An intuitive representation of context-free languages

By László KALMÁR in Szeged, Hungary

1. In this paper, the following conception of language is used. A language is an ordered triple $L = \langle V, C, f\rangle$ where V and C are two disjoint, non-empty, finite sets and f is an application of C into the set of all subsets of the free semigroup $F(V)$ generated by V. The set V is called the vocabulary (in Chomsky [1], terminal vocabulary), its elements are called words, those of $F(V)$ word strings. The elements of C (which corresponds to the auxiliary vocabulary in Chomsky [1]) are called (grammatical) categories. For any category $c \in C$, the elements of $f(c) \subseteq F(V)$ are called the word strings belonging to the category c.

The usual conception of language, viz. a subset S of $F(V)$, is a particular case in which $C = \{s\}$ contains a single element s (the category of sentences, a sentence, i. e. a word string belonging to the category s, being an element of S). However, both for natural and formal languages, the above, more general conception seems to be more appropriate, for we are not only interested, in the case of a natural language, in what are the sentences, and, for a programming language, say, what are the programs, but also, what are the noun phrases, verbal phrases, etc., and, what are the declarations, statements, expressions, etc., respectively. Another advantage of our more general conception is that for a generative grammar of L, we can use the set C of categories (but of course, we can use any superset of C as well) as auxiliary vocabulary.

Accordingly, we define a context-free grammar as an ordered triple $G = \langle V, C, R\rangle$ where V and C are two disjoint, non-empty, finite sets and R is a subset of the Cartesian product of C with the free semigroup $F(V \cup C)$ generated by the union of V and C. V and C are called the vocabulary and set of categories (or terminal and auxiliary vocabulary), respectively; the elements r of R, which are of the form $\langle c, \sigma\rangle$ with $c \in C$ and $\sigma \in F(V \cup C)$, are called (production) rules. A rule $\langle c, \sigma\rangle$ will be written in the sequel as "$c:\sigma$" as in the presentation of ALGOL 68 [2], rather than "$c \to \sigma$" as in Chomsky or "$c::=\sigma$"

as in the presentation of ALGOL 60 [3]). A "mixed string," i. e. an element σ of $F(V \cup C)$ is called a direct production of a category c if $c{:}\sigma$ is a rule; productions of a category c are defined recursively as (i) its direct productions and (ii) mixed strings $\sigma = \sigma_1\sigma_2\sigma_3$ formed of productions $\sigma' = \sigma_1 c' \sigma_3$ of c by replacing a category c' by a direct production σ_2 of c'. (We denote the semigroup operation of $F(V \cup C)$ by juxtaposition and do not distinguish in notation a string formed of a single element (of $V \cup C$) from that element.) Terminal productions of c are those of its productions which are elements of $F(V)$ (i. e. formed of words only); and the language L generated by a context-free grammar $G = \langle V, C, R\rangle$ is defined as $L = \langle V, C, f\rangle$ where, for any category $c \in C$, $f(c)$ is the set of all terminal productions of c. (Also, any language $L' = \langle V, C', f'\rangle$ with $C' \subset C$ where f' is the mapping f above restricted to C' could be regarded as a language generated by G as well.) A language L is a context-free language if it is generated by some context-free grammar.

2. The intuitive representation of context-free languages about which I shall speak is a representation by means of flag diagrams. A flag diagram is an ordered septuple $D = \langle V, C, H, f_1, f_2, g_1, g_2\rangle$, where V and C are disjoint, non-empty, finite sets; H is a finite oriented graph; f_1 and f_2 are mappings of two disjoint, non-empty subsets P_1 and P_2, respectively, of the set P of points (vertices) of H onto C; and g_1 and g_2 are mappings of two disjoint subsets E_1 and E_2, respectively, of the set E of edges of H, of which E_1 is non-empty, onto V and into C, respectively. The sets V and C are called again vocabulary (set of words) and set of categories, respectively. A point $p_1 \in P_1$ and a point $p_2 \in P_2$ with $f_1(p_1) = f_2(p_2) = c \in C$ are called a starting c-point and an ending c-point, respectively; an edge $e_1 \in E_1$ with $g_1(e_1) = v \in V$ and an edge $e_2 \in E_2$ with $g_2(e_2) = c \in C$ are called a v-edge and a c-edge, respectively. Starting and ending c-points are marked by a flag-head, pointing to the left and to the right (i. e. by a pentagon with two horizontal, one vertical and two slant sides which form an angle pointing to the left and to the right), respectively, bearing the symbol c; v-edges e are marked by the word v written above the edge e, and c-edges are marked by a double flag-head, pointing to both sides (i. e. by a hexagon with two horizontal and four slant sides which form two angles, pointing to the left and to the

right), bearing the symbol c. (See Fig. 1.)

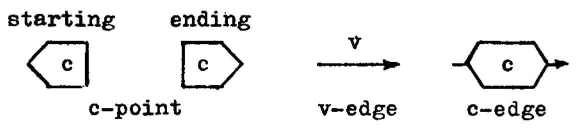

Fig. 1.

In the case $E_2 = \emptyset$ we call $D = \langle V, C, H, f_1, f_2; g_1, g_2\rangle$, which can be written for short as $D = \langle V, C, H, f_1, f_2, g_1\rangle$ for g_2 is the empty mapping, viz. the mapping of the empty set E_2 into C, a <u>finite state flag diagram</u>. In this case, an (oriented, possibly self-intersecting) path Q (i. e. going possibly several but a finite number of times through the same point or edge) of H is called a <u>c-path</u> of H ($c \in C$), if it leads from a starting c-point p_1 to an ending c-point p_2 but otherwise, does not go through any starting or ending c-point (implying the condition that Q has not to go through p_1 or p_2 once more).

Let be $e_1, e_2, \ldots, e_n$ the edges belonging to E_1 of a c-path Q of H, each written as many times as Q goes through it and written in the order in which Q goes through them. The word string $v_1 v_2 \ldots v_n$, where, for $i = 1, 2, \ldots, n$, $v_i = g_1(e_i)$, is called the <u>word string to be read along the c-path Q</u>. The <u>language represented by a finite state flag diagram</u> $D = \langle V, C, H, f_1, f_2, g_1\rangle$ is defined as $L = \langle V, C, f\rangle$, where, for any $c \in C$, f(c) is the set of all word strings to be read along some c-path of H. A language is a <u>finite state language</u> if it is represented by some finite state flag diagram.

Flag diagrams in general are generalizations of finite state flag diagrams. In the case of a flag diagram $D = \langle V, C, H, f_1, f_2, g_1, g_2\rangle$ in general, an (oriented, possibly self-intersecting) path Q of H is called a c-path of H ($c \in C$) if, besides leading from a starting c-point to an ending c-point but otherwise not going through any starting or ending c-point, it does not go through any c'-edge of H ($c' \in C$). The word string to be read along a c-path of H is defined in the same way as in the case of a finite state flag diagram.

In order to define the language represented by a flag diagram in general, we need still some auxiliary notions. Consider two different pints p_1 and p_2 of the oriented graph H of a flag diagram $D = \langle V, C,$

$H, f_1, f_2, g_1, g_2\rangle$. The <u>subgraph H' of H connecting</u> p_1 <u>with</u> p_2 consists, by definition, of all points p of H for which both from p_1 to p and from p to p_2 at least one (oriented) path leads, together with the edges which connect these points p in H. (If there is no such point p then H' is the empty graph; otherwise, both p_1 and p_2 are points of H'). A subgraph of H connecting a starting c-point with an ending c-point ($c \in C$), provided it is not empty, is called a <u>c-subgraph of H</u>.

The <u>derivatives of a flag diagram</u> $D = \langle V, C, H, f_1, f_2, g_1, g_2\rangle$ are defined by recursion as (i) D itself, and (ii) any flag diagram of the form $D' = \langle V, C, H', f_1', f_2', g_1', g_2'\rangle$ which can be obtained from some derivative $D'' = \langle V, C, H'', f_1'', f_2'', g_1'', g_2''\rangle$ of D by replacing one of its c-edges ($c \in C$) e by any c-subgraph H''' of H. Here, replacing has to be understood in the following sense. First, the starting c-point p_1 and the ending c-point p_2 of H connected by H''' are replaced by the starting point p_3 and the ending point p_4, respectively, of the edge e of H''; then, the graph H''' modified thus is inserted, instead of e, between p_3 and p_4 in H''. Any point or edge of H'' and H''' which was a starting c'-point, an ending c'-point, a v-edge or a c'-edge in H'' and H, respectively ($c' \in C$, $v \in V$), remains so after the replacement; in particular, p_3 and p_4 remain marked or unmarked as they were in H'' rather than getting marked by a flag-head, bearing the symbol c and pointing to the left and to the right, respectively, as p_1 and p_2, respectively, were marked in H.

Now, we define the <u>language represented by a flag diagram</u> $D = \langle V, C, H, f_1, f_2, g_1, g_2\rangle$ as $L = \langle V, C, f\rangle$ where, for any $c \in C$, $f(c)$ is the set of all word strings to be read along some c-path of the oriented graph H' of some derivative $D' = \langle V, C, H', f_1', f_2', g_1', g_2'\rangle$ of D.

As a simple example, Fig. 2. shows a flag diagram D representing the language $L = \langle V, C, f\rangle$ with $V = \{0, (,)\}$, $C = \{s\}$ and $f(s) = \{0, (0), ((0)), (((0))), \dots\}$. On Fig. 3, some of the derivatives of D are shown.

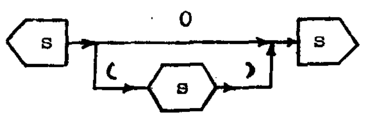

Fig. 2.

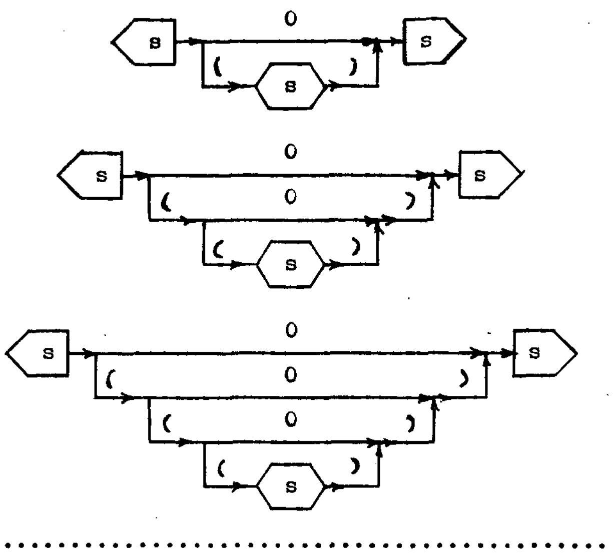

Fig. 3.

3. Obviously, any context-free language can be represented by some flag diagram. Indeed, let G = ⟨V, C, R⟩ be a context-free grammar. To each rule r = c:σ ∈ R, where σ = $s_1 s_2 \ldots s_n$, c ∈ C, s_1, s_2, ..., s_n ∈ V ∪ C, form an oriented graph H_r with n + 1 points p_0, p_1, p_2, ..., p_n of which p_0 is a starting c-point, p_n an ending c-point and, for i = 1, 2, ..., n, p_{i-1} is connected with p_i by an s_i-edge (i. e., for s_i = v ∈ V a v-edge, for s_i = c ∈ C a c -edge) e_i, oriented towards p_i. The (disconnected) union of these oriented graphs H_r, for all rules r ∈ R, defines a flag diagram obviously representing the language generated by G.

Using this constuction e. g. for the context-free grammar G = ⟨V, C, R⟩ with V = {0, (,)}, C = {s} and R = {s:0, s:(s)} generating the language represented by the flag diagram shown by Fig. 2, we should obtain the flag diagram shown by Fig. 4. Replacing this disconnected flag diagram by the connected one shown by Fig 2 corresponds to the AL-GOL 68-like way of writing s:0;(s) (or the ALGOL 60-like way of writing s::=0|(s)) of the rules belonging to R.

Besides the possibility of reduction of the number of starting

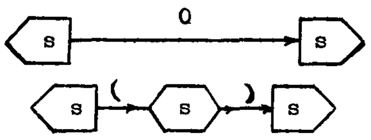

Fig. 4.

and ending c-points in a similar way, we have often the more important possibility of reduction of the number of c-edges. E. g. the language generated by the context-free grammar $G = \langle V, C, R \rangle$ with $V = \{a, b, \ldots, z, 0, 1, \ldots, 9\}$, $C = \{$letter, digit, identifier$\}$, $R = \{r_1, r_2, \ldots, r_{26}, r_{27}, r_{28}, \ldots, r_{36}, r_{37}, r_{38}, r_{39}\}$, where

r_1 = letter:a

r_2 = letter:b

....

r_{26} = letter:z

r_{27} = digit:0

r_{28} = digit:1

....

r_{36} = digit:9

r_{37} = identifier:letter

r_{38} = identifier:identifier letter

r_{39} = identifier:identifier digit

(here, space has been used between "identifier" and "letter" or "digit" to denote the semigroup operation), can be represented, instead of the flag diagram indicated by Fig. 5, which we get using the above constuction and for which the overall number of c-edges is 5, by the flag diagram indicated by Fig. 6, for which this number is 0, the language in question being actually a finite state language.

To show a more complicated example, take the language of the Church lambda conversion [4] where we use x, x|, x||, x|||, ... as variables and for simplicity (as a matter of fact, for obtaining a context-free language at all) we allow the abstraction $\lambda v[g]$ even if the variable v is not contained, or contained as a bound variable, in the (well-formed) formula g. This language is generated by the context-free grammar $G = \langle V, C, R \rangle$ with $V = \{x, |, \lambda, \{, \}, [,], (,)\}$, C =

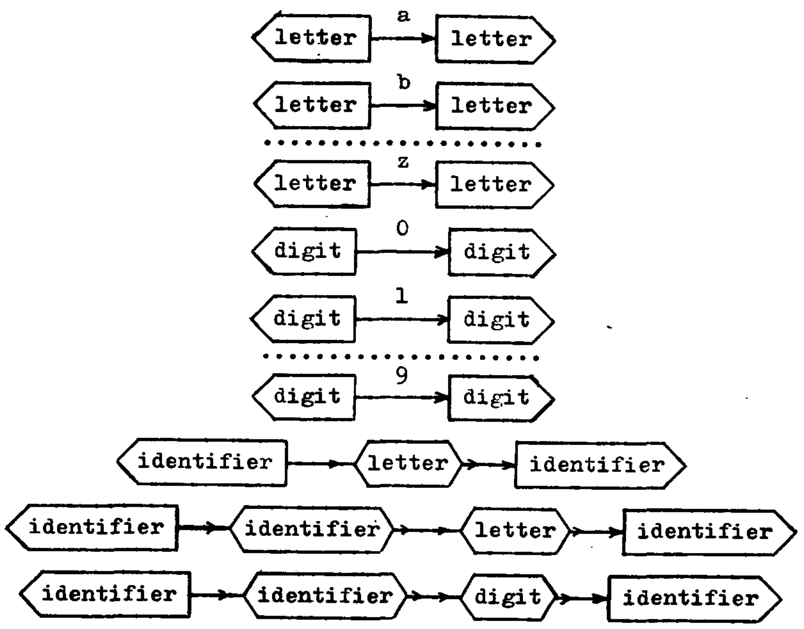

Fig . 5.

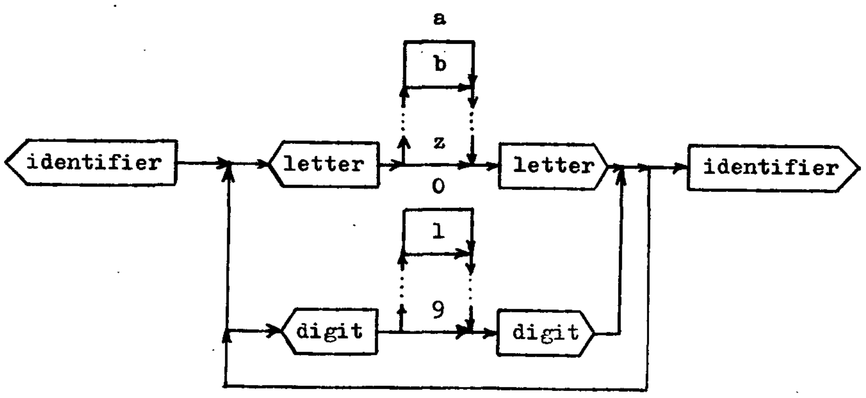

Fig . 6.

{variable, formula} and R = $\{r_1, r_2, r_3, r_4, r_5\}$, where

r_1 = variable:x

r_2 = variable:variable|

r_3 = formula:variable

r_4 = formula:{formula}(formula)

r_5 = formula:λvariable[formula]

(here, the semigroup operation is denoted by juxtaposition again). Fig. 7 shows a simple flag diagram representing this language. Here, the overall number of c-edges is 4.

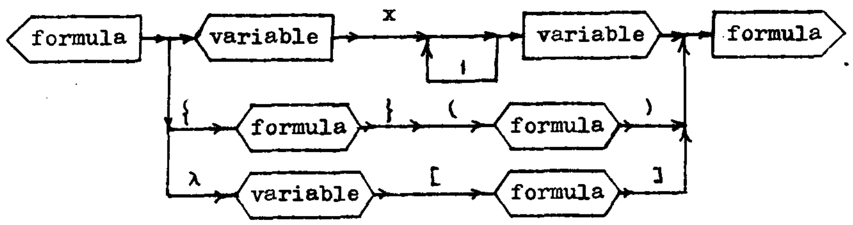

Fig. 7.

Flag diagrams can be used with advantage as a tutorial tool in teaching programming languages. In the case of a great number of categories, starting and ending c-points can be marked by flags with handle of different length for different categories c, rather than just flag-heads, pointing to the left and to the right, respectively, as shown by Fig. 8, serving to give a survey over the possible modes in ALGOL 68. Here, the flags instead of flag-heads are not really needed, for we have two categories only. However, inserting some more flags (which needs some more branchings too), we can get a flag diagram which is equivalent to the metaproduction rules of modes (with 25 categories) of [2], 1.2.1 .

Besides such tutorial use of flag diagrams, they might have a theoretical interest in furnishing a natural classification of context-free languages according to the minimum of the overall number of c-edges in the flag diagrams representing a given such language. This minimum can be considered as a measure of the non-finite state character of the given language. However, for such a theoretical use, a method for calculation of the minimum in question would be needed.

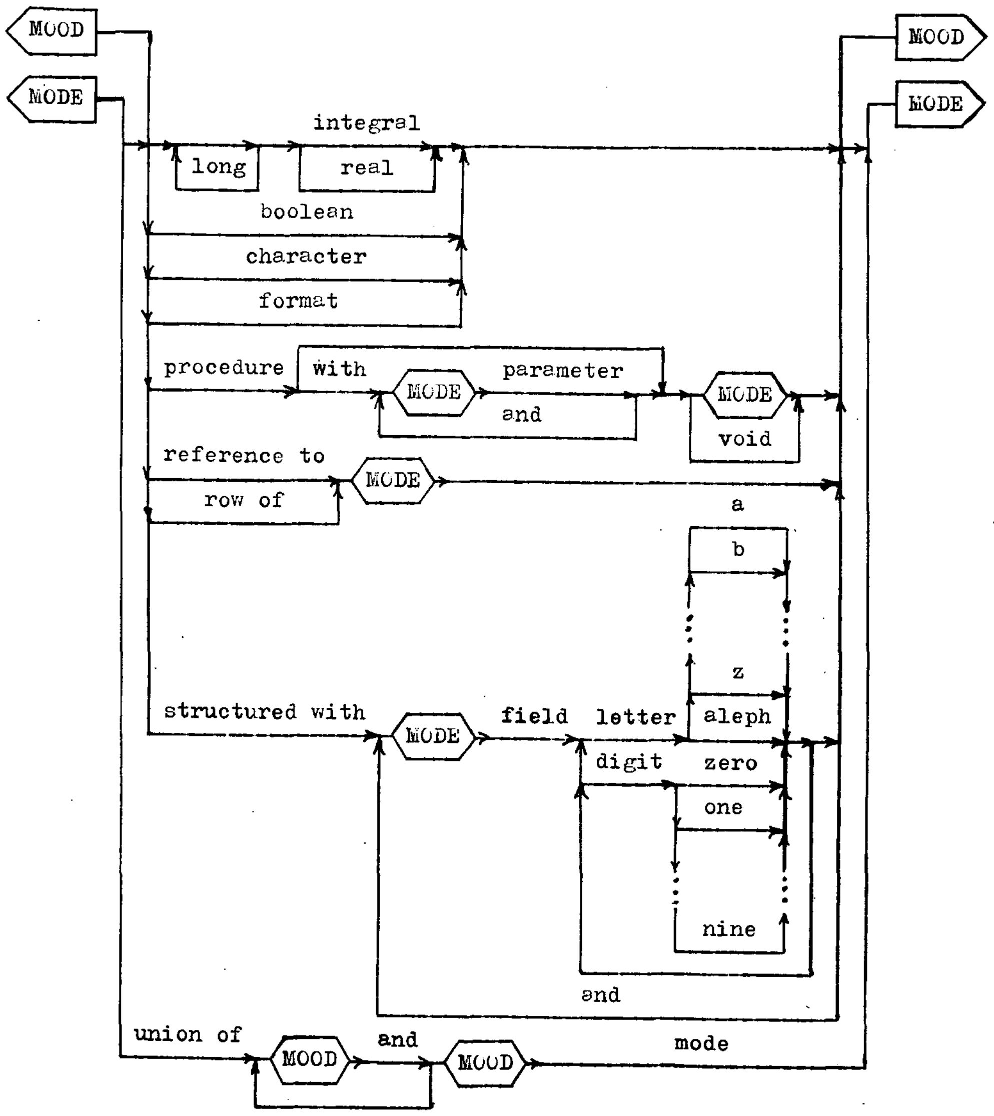
MOOD
MODE
MOOD
MODE
integral
long
real
boolean
character
format
procedure
with
MODE
parameter
and
MODE
void
reference to
row of
MODE
a
b
z
structured with
MODE
field
letter
aleph
digit
zero
one
nine
and
union of
MOOD
and
MOOD
mode

Fig. 8.

References

[1] See e. g. N. Chomsky, Three models for the description of language, IRE Transactions, 2 (1956), 113-124.

[2] See A. van Wijngaarden (editor), B. J. Mailloux, J. E. L. Peck, and C. H. A. Koster, Final Draft Report on the Algorithmic Language ALGOL 68, Stichting Mathematisch Centrum, Amsterdam, Rekenafdeling, MR 100 (1968).

[3] See P. Naur (editor), J. W. Backus, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. G. Wegstein, A. van Wijngaarden, and M. Woodger, Report on the Algorithmic Language ALGOL 60, Numerische Math., 2 (1960), 106-136, or Communications ACM, 3 (1960), 299-314.

[4] See e. g. A. Church, A set of postulates for the foundation of Logic, second paper, Annals of Math., (2) 34 (1933).