*; the same condition is an ISC for L and will be denoted by $\underline{C}(m; 0)$.

We can derive a simple separator for any ISC $\underline{C}(m; 0)$:

3.4. PROCEDURE: Let us consider the finite set $T(L; m)$:

(*a*) let $\hat{w}$ be the prefix of w of length m;

(*b*) if $\hat{w} \in T(L; m)$ then the procedure is over;

(*c*) otherwise, drop the first character from w and go to step (*a*).

The ISC's $\underline{C}(m; 0)$ constitute a sort of hierarchy:

3.5. THEOREM: *For every* $m < m'$ *we have* $\underline{C}(m; 0) < \underline{C}(m'; 0)$.

Proof: Let us proceed by induction on m; first of all $\underline{C}(0; 0) < \underline{C}(1; 0)$ since $\underline{C}(0; 0)$ cannot be defined; then we have:

$$T(L; m+1) \cap W'(L; m+1)$$

$$\subseteqq W(L)\, T(L; m) \cap W(L)\, W'(L; m)$$

$$= W(L)(T(L; m) \cap W'(L; m)) = W(L)\emptyset = \emptyset,$$

so that $\underline{C}(m; 0) \leqq \underline{C}(m+1; 0)$ by the induction hypothesis; moreover, $L(m+1; 0)$ has $\underline{C}(m+1; 0)$ as ISC by the theorem 3.2, but:

$$T(L(m+1; 0); m) = \{a_2 a_3 \ldots a_{m+1}\}$$

is a non-suffix subword of the elements in L $(m+1; 0)$ so that $\underline{C}(m; 0)$ is properly weaker than $\underline{C}(m+1; 0)$.

In the next section we shall see that the ISC's $\underline{C}(m; 0)$ are a particular case of separation criteria $\underline{C}(m; n)$.

For what concerns parenthetized languages, let us first consider the language Q:

3.6. LEMMA: *Let $a, b \in A$ and let us define the function $F : A^* \to \underline{Z}$:*

$$F(w) = \#_a(w) - \#_b(w), \qquad \forall w \in A^*;$$

then F is a monoid homomorphism.

Proof: The proof is immediate.

Now let us consider the restriction of F to Q:

3.7. THEOREM: *Let us call F again the restriction $F|_Q$; then $\forall w \in Q$ we have $F(w) = 0$ and $F(x) > 0$, $\forall x \, H \, w$.*

Proof: The proof is obtained by induction on the definitions of D and Q, using lemma 3.6.

From this theorem, it follows that any procedure computing the function F constitutes a separator for Q; in fact, we can have a counter (initially set to zero) to which we add one whenever we find a "b"; when we reach zero, without having obtained any negative number and found any extraneous character, we have separated the only prefix (possibly) belonging to Q. The importance of the language Q stems from the following considerations:

3.8. DEFINITION: A homomorphism h has the *closure property* relative to a language L iff $h(a) \neq \underline{e}$, $\forall a \in T(L)$. A language L is a *parenthetized* language iff there exists a homomorphism $h : L \to Q$ with the closure property relative to L.

Now, we can find a separator for any parenthetized language:

3.9. THEOREM: *Let L be a parenthetized language and h the homomorphism $h : L \to Q$ of definition* 3.8*; then the following procedure is a separator for L (w any word on A):*

(a) set a counter to zero;

(b) let x be the homomorphic image of the first character in w;

(c) if x is empty go to step (b); else begin the scanning of x;

(d) if the first character in x is "a" add 1 *to the counter; if it is "b" subtract* 1 *from the counter;*

(*e*) *if the counter is zero and the scanning of x is over, then we have found the prefix of w with the required property;*

(*f*) *if the counter is less than or equal to zero, then the procedure is over with a negative answer;*

(*g*) *if the counter is greater than zero then:*

- *if the scanning of x is over, drop a character from w and go to* (*b*);
- *else, drop a character from x and go to step* (*d*).

Proof: Let w_1 be the separated head of w; we must show that if $w_2 H w$ and $w_2 \neq w_1$ then $w_2 \notin L$. If $w_2 H w_1$, by construction we have $F(h(w_2)) > 0$, so $w_2 \notin L$ by theorem 3.7; on the other hand, if $w_1 H w_2$, $w_2 \in L$, then $h(w_1) H h(w_2)$, because h has the closure property relative to L (the last character in w_2 is not mapped in $\underline{e}$); thus, $h(w_2)$ has a proper prefix w_1 such that $F(h(w_1)) = 0$, which contradicts the assertion of theorem 3.7.

There are many examples of parenthetized languages in every programming language; as a non-standard example we mention here procedure headings and procedure calls in, e. g., Algol 60.

The three classes above do not exhaust the set of prefix-free languages, that is the set of languages with some ISC; it is a simple exercise to show that, although the language T is a parenthetized language, the language $T_1 = \{ a^n b^n a^n \mid n \geqq 1 \}$ and the language of the prefix Polish expressions on a set of operators $\Omega \subseteqq A$ and a set of symbols $X \subseteqq A$, $X \cap \Omega = \emptyset$ (with at least one operator in Ω with arity greater than 1) do not fall in any one of the three classes above. However, following [8] we define:

3.10. DEFINITION: A *deterministic push-down transducer with final states* (abbreviated ptf) is an 8-tuple $M = (A, B, C, S, F, s_0, z_0, u)$ where:

A is our basic *input* alphabet;

B is the *output* alphabet;

C is the alphabet of *push-down characters;*

S is the set of *states*;

$F \subseteqq S$ is the set of *final* states;

$s_0 \in S$ is the *start* state;

$z_0 \in C$ is the *start* push-down character;

$u : S \times (A \cup E) \times C \to S \times C^* \times B^*$ is the *transition* function.

If $u(s, a, c) = (t, w, y)$ then at configuration (s, ax_1, cx_2) with $x_1 \in A$, $x_2 \in C$, the ptf goes to the state t, writes w on the push-down tape and emit the word y as output.

A detailed description of how a ptf operates can be deduced by [8]; we are only interested in ptf's which translate a language L into Q, so that $B = \{ a, b \}$.

3.11. DEFINITION: A ptf M has the *closure property* relative to a language L iff there is no $u(s, a, c)=(t, w, y)$ such that $a \in T(L)$, $t \in F$ and $y=\underline{e}$.

This property allows us to give the following:

3.12. THEOREM: *Let M be a ptf which translates a language L into Q; if M has the closure property relative to L then a separator for L is given by the procedure of the theorem* 3.9 *in which step* (e) *has been changed to:*

(e') *if the counter is zero, the scanning of x is over and the ptf M has reached a final state, then we have found the prefix of w with the required property.*

Proof: This is analogous to the proof of theorem 3.9.

We remark the following fact: a ptf can be considered as a push-down transducer with a push-down acceptor associated to it, as they are defined in [8]; thus, it may be argued that if L is a CFL (context-free language) the ptf translating L would be a syntactic recognizer for L; this may be not necessarily true, since the ptf (or the pda which can be defined from it) does not, in general, recognize L, but some CFL containing L; this should be clear also from what follows.

The approach to ISC's by means of ptf's is rather general:

3.13. THEOREM: *If L is a fixed length language, or a right closed language or a parenthetized language, then there exists a ptf M translating L into Q and having the closure property relative to L.*

Proof: For a fixed length language L such that $|w|=n, \forall w \in L$, the ptf outputs an "a" at the beginning, passes through n different states and finally outputs a "b" passing to a final state (the associated pda recognizes A^n). For right closed languages, the ptf outputs an "a" at the beginning, starts a recognizing procedure for the tails (which are in a finite number) and finally outputs a "b" when a tail has been found. For parenthetized languages, we remark that a homomorphism with the closure property relative to L is a particular ptf with the closure property relative to L such that $Z=\{z_0\}$, $S=F=\{s_0\}$ and we write $h(a)=y$ instead of $u(s_0, a, z_0)=(s_0, z_0, y)$.

A simple exercise is the construction of ptf's with the required properties and relative to T_1 and to the language of the prefix Polish expressions, as quoted before.

4. EXTERNAL SEPARATION

For the moment, let us consider only $\underline{e}$-free languages.

If we have a pattern $\underline{v} = T f v'$, where $T = \{ a^n b^n c^n \mid n \geqq 1 \}$ and $f \in A$, a simple way of separating the prefix w_1 of any word w which possibly belongs to T is to find the first character f in w.

This procedure may be much faster than any ISC for T, though it can give less information.

4.1. DEFINITION: Let $v = L_p M_q v'$ be a pattern; L is separable in v by the $\underline{C}(0; 0)$ external separation criterion (ESC) iff:

$$W(L) \cap W(M) = \emptyset.$$

A separator which realizes this condition will simply scan the word w in order to find a character not belonging to $W(L)$; if it belongs to $W(M)$ the separation is successfull, otherwise there is no prefix of w which possibly belongs to L and allows the rest of the word to match the rest of the pattern.

In reality, we can require that $W(L)$ be disjoint from the set of leftmost characters of M, so that we are led to the following:

4.2. DEFINITION: Let $v = L_p M_q v'$ be a pattern; L is separable in v by the ESC $\underline{C}(m; n)$, $m \geqq 0$, $n \geqq 1$, iff:

$$T(L; m) H(M; n) \cap W(L; m+n) = \emptyset.$$

If $m = 0$ and $n = 1$, we have $T(L; 0) = E$ so that the ESC $\underline{C}(0; 1)$ can be given by the relation:

$$H(M) \cap W(L) = \emptyset;$$

in the example above, T is separable in $\underline{v}$ by both criteria $\underline{C}(0; 0)$ and $\underline{C}(0; 1)$.

4.3. THEOREM: *If L is separable in $v = L_p M_q v'$ by the ESC $\underline{C}(m; n)$, $m \geqq 0$, $n \geqq 1$, then the following procedure constitutes a separator for L (w being any word):*

(a) let us call y the prefix of w of length $m+n$;

(b) if $y \in W(L; m+n)$ then drop the first character from w and go to step (a);

(c) otherwise the prefix of w, we are looking for, is formed up by the scanned characters and the first m characters in y.

Proof: by the relation in definition 4.2.

Up to now, we have introduced criteria $\underline{C}(n; 0)$, $\underline{C}(0; 0)$, and $\underline{C}(m; n)$, $m \geqq 0$, $n \geqq 1$; they are related among them to form a peculiar structure in the partially ordered set $\underline{S}$:

4.4. LEMMA: *Let L be any language, then $\forall m \geqq 0$ we have:*

$$T(L; m+1) \subseteqq W(L)\, T(L; m),$$
$$H(L; m+1) \subseteqq H(L; m)\, W(L),$$
$$W(L; m+1) \subseteqq W(L)\, W(L; m),$$
$$W(L; m+1) \subseteqq W(L; m)\, W(L).$$

Proof: Let us consider the first relation; *a* suffix $x_1 x_2 \ldots x_m x_{m+1}$ can be thought of as the concatenation of $x_1 \in W(L)$ and a suffix $x_2 \ldots x_m x_{m+1}$ of length m; however, equality may not hold, because some characters may or may not occur in a specific position. The other relations are proved analogously.

This helps us to prove:

4.5. THEOREM: *The following relations hold $\forall m \geqq 0$, $\forall n \geqq 1$:*

$$(\star) \qquad \underline{C}(m; n) < \underline{C}(m+1, n),$$

$$(\star\star) \qquad \underline{C}(m; n) < \underline{C}(m; n+1).$$

Proof: Let us show that $\underline{C}(m; n) \leqq \underline{C}(m+1; n)$; if:

$$T(L; m)\, H(M; n) \cap W(L; m+n) = \emptyset,$$

then by lemma 4.4 we have:

$$T(L; m+1)\, H(M; n) \cap W(L; m+n+1)$$
$$\subseteqq W(L)\, T(L; m)\, H(M; n) \cap W(L)\, W(L; m+n)$$
$$= W(L)\,(T(L; m)\, H(M; n) \cap W(L; m+n)) = W(L)\,\emptyset = \emptyset.$$

Analogously, we can prove that $\underline{C}(m; n) \leqq \underline{C}(m; n+1)$:

$$T(L; m)\, H(M; n+1) \cap W(L; m+n+1)$$
$$\subseteqq T(L; m)\, H(M; n)\, W(M) \cap W(L; m+n)\, W(L)$$
$$\subseteqq T(L; m)\, H(M; n)\, W(L \cup M) \cap W(L; m+n)\, W(L \cup M)$$
$$= (T(L; m)\, H(M; n) \cap W(L; m+n))\, W(L \cup M) = \emptyset\, W(L \cup M) = \emptyset.$$

Now to complete our proof, let us show specific examples in which the relations $(\star)$ and $(\star\star)$ hold properly. In the pattern:

$$v = L(m+1; n)\, M(m+1; n),$$

$L(m+1; n)$ is separable by $\underline{C}(m+1; n)$ because:

$$T(L(m+1; n); m+1)\,H(M(m+1; n); n) = \{a_1 \ldots a_{m+1}\, a_{m+2} \ldots a_{m+n+1}\}$$

and this word is not a subword of any element in $L(m+1; n)$; on the contrary $L(m+1; n)$ is not separable by $\underline{C}(m; n)$ because:

$$T(L(m+1; n); m)\,H(M(m+1; n); n) = \{a_2 \ldots a_{m+1}\, a_{m+2} \ldots a_{m+n+1}\},$$

which is a subword of the elements in $L(m+1; n)$.

In a similar way we can show that in the pattern:

$$v = L(m; n+1)\,M(m; n+1),$$

$L(m; n+1)$ is separable by $\underline{C}(m; n+1)$ but not by $\underline{C}(m; n)$.

Now let us consider the ISC's $\underline{C}(m; 0)$:

4.6. THEOREM: *For every $m>0$ and $n>0$ we have:*

$$\underline{C}(m; 0) < \underline{C}(m; n).$$

Proof: It is sufficient to show that $\underline{C}(m; 0) < \underline{C}(m; 1)$. Since:

$$W(L; m+1) \subseteqq W'(L; m)\,T(L) \subseteqq W'(L; m)\,W(L),$$

if we suppose that $\underline{C}(m; 0)$ holds, we have:

$$T(L; m)\,H(M) \cap W(L; m+1) \subseteqq T(L; m)\,W(M) \cap W'(L; m)\,W(L)$$
$$\subseteqq (T(L; m) \cap W'(L; m))\,W(L \cup M) = \emptyset\, W(L \cup M) = \emptyset,$$

which proves $\underline{C}(m; 0) \subseteqq \underline{C}(m; 1)$. Now, considering the pattern:

$$v = L(m; 1)\,M(m; 1)$$

we can show that the relation holds properly.

Finally, for the criterion $\underline{C}(0; 0)$:

4.7. THEOREM: *The ESC $\underline{C}(0; 0)$ cannot be compared with $C(1; 0)$ but*

$$(\star) \qquad \underline{C}(0; 0) < \underline{C}(0; 1).$$

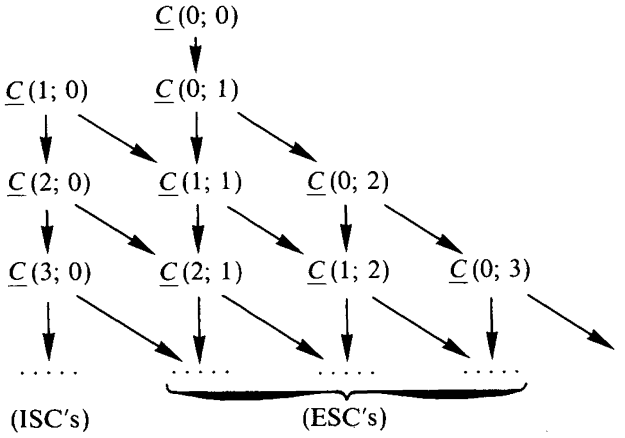
Proof: To show that $\underline{C}(0; 0) \subseteqq \underline{C}(0; 1)$ let us remark that:

$$H(M) \cap W(L) \subseteqq W(L) \cap W(M) = \emptyset;$$

then, if N is the language of numerals and I the language of identifiers, in the pattern $v=NI$, N is separable by $\underline{C}(0;\ 1)$, because an identifier must begin by a letter, but not by $\underline{C}(0;\ 0)$ since $W(I)\cap W(N)$ is the set of digits.

Now, if L is the set of the letters, in the pattern $v=NL$, N is separable by $\underline{C}(0;\ 0)$, but not by $\underline{C}(1;\ 0)$, because it has no ISC; on the contrary, in the pattern $v=L\ (1;\ 0)z$, $L\ (1;\ 0)$ has the ISC $\underline{C}(1;\ 0)$ but it is $W(L\ (1;\ 0))\cap W(z)=\{z\}\neq\emptyset$.

We can make a picture of the situation:



and show that there exist no relations among criteria being on the same line (i. e. $m+n=$constant).

4.8. THEOREM: *The criterion* $\underline{C}(m;\ n)$ *is not comparable with any* $\underline{C}(m+k;\ n-k)$, $k\neq 0$, *for which the criterion exists.*

Proof: The statement "the criterion exists" means $m+k\geqq 0$ and $n-k\geqq 0$; so, there exist the languages $L\ (m+k;\ n-k)$ and $M\ (m+k;\ n-k)$, also. However, it is simple to show that in the pattern $v=L\ (m;\ n)\ M\ (m;\ n)$, the language $L\ (m;\ n)$ is separable by $\underline{C}(m;\ n)$ but not by $\underline{C}(m+k;\ n-k)$ (*see* the proof of theorem 4.5), and in the pattern $v'=L\ (m+k;\ n-k)\ M\ (m+k;\ n-k)$ the language $L\ (m+k;\ n-k)$ is separable by $\underline{C}(m+k;\ n-k)$ but not by $\underline{C}(m;\ n)$.

Obviously, there exist patterns and languages which cannot be separated in these patterns by any separation criteria; for example, in the pattern $v=N_1 N_2$, where N is the language of numerals, N cannot be separated, because it has no ISC and $\forall\, m,\ n$:

$$T(N;\ m)\,H(N;\ n)\cap W(N;\ m+n)=W(N;\ m+n)\neq\emptyset.$$

In practice, such cases are very rare and some $\underline{C}(m; n)$ or ISC will always be successfull; however, the $\underline{C}(m; n)$ criteria require the construction of larger and larger tables as m and n grow; our experience has led us to consider only, for practical applications, $\underline{C}(1; 0)$ and $\underline{C}(1; 1)$.

In fact, by the theorem 4.7, $\underline{C}(0; 0)$ is properly contained in $\underline{C}(1; 0)$ and cannot be realized much more easily. $\underline{C}(0; 1)$ is applicable to languages, the elements of which are delimiters and with some definite set of characters; however, in general, it is more convenient to use explicitly such characters in the definition of a pattern; e. g., it is better to define the language S of Algol 60 statements without the ending semicolon and consider patterns of the form $v = v_1 S; v_2$.

The criteria $\underline{C}(1; 0)$ and $\underline{C}(1; 1)$ can be implemented easily; we remark that the ESC $C(1; 1)$ has to be used to separate the representation of real numbers within arithmetic expressions in programming languages as FORTRAN and Algol 60 (*see* [6]).

Till now, we have considered only $\underline{e}$-free languages [in reality, something more is required whenever we try to apply an ESC $\underline{C}(m; n)$]; now we wish to extend our results to languages, which possibly contain the empty word.

4.9. LEMMA: *If L and M are any two languages, then:*

(i) $H(L) = H(L \setminus E)$ $\quad (E = \{\underline{e}\})$;

(ii) $H(L \cup M) = H(L) \cup H(M)$;

(iii) $H(LM) = H(L)$ *if L is $\underline{e}$-free;*

(iv) $H(LM) = H(L) \cup H(M)$ *if* $\underline{e} \in L$.

Proof: Obvious.

For languages defined by patterns:

4.10. THEOREM: *Let* $L = L_{n_1}^{(1)} L_{n_2}^{(2)} \ldots L_{n_k}^{(k)}$ *and let h be the smallest index for which* $L_{n_h}^{(h)}$ *is $\underline{e}$-free, or $h = k$; then we have:*

$$H(L) = \bigcup_{i=1}^{h} H(L_{n_i}^{(i)}).$$

Proof: By induction on h, using lemma 4.9.

Now we can generalize the ESC $\underline{C}(m; n)$ to not $\underline{e}$-free languages:

4.11. DEFINITION: Let $v = L_n v_0'$ be a pattern and let us call v_0' also the language defined by the pattern v_0'; then L is separable in v by the $\underline{G}(m; n)$ external separation criterion iff:

$$T(L; m) H(v_0'; n) \cap W(L; m+n) \neq \emptyset.$$

For the ESC $\underline{G}(m; n)$ we must consider all the languages in the subpattern v'_0, but all the considerations made for $\underline{C}(m; n)$ also apply. The ESC's $\underline{G}(0; 1)$ and $\underline{G}(1; 1)$ worth a particular mention; let us suppose that $v = L_n v'_0 = L_n L^{(1)}_{n_1} L^{(2)}_{n_2} \ldots L^{(k)}_{n_k}$ and there exists a smallest index h for which $L^{(h)}_{n_h}$ is not $\underline{e}$-free.

4.12. THEOREM: *L is separable in v by the ESC $\underline{G}(0; 1)$ iff:*

$$W(L) \cap \bigcup_{i=1}^{h} H(L^{(i)}) = \emptyset;$$

L is separable in v by the ESC $\underline{G}(1; 1)$ iff:

$$T(L) \bigcup_{i=1}^{h} H(L^{(i)}) \cap W(L; 2) = \emptyset.$$

Proof: From the definition 4.11 and the theorem 4.10.

From these relations two separators can be easily designed; for what concerns our experience, they provide an almost general solution to the practical problem of external separation.

5. CONCLUSIONS

In this paper we presented the concept of a separator, a weaker form of a recognizer or a parser. We developed a theory of separators, distinguishing between internal and external separation criteria and showing how the set of separators is forming a hierarchical structure of more and more complex criteria.

As we mentioned in the introduction, separators were first used in the implementation of APS, a system for the formal definition of the operational semantics of programming languages and other non-numerical problems.

However, the concept of language separability was proved to be usefull in many other fields; we mentioned already the construction of optimized compilers for (supposedly) correct programs; the use of separators instead of parsers may improve considerably the speed of compilation.

Another area of application is "interpretation", in a wide sense of this word; whenever we are faced to an interpretive solution of a problem, the speed of interpretation is crucial and may overcome the need of a precise error recovery. For example, we mention interpretive programming languages and query processing in data base management systems (DBMS).

Also related to DBMS's are two other applications of separators:

(i) the input of records in the data base; the separation criteria are used to distinguish the various fields inside the record;

(ii) the use of quasi-natural languages in queries and data; in this case separators may be used to isolate terms and recognize some simple grammatical structures and syntactic constructs.

REFERENCES

1. A. V. AHO and J. D. ULLMAN, *The Theory of Parsing, Translation and Compiling*, Prentice-Hall, Englewood Cliffs, N.J., 1972.
2. R. E. GRISWOLD, J. F. POAGE and J. P. POLONSKY, *The SNOBOL4 Programming Language*, Prentice-Hall, Englewood Cliffs, N.J., 1971.
3. J. F. GIMPEL, *A Theory of Discrete Patterns and their Implementation in SNOBOL4*, Comm. ACM, Vol. 16, No. 2, 1973, pp. 91-100.
4. A. CARACCIOLO DI FORINO, *Generalized Markov Algorithms and Automata.* In *Automata Theory*, CAIANELLO, Ed., 1966, p. 115-130, Academic Press, New York.
5. G. AGUZZI, R. PINZANI and R. SPRUGNOLI, *An Algorithmic Approach to the Semantics of Programming Languages.* In *Automata, Languages and Programming* NIVAT, Ed., 1973, p. 147-166, North Holland Pub. Co., Amsterdam.
6. G. AGUZZI, F. CESARINI, R. PINZANI, G. SODA and R. SPRUGNOLI, *Towards an Automatic Generation of Interpreters*, Lecture Notes in Computer Science, No. 1, 1973, pp. 94-103, Springer Verlag, Berlin.
7. G. F. ROSE: *An Extension of Algol-Like Languages*, Comm. ACM, Vol. 7, No. 2, 1964, pp. 52-61.
8. S. GINSBURG: *The Mathematical Theory of Context-Free Languages*, McGraw Hill, New York, 1966.