ANNALES ACADEMIAE SCIENTIARUM FENNICAE

Series A

I. MATHEMATICA

427

OPERATOR/OPERAND LANGUAGES

BY

V. J. NUMMI

CORRIGENDA

At the end of page 19, the following sentence should be inserted:
»Besides, the relation $\varrho_1 \mid \varrho_2$ holds between every pair of operators except when 1) ϱ_2 is a right parenthesis, or 2) ϱ_1 is either a $\equiv$ or a left parenthesis.»

Page 21, line 15:
»the condition formulated at the beginning of this section is met.»

should read:
»the prime phrases of sentential forms are unambiguously determined.»

HELSINKI 1968
SUOMALAINEN TIEDEAKATEMIA

doi:10.5186/aasfm.1969.427

Communicated 10 May 1968 by G. JÄRNEFELT and P. KUSTAANHEIMO

KESKUSKIRJAPAINO
HELSINKI 1968

Preface

The first draft of this paper was written during the academic year 1966—67 while I worked as a mathematician at the Computing Centre of the University of Helsinki. I wish to express my deepest gratitude to Professor Ilppo Simo Louhivaara, the director of the Computing Centre at that time, for the arrangements that gave me ample time, free from my other duties, to pursue this study. I also thank him for his encouragement and constant willingness to help me in all practical matters, without which the completion of this paper would have been difficult if not impossible.

To Professor Reino Kurki-Suonio I am indebted for his interest and for his valuable criticism.

Thanks are also due to Mr. Glenn Harma, B.Sc., for checking the language of the manuscript.

Veli Jaakko Nummi

Contents

I. Basic concepts

1. Preliminary definitions

Let X be a set and F_X the set of all finite sequences of elements of X. If $(\xi_1, \ldots, \xi_n)$ and $(\eta_1, \ldots, \eta_m)$ are two elements of F_X then we define their *product* as the concatenation

$$(\xi_1, \ldots, \xi_n)(\eta_1, \ldots, \eta_m) = (\xi_1, \ldots, \xi_n, \eta_1, \ldots, \eta_m).$$

With respect to the multiplication thus defined, F_X is a semi-group, more precisely, the *free semi-group generated by* X. If we identify the element ξ of X with the sequence (ξ), we have

$$(\xi_1, \xi_2, \ldots, \xi_n) = (\xi_1)(\xi_2)\ldots(\xi_n) = \xi_1\xi_2\ldots\xi_n.$$

Thus we always write the elements of F_X without commas and parentheses.

We say that the *length* of the sequence $\xi_1\ldots\xi_n$ is n.

The *free semi-group with identity generated by* X, W_X, is formed by adjoining to F_X the *empty sequence* ε, the length of which is 0 and which satisfies the condition that for any element x of W_X, $x = x\varepsilon = \varepsilon x$.

Two sequences $\xi_1\ldots\xi_n$ and $\eta_1\ldots\eta_m$ are considered equal if $n = m$ and $\xi_i = \eta_i$ for all $i = 1, \ldots, n$. The empty sequence is not equal to any sequence of non-zero length.

An *alphabet* is a nonempty finite set. The elements of an alphabet are called *letters* or *characters*. Sometimes an alphabet is also called a *vocabulary*.

A *word* over an alphabet V is an element of W_V. We shall use lower case italic letters to denote words.

The word x *contains* the word y if there exist two words v and w such that $x = vyw$. We also say that y *occurs* in x.

2. Grammars

A *context-free grammar* is a quadruple $G = (V, T, P, S)$, where

V is an alphabet,

T is a proper subset of V; we will denote $V \setminus T$ by N,

P is a finite set of ordered pairs (U, v), where U is an element of N and v is a word over V, and

S is an element of N.

Instead of writing $(U , v) \in P$ we will write $U \to v$. Such a pair is called a *production*. We will refer to U as the *left-hand side* of the production, and to v as its *right-hand side*. The length of v is called the *length* of the production. We assume that the number of productions is $r > 0$ and that the productions are numbered in an arbitrary but fixed order from 1 to r.

When x and y are two words, we write

$$x \rightarrow y$$

if there exists a pair of numbers (α , β) such that by applying the βth production to the αth character of the word x we obtain the word y, i.e. there exist words u, v, w, and a letter U such that

the length of u is $\alpha - 1$,

$U \to v$ is the βth production of the grammar,

$x = uUw$; $y = uvw$.

We may also write

$$x \xrightarrow{(\alpha, \beta)} y .$$

If x and y are words, we write

$$x \Rightarrow y$$

if there exists a sequence of words

$$w_0 , w_1 , \ldots , w_n$$

such that

$$x = w_0 , \quad y = w_n , \quad w_{i-1} \rightarrow w_i \quad (i = 1 , \ldots , n) .$$

In other words, there exists a sequence of ordered pairs of natural numbers,

$$d = (\alpha_1 , \beta_1)(\alpha_2 , \beta_2) \ldots (\alpha_n , \beta_n)$$

such that

$$w_{i-1} \xrightarrow{(\alpha_i, \beta_i)} w_i \qquad (i = 1 , \ldots , n) .$$

The sequence d is called the *derivation* of y from x. y is called an *x-derivative*. The pairs (α_i , β_i) are called *steps* of d. If x consists of a single character U, then y is a *U-phrase*. S-phrases are called *sentential forms*.

We call the elements of the alphabet T *terminal letters* or *terminal characters*. The elements of N will be referred to as *nonterminal letters* or *nonterminal characters*. A terminal *U-phrase* is a U-phrase which belongs to W_T. A terminal S-phrase is a *sentence*. The nonterminal letters can be thought of as syntactical variables; if U is a nonterminal letter, we could call the set of all U-phrases the syntactical category defined by U.

The set L_G of all sentences defined by the grammar G is called a *context-free language.*

We make the following assumptions about all grammars which we are going to discuss:

S is the only element of V which does not occur in the right-hand side of any of the productions.

If U is a nonterminal character different from S, then there exists a sentential form containing U; i.e. there exist words u and w such that $S \Rightarrow uUw$.

If U is a nonterminal character then there exists a terminal U-phrase.

The empty word is not a phrase, i.e. every production has a non-zero length.

(If G is a grammar defining a non-empty language L_G, then a grammar G' can be formed by making trivial changes to G, such that G' has the above properties and defines the same language, excluding only the empty word which may belong to L_G.)

3. The equivalence of derivations

We define for $i = 0, 1, 2, \ldots$ an operator Φ_i whose domain is the set of those derivations which contain at least $i + 1$ steps.

For $i = 0$ we put

$$\Phi_0 d = d .$$

When $i > 0$ the derivations belonging to the domain of Φ_i have the form

$$d = (\alpha_1 , \beta_1) \ldots (\alpha_i , \beta_i)(\alpha_{i+1} , \beta_{i+1}) \ldots (\alpha_n , \beta_n) .$$

We denote the length of the production β_j by l_j $(j = 1, \ldots, n)$. Now we have

$$\Phi_i d = (\alpha_1 , \beta_1) \ldots (\tilde{\alpha}_i , \tilde{\beta}_i)(\tilde{\alpha}_{i+1} , \tilde{\beta}_{i+1}) \ldots (\alpha_n , \beta_n) ,$$

where we distinguish between the following cases:

1) a) If $\alpha_{i+1} < \alpha_i$ then

$$\tilde{\alpha}_i = \alpha_{i+1} ; \qquad \tilde{\alpha}_{i+1} = \alpha_i + l_{i+1} - 1 .$$

b) If $\alpha_{i+1} > \alpha_i + l_i - 1$ then

$$\tilde{\alpha}_i = \alpha_{i+1} - l_i + 1 ; \qquad \tilde{\alpha}_{i+1} = \alpha_i .$$

In both a) and b) we have

$$\tilde{\beta}_i = \beta_{i+1} . \qquad \tilde{\beta}_{i+1} = \beta_i .$$

2) If $\alpha_i \leqq \alpha_{i+1} \leqq \alpha_i + l_i - 1$ then

$$\tilde{\alpha}_i = \alpha_i ; \quad \tilde{\alpha}_{i+1} = \alpha_{i+1} ,$$
$$\tilde{\beta}_i = \beta_i ; \quad \tilde{\beta}_{i+1} = \beta_{i+1} .$$

The operations Φ_i are called *transpositions*.

We use the notation

$$d \simeq \tilde{d}$$

if there exists a sequence of transpositions $\psi^{(j)}$ $(j = 1 , \ldots , m)$ such that

$$\tilde{d} = \psi^{(1)}\psi^{(2)} \ldots \psi^{(m)} d .$$

The relation $\simeq$ is an equivalence relation; we call the derivations d and $\tilde{d}$ *equivalent*. If d is the derivation of the word y from the word x then $\tilde{d}$ is also a derivation of y from x and consists of the same productions as d.

Each equivalence class can be represented by its *leftmost* derivation: a derivation

$$(\alpha_1 , \beta_1) \ldots (\alpha_n , \beta_n)$$

is called leftmost if $\alpha_{i-1} \leqq \alpha_i$ for $i = 2 , \ldots , n$.

If for every sentence x of the language L_G there is only one leftmost derivation of x from S the grammar G is *unambiguous*. Otherwise the grammar is *ambiguous*.

4. Abstract trees

Suppose E is a given finite nonempty set on which a relation R is defined with the following properties:

If xRy and yRz then xRz.

If xRy is true then yRx is not true.

There exists an element e of E such that for any element x of E different from e the relation eRx is true.

If we use $x\tilde{R}y$ to denote that either $x = y$ or xRy or yRx then xRz and yRz together imply $x\tilde{R}y$.

The couple (E , R) is called an *abstract tree*. The elements of E are called the *nodes* of the tree and e is called the *root*. If xRy we call y a *descendant* of x, and x an *ancestor* of y. Those nodes which have no descendants are called *leaves*.

A nonempty subset E_1 of E is a *cut set* if for each element x of E:

$x \in E_1$ if and only if there does not exist any element z of E_1 such that xRz or zRx.

So for example the singleton $\{e\}$ and the set of the leaves of the tree are cut sets.

The relation R gives rise to another relation R' defined as follows: $xR'y$ means that zRy if and only if either $z = x$ or zRx.

If $xR'y$ we call y a *successor* of x, and x, the *predecessor* of y.

Two abstract trees (E_1, R_1) and (E_2, R_2) are *isomorphic* if there exists a one-to-one mapping f of E_1 onto E_2 such that for any two elements x and y of E_1,

$$xR_1y \text{ if and only if } fx\, R_2\, fy\,.$$

If V is an alphabet and (E, R) is an abstract tree, and F is a given mapping of E into V, then we call the triple (E, R, F) a *labeled tree*. If x is a node, then Fx is the *label* of x.

Two labeled trees (E_1, R_1, F_1) and (E_2, R_2, F_2) are isomorphic if there exists a one-to-one mapping f of E_1 onto E_2 such that for any two elements x and y of E_1,

$$xR_1y \text{ if and only if } fx\, R_2\, fy\,,$$

$$F_1x = F_1y \text{ if and only if } F_2fx = F_2fy\,.$$

If we have a tree (E, R) we can (usually in several different ways) define on E a relation Q subject to the following conditions:

If xQy and yQz then xQz.

If xQy is true then yQx is not true.

$x\tilde{R}y$ is true if and only if neither xQy nor yQx.

If xQy and $x\tilde{R}z$ and $y\tilde{R}t$ then either zQt or $z\tilde{R}t$.

If xQy we say that x is to the *left* of y, or, equivalently, that y is to the *right* of x.

The relation Q defines a unique left-to-right ordering on every cut set: if $x \neq y$ then either xQy or yQx. A cut set whose elements are ordered from left to right is an *ordered cut set*.

The triple (E, R, Q) is an *ordered tree*.

Two ordered trees (E_1, R_1, Q_1) and (E_2, R_2, Q_2) are isomorphic if there exists a one-to-one mapping f of E_1 onto E_2 such that for any two elements x and y of E_1,

$$xR_1y \text{ if and only if } fx\, R_2\, fy\,,$$

$$xQ_1y \text{ if and only if } fx\, Q_2\, fy\,.$$

The property of being isomorphic defines an equivalence relation on the set of all ordered trees. The equivalence classes may be represented by so-

called *sequence trees*. The sequence tree $\mathcal{M} = (\mathcal{E}, \mathcal{R}, \mathcal{Q})$ isomorphic to a given tree $M = (E, R, Q)$ can be constructed as follows.

The elements of the set $\mathcal{E}$ are finite sequences of positive integers. The root of $\mathcal{M}$ is the sequence »(1)», the only sequence having only one component. The mapping f maps the element e of E to this sequence:

$$fe = (1) .$$

Then we proceed inductively: If we already have defined a node ξ of $\mathcal{M}$,

$$\xi = (\alpha_1, \ldots, \alpha_k)$$

such that $\xi = fx$ for some x in E, then

if x is a leaf of M, then ξ will be a leaf of $\mathcal{M}$,
if the successors of x are $y_1, \ldots, y_h$ when listed from left to right then we define

$$\eta_i = (\alpha_1, \ldots, \alpha_k, i)$$

and let

$$\xi \mathcal{R}' \eta_i , \qquad f y_i = \eta_i , \qquad \eta_i \mathcal{Q} \eta_j \text{ if and only if } i < j$$

where $i, j = 1, \ldots, h$.

Thus, if ξ and η are two nodes of the sequence tree then

1) $\xi \mathcal{R} \eta$ if and only if there exist two integers k and h $(h > k)$ such that

$$\xi = (\alpha_1, \ldots, \alpha_k) , \qquad \eta = (\alpha_1, \ldots, \alpha_k, \alpha_{k+1}, \ldots, \alpha_h) ;$$

2) $\xi \mathcal{Q} \eta$ if and only if there exist three integers k, h, j $(h, j > k)$ such that

$$\xi = (\alpha_1, \ldots, \alpha_k, \beta_{k+1}, \ldots, \beta_h) ,$$

$$\eta = (\alpha_1, \ldots, \alpha_k, \gamma_{k+1}, \ldots, \gamma_j) ,$$

$$\beta_{k+1} < \gamma_{k+1} .$$

If $M = (E, R, Q)$ is an ordered tree (where E contains at least two elements) then (E_1, R_1, Q_1) is a *subtree* of M if

E_1 is a proper subset of E,
R_1 is the restriction of R to E_1,
Q_1 is the restriction of Q to E_1,
(E_1, R_1, Q_1) has a root.

We also say that (E_1, R_1, Q_1) is the result of *pruning off* those nodes which are elements of $E \setminus E_1$.

Let M be a tree, and E_1, a cut set of M. The *stump* of M defined by

E_1 is the subtree of M obtained by pruning off the descendants of all elements of E_1.

The triple $M = (E , R , Q)$ is called an *ordered forest* if E is a finite non-empty set and R and Q are two relations on E such that all axioms of an ordered tree are fulfilled except the existence of a root.

A *branch* $M_1 = (E_1 , R_1 , Q_1)$ of an ordered tree $M = (E , R , Q)$ is a subtree of M such that E_1 consists of a node x of M and all its descendants. M_1 is *defined by* x. If A is a cut set then each member of A defines a branch such that the set of branches is an ordered forest.

A *labeled ordered tree* is a quadruple (E , R , Q , F) such that

(E , R , F) is a labeled tree,
(E , R , Q) is an ordered tree.

If we have a totally ordered set of nodes of such a tree, then the *label* of this set is defined to be the word formed by juxtaposing the labels of the elements of the set in question from left to right in the sense of Q.

5. Derivation trees

Let a U-phrase v have a derivation from U,

$$(\alpha_1 , \beta_1) \ldots (\alpha_n , \beta_n) .$$

Then we have a sequence of words over V,

$$w_0 , w_1 , \ldots , w_n$$

such that

$$w_0 = U , \qquad w_n = v , \qquad w_{i-1} \xrightarrow{(\alpha_i , \beta_i)} w_i \qquad (i = 1 , \ldots , n)$$

We construct a sequence of labeled sequence trees

$$M_0 , M_1 , \ldots , M_n$$

where

$$M_i = (E_i , R_i , Q_i , F_i)$$

in the following way.

First we form M_0:

$$E_0 = \{(1)\} ,$$

R_0 and Q_0 are empty.

$$F_0(1) = U .$$

For $i = 1 , \ldots , n$ we proceed as follows. The tree

$$M_{i-1} = (E_{i-1} , R_{i-1} , Q_{i-1} , F_{i-1})$$

is a labeled sequence tree which has the word w_{i-1} as the label of the ordered set of its leaves. The production β_i applies to the label of the α_ith leaf of M_{i-1}. Let that leaf be the sequence $(\xi_1, \ldots, \xi_p)$ and its label, A. Let the production β_i be $A \to B_1 \ldots B_k$. Then

$$E_i = E_{i-1} \cup \{x_1, \ldots, x_k\}$$

where for $j = 1, \ldots, k$,

$$x_j = (\xi_1, \ldots, \xi_p, j),$$

$$F_i x_j = B_j.$$

The relations R_i and Q_i are determined by the requirement that M_i be a sequence tree. To have F_i completely defined we require that if $y \in E_{i-1}$ then $F_i y = F_{i-1} y$.

One immediately sees that if we made valid assumptions about M_{i-1} then the corresponding also hold conditions for M_i.

The end result is the tree M_n, which has as the label of the ordered set of leaves the word v. M_n is the *derivation tree* corresponding to the given derivation.

We observe that if d and $\tilde{d}$ are two derivations of v from U such that $\tilde{d}$ is obtained as the result of applying a transposition to d then d and $\tilde{d}$ have the same derivation tree. Thus, equivalent derivations have the same derivation tree. On the other hand, if we are given a derivation tree we can in a unique way construct the leftmost derivation corresponding to it. As a consequence we have the theorem:

Two derivations of a U-phrase v from U are equivalent if and only if they have the same derivation tree.

So if a grammar is unambiguous, there exists only one derivation tree for the derivation of an arbitrary sentential form x from S.

6. The syntactic analysis

In *syntactic analysis* we are concerned with the problem of recognizing the sentential forms of a given grammar and assigning structure to them. That is, we are given a word x and we must find a derivation of x from S, if there is any.

If we have a sentential form

$$s = xuy$$

such that

$$S \Rightarrow xUy, \qquad U \Rightarrow u$$

then u is called a *phrase of* s. If u contains at least two characters but

no phrase of s (other than itself) of length > 1, then u is a *prime phrase*. The »bottom-up» method of syntactic analysis works in the following way: The sentential form is inspected in order to recognize its prime phrases, and each prime phrase is replaced by the nonterminal character from which it was derived. Then we have a new word which is treated in the same way as the given word. Finally we arrive at a one-letter word.

The problem arising here is that the word to be analyzed may contain a sequence of characters that could have been derived from a nonterminal character but which is not a phrase of the sentential form. That is, we have a partitioning of s into three sub-words

$$s = abc$$

such that there is a nonterminal character B so that

$$B \Rightarrow b$$

but

$$S \not\Rightarrow aBc\,.$$

Thus we may have to try a great number of partitionings in order to find a derivation of s from S. In any case, the number is finite for any fixed s.

A question of major importance in connection with languages and grammars is how to exploit the particular properties of a given grammar or a given class of grammars to cut down the number of steps needed in the syntactic analysis of the sentential forms of the grammar. If the bottom-up method of analysis is used, the approach is to find some easily-checked properties of the sub-words a and c which can be used as criteria by means of which we can tell whether the sub-word b of the word abc is a prime phrase of a sentential form or not, when there is a derivation $B \Rightarrow b$.

II. Operator/operand grammars

1. Definition

A grammar $G = (V\,, T\,, P\,, S)$ is an *operator/operand grammar* if the alphabet V can be partitioned into two classes R and D such that

1) $$R \cap D = \emptyset\,, \qquad R \cup D = V$$

2) all productions take either the form

a) $$A \to B$$

where A and B are two characters belonging to the same class, or the form

b) $$A \to a$$

where a is a word which contains at least one character from each class.

Until now the classes are interchangeable. We break the symmetry by fixing the classes so that S is a member of D. We call the elements of R *operators*, and the members of D, *operands*.

We also call an operator/operand grammar an R/D *grammar*.

Later we will make use of a further restriction:

If ϱ is a nonterminal operator such that

$$\varrho \to x_1 U_1 y_1$$
$$U_1 \to x_2 U_2 y_2$$
$$- - - - -$$
$$U_{k-1} \to x_k \varrho y_k$$

where for each i, x_i and y_i are words in W_V, and each $U_i \in N$, then at least one of the U-characters is an operand.

This is a restriction on the kind of recursiveness that is allowed for the operators. As an example, the grammar

$S \to E$ $\qquad$ $h \to ht+$

$E \to t$

$E \to ht$ $\qquad$ $R = \{+ , h\}$

$h \to t+$ $\qquad$ $D = \{S , E , t\}$

does not fulfil this requirement whereas the grammar

$S \to E$ $\qquad$ $R = \{+ , h\}$

$E \to t$ $\qquad$ $D = \{S , E , t\}$

$E \to ht$

$h \to E+$

does. (It is obvious that both grammars are unambiguous and generate the same language.)

2. The operator/operand structure tree

In what follows we shall make use of the concept of a *compound operator*. So we give here the definition:

Let there be given an R/D grammar G. We consider the set of all derivations of the form

$$A \to x_1 \rightarrow \ldots \rightarrow x_n \qquad (n \geqq 1)$$

where A is an operand and (in case $n > 1$) for $i = 2, \ldots, n$,

$$x_{i-1} \xrightarrow{(\alpha_i, \beta_i)} x_i$$

with the left hand side of β_i an operator; x_n contains only operands and terminal operators. In the grammar there are only a finite number of derivations of this kind because of our restriction on the recursiveness of operators. Thus we may introduce an auxiliary alphabet V_a, disjoint from V, such that the characters of V_a are in one-to-one correspondence with these derivations. These characters are called compound operators.

Now we are able to discuss the particular kind of structure of the sentences of an R/D language: all phrases of length > 1 are operator/operand combinations. This structure can be represented by means of a labeled tree which can be constructed, given the derivation tree of the sentence.

Let s be a sentence of a language generated by an operator/operand grammar G. Let M be the derivation tree of s.

If x is a node labeled by an operand we proceed as follows:

If x has only a single successor we delete x.

If x has several successors we consider the branch M_x defined by x. There exists at least one cut set of this branch such that the labels of the elements of the cut set are either operands or terminal operators. We select the minimal stump of M_x defined by such a cut set. A compound operator is associated with this stump. We prune off from this stump all nodes labeled by operators. Then we use the compound operator in question as the label of the node x.

We apply this procedure repetitively, starting with the root of M and working towards the leaves. The end result is a tree having terminal operands as the labels of the leaves and compound operators as the labels of the other nodes.

The sequence tree isomorphic to this tree, with the same labelling of the corresponding nodes, is the *R/D structure tree* of the sentence s.

The R/D structure tree is uniquely determined by the derivation tree of the sentence. The derivation tree in turn can be uniquely constructed from the R/D structure tree if the grammar does not contain such obvious ambiguities as

$$\begin{aligned} A &\to B_1 \to \ldots \to C \\ A &\to B_2 \to \ldots \to C \end{aligned} \qquad (A, B_1, B_2, C \in V)$$

or

$$A \to \ldots \to A \qquad (A \in V)$$

which are impossible to reconstruct once the above procedure is applied.

3. Postfix and prefix grammars

An R/D grammar is a *postfix grammar* if all productions of length > 1 take the form

$$A \to \delta_1 \ldots \delta_k \varrho \qquad (k \geqq 1)$$

where the δ's are operands and ϱ is an operator.

The sentences of a postfix language are formed by listing the labels of the nodes of the R/D structure tree in the following order:

We start with the leftmost leaf. As soon as the label of a node has been listed, we prune off the node. If the resulting tree is not empty we apply the same procedure to it.

An R/D grammar is a *prefix grammar* if all productions of length > 1 take the form

$$A \to \varrho \delta_1 \ldots \delta_k \qquad (k \geqq 1)$$

where the δ's are operands and ϱ is an operator.

The sentences of a prefix language are formed by listing the labels of the nodes of the R/D structure tree in the following order:

We start with the root. As soon as the label of a node has been listed, we prune off the node. If the resulting ordered forest is not empty we apply the same procedure to its leftmost tree.

To be accurate we should have distinguished between compound operators, which are labels of the nodes of the R/D structure tree, and the operators of the prefix or postfix language itself. In this case, however, there is no danger of confusion, because the set of compound operators is in a natural correspondence (possibly many-to-one) with the set of the terminal operators of the grammar.

If a postfix or prefix grammar is unambiguous we can construct an unambiguous grammar of the same type, generating the same language, such that

all operators are terminal characters, and

all operands in productions of length > 1 are nonterminal characters.

Moreover, the R/D structure tree of each sentence remains the same in this modified grammar as in the original one.

Thus we are allowed to assume that a postfix or prefix grammar always meets these two conditions.

4. Precedence grammars

Let $G = (V, T, P, S)$ be an R/D grammar. We define on the operator alphabet R the following four relations.

1) If there exist a production

$$A \to xBy\varrho_2 z$$

and a derivation

$$B \Rightarrow u\varrho_1 v$$

such that

$$v\,,\, y \in W_D\,, \qquad \varrho_1\,,\, \varrho_2 \in R\,, \qquad x\,,\, u\,,\, z \in W_V$$

then

$$\varrho_1 \cdot > \varrho_2\,.$$

If $\varrho_2 \in N$ and there exists and operator ϱ_3 such that $\varrho_2 \Rightarrow \varrho_3$ then the definition is extended so that also

$$\varrho_1 \cdot > \varrho_3\,.$$

2) If there exist a production

$$A \to x\varrho_1 yBz$$

and a derivation

$$B \Rightarrow u\varrho_2 v$$

such that

$$u\,,\, y \in W_D\,, \qquad \varrho_1\,,\, \varrho_2 \in R\,, \qquad x\,,\, v\,,\, z \in W_V$$

then

$$\varrho_1 <\cdot\, \varrho_2\,.$$

If $\varrho_1 \in N$ and there exists an operator ϱ_3 such that $\varrho_1 \Rightarrow \varrho_3$ then the definition is extended so that also

$$\varrho_3 <\cdot\, \varrho_2\,.$$

3) If there exists a production

$$A \to x\varrho_1 y\varrho_2 z$$

such that

$$y \in W_D\,, \qquad \varrho_1\,,\, \varrho_2 \in R\,, \qquad x\,,\, z \in W_V$$

then

$$\varrho_1 \doteq \varrho_2\,.$$

If $\varrho_1 \doteq \varrho_2$ and ϱ_3 and ϱ_4 are two operators such that the conditions

either $\varrho_1 = \varrho_3$ or $\varrho_1 \Rightarrow \varrho_3$, and

either $\varrho_2 = \varrho_4$ or $\varrho_2 \Rightarrow \varrho_4$

hold, then the definition is extended so that

$$\varrho_3 \doteq \varrho_4 .$$

4) If there exist a production

$$A \to xByCz$$

and two derivations

$$B \Rightarrow t\varrho_1 u , \qquad C \Rightarrow v\varrho_2 w$$

such that

$$x , t , w , z \in W_V , \qquad u , y , v \in W_D ,$$

and both $t\varrho_1 u$ and $v\varrho_2 w$ are of length $\geqq 2$, then

$$\varrho_1 \mid \varrho_2 .$$

These four relations are called *precedence relations.* If for any ordered pair of operators ϱ_1 , ϱ_2 at most one of the relations 1) — 3) holds and if in addition the relations 3) and 4) are mutually exclusive, then the grammar G is called a *precedence grammar.*

If in any sentential form of a precedence grammar G an operator ϱ_2 can occur as the nearest operator to the right of another operator ϱ_1 then at least one of the precedence relations must hold between the ordered pair of operators ϱ_1 , ϱ_2. If, in particular, $\varrho_1 \doteq \varrho_2$ then in all sentential forms of the type

$$a\varrho_1 b\varrho_2 c$$

where

$$a , c \in W_V , \qquad b \in W_D ,$$

ϱ_1 belongs to a phrase p of length > 1 if and only if ϱ_2 belongs to p.

In the syntactic analysis, it is favourable to have »end-markers» on the word to be analyzed. So, if we have a grammar

$$G = (V , T , P , S) \qquad (N = V \setminus T)$$

we introduce a new grammar

$$G' = (V', P', T', S') \qquad (N' = V' \setminus T')$$

such that

$$\begin{aligned} N' &= N \cup \{S'\} && (S' \notin V) \\ T' &= T \cup \{\vdash , \dashv\} && (\vdash , \dashv \notin V) \\ P' &= P \cup \{S' \to \vdash S \dashv\} \end{aligned}$$

The two new terminal characters $\vdash$ and $\dashv$ are called *end-markers:* they are used only to enclose each sentential form of G'.

If G is an R/D grammar, so is G'. It is convenient to regard the end-markers as operators. If G is a precedence grammar then so is G': the relation

$$\vdash \doteq \dashv$$

holds between the end-markers; if an operator ϱ_1 can occur as the leftmost operator in a sentential form of G then

$$\vdash \lessdot \varrho_1 ;$$

if an operator ϱ_2 can occur as the rightmost operator in a sentential form of G then

$$\varrho_2 \gtrdot \dashv .$$

These are the only possible precedence relations between the end-markers and other operators.

The grammar G' is a *precedence grammar with end-markers*. Every sentential form of G' contains at least three characters.

5. An example

As an example of a precedence grammar we present the following distorted way of writing Boolean expressions (without negation) in a single variable λ:

$S \to E$	$I \to \supset ID$	$C \to \wedge CP$
$E \to I$	$D \to C$	$P \to \lambda$
$E \to IE \equiv$	$D \to CD \vee$	$P \to (E)$
$I \to D$	$C \to P$	

$$R = \{\equiv , \supset , \vee , \wedge , (,)\}$$
$$D = \{S , E , I , D , C , P , \lambda\}$$

By testing all possibilities we obtain the following table of precedence relations, where the left-hand operator determines the row and the right-hand operator the column.

	$\equiv$	$\supset$	$\vee$	$\wedge$	(	)
$\equiv$	$\gtrdot$					$\gtrdot$
$\supset$	$\gtrdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\gtrdot$
$\vee$	$\gtrdot$		$\gtrdot$			$\gtrdot$
$\wedge$	$\gtrdot$		$\gtrdot$	$\lessdot$	$\lessdot$	$\gtrdot$
(	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\doteq$
)	$\gtrdot$		$\gtrdot$			$\gtrdot$

As an example of a sentence in this grammar we give the word

$$\supset\supset\supset\lambda\lambda\lambda\vee\lambda\lambda\vee\lambda\lambda\equiv\equiv$$

6. The syntactic analysis of precedence languages

In this section we discuss special classes of precedence grammars fulfilling additional conditions under which the syntactic analysis can be performed by taking advantage of the precedence relations.

The syntactic analysis of precedence languages can be based on the following fact:

If s is a sentential form of length > 3, then s can be represented in the form

$$s = x_0\varrho_0x_1\varrho_1x_2 \ldots x_{k-1}\varrho_{k-1}x_k\varrho_kx_{k+1} \qquad (k \geqq 2)$$

where

$$x_0 , x_{k+1} \in W_V , \qquad \varrho_0 , \ldots , \varrho_k \in R ,$$
$$x_1 , \ldots , x_k \in W_D$$

so that

$$\varrho_1 \doteq \varrho_2 \doteq \cdots \doteq \varrho_{k-1}$$

(this sequence of relations is empty if $k = 2$) and

$$\varrho_0 \dot{\neq} \varrho_1 \qquad \varrho_0 \cdot\!\!\not> \varrho_1$$
$$\varrho_{k-1} \dot{\neq} \varrho_k \qquad \varrho_{k-1} \not<\!\!\cdot\ \varrho_k .$$

Thus s can also be represented in the form

$$s = x_0\varrho_0y_1uz_k\varrho_kx_{k+1}$$

by putting

$$x_1 = y_1z_1 , \qquad x_k = y_kz_k$$
$$u = z_1\varrho_1x_2 \ldots x_{k-1}\varrho_{k-1}y_k$$

in such a way that u is a prime phrase.

When we make use of this fact in the syntactic analysis we scan through the word to be analyzed and fix our primary attention to the pairs of consecutive operators. In this way we can find at least one prime phrase in each scan. The resulting algorithm may imply several scans through the sentential form before the analysis is completed. Also, we are not always able to replace a prime phrase by the nonterminal character in question as soon as we encounter it when scanning through the word from left to right, say. The main cause for this phenomenon is that a reducible phrase may occur in a context where the next operator to the right of it has »higher»

precedence than the rightmost operator in the phrase. Secondly, there may exist several different productions with identical patterns of operators (but possibly differing patterns of operands) in the right-hand sides. Even in the case of exactly identical right-hand sides we may still have the minor difficulty of deciding to which nonterminal character the phrase should be reduced.

Here we have a not entirely uninteresting class of languages which are not necessarily analyzable by means of a left-to-right or right-to-left algorithm or an algorithm working from both ends toward the middle. Yet it is not difficult to formulate conditions which are sufficient to ensure the unambiguousness of the language. (Cf. Knuth [6], p. 611.) The grammar of the preceding section may serve as a simple example.

A couple of special cases deserve mentioning. First, if each operator of a precedence grammar occurs in the right-hand side of only one production, the condition formulated at the beginning of this section is met. Secondly, prefix and postfix grammars also belong to this category; here the syntactic analysis is trivial, because the sentence can be regarded as its own structural description.

An interesting class of grammars results from the further requirement that no production contain two adjacent operands in its right-hand side. Because of the restriction on the recursiveness of nonterminal operators, there is a limit to the number of adjacent operands in the sentential forms. Thus, if, during a scan through the sentential form, we encounter a sequence of characters that might be a prime phrase, we can test this possibility by inspecting a fixed number of characters to the left and to the right of the sequence. The grammar is thus a bounded-context grammar. (Cf. Floyd [5].)

Finally, we add the following requirement to the previous ones:

If the left-hand side of a production is an operator, then the right-hand side may neither begin nor end with an operand.

In this special case no sentential form can contain two adjacent operands. The grammar can be transformed by straightforward expansion so that the resulting grammar does not contain any nonterminal operators and fulfils the above condition. For the transformed grammar we have a theory virtually identical to that of R. W. Floyd [4].

The question of whether a given R/D grammar is a precedence grammar or not can be answered by means of a mechanical procedure that determines all ordered pairs of operators which can occur in phrases of the grammar, separated only by a (possibly empty) string of operands. The special properties of precedence grammars discussed in this section can also be mechanically tested.

7. Algol 60 as an operator/operand language

In the Algol 60 syntax [8] there are a few productions which do not fit into the framework of R/D grammars. If we want to treat this language as an operator/operand language we must

1) leave some of the productions out of consideration; in the syntactic analysis we correspondingly need some preliminary processing before the analysis proper,
2) make some minor revisions to the grammar which do not have any effect on the language itself.

To the first category belong the productions defining

⟨identifier⟩
⟨unsigned number⟩
⟨open string⟩
⟨letter string⟩

and the nonterminals needed in introducing them. So we regard all identifiers, unsigned numbers, and strings as terminal characters and let the preliminary processing take care of their actual representations. Thus we also do not have any use for ⟨letter⟩s, ⟨digit⟩s, decimal point, or the character $_{10}$. The space character can also be discarded because it can only have significance in strings. The comments can also be eliminated in the preliminary processing.

As for the second category, we take first the syntax for assignment statements. The productions defining ⟨left part⟩, ⟨left part list⟩, and ⟨assignment statement⟩ are replaced by the following three productions

⟨destination⟩ ::= ⟨variable⟩ | ⟨procedure identifier⟩
⟨left hand side⟩ ::= ⟨destination⟩ | ⟨left hand side⟩ := ⟨destination⟩
⟨assignment statement⟩ ::= ⟨left hand side⟩ := ⟨arithmetic expression⟩ |
⟨left hand side⟩ := ⟨Boolean expression⟩

Our specifications do not allow the nonterminal character ⟨empty⟩. Thus we must, at the cost of compactness, rewrite the productions for ⟨function designator⟩, ⟨procedure statement⟩, and ⟨procedure heading⟩.

⟨function designator⟩ ::= ⟨procedure identifier⟩ |
⟨procedure identifier⟩ (⟨actual parameter list⟩)
⟨procedure statement⟩ ::= ⟨procedure identifier⟩ |
⟨procedure identifier⟩ (⟨actual parameter list⟩)
⟨heading 1⟩ ::= ⟨procedure identifier⟩
⟨heading 2⟩ ::= ⟨procedure identifier⟩ (⟨formal parameter list⟩)
⟨heading 3⟩ ::= ⟨heading 2⟩ | ⟨heading 2⟩ ; ⟨value part⟩
⟨heading 4⟩ ::= ⟨heading 3⟩ | ⟨heading 3⟩ ; ⟨specification part⟩
⟨procedure heading⟩ ::= ⟨heading 1⟩ | ⟨heading 4⟩

In the syntax for procedure declarations we make the following further modifications:

⟨value part⟩ ::= **value** ⟨identifier list⟩
⟨specifier⟩ ::= **string** | ⟨type⟩ | **array** | **label** | **switch** | **procedure**
⟨specification⟩ ::= ⟨specifier⟩ ⟨identifier list⟩ |
⟨type⟩ **array** ⟨identifier list⟩ |
⟨type⟩ **procedure** ⟨identifier list⟩
⟨specification part⟩ ::= ⟨specification⟩ |
⟨specification part⟩ ; ⟨specification⟩
⟨procedure declaration⟩ ::=
procedure ⟨procedure heading⟩ ; ⟨procedure body⟩ |
⟨type⟩ **procedure** ⟨procedure heading⟩ ; ⟨procedure body⟩

If we regard the character **own** as an operand, we need not change the syntax for ⟨type declaration⟩ or ⟨array declaration⟩. If **own** is regarded as an operator, we will have the following productions

⟨type declaration⟩ ::= ⟨type⟩ ⟨identifier list⟩ |
own ⟨type⟩ ⟨identifier list⟩
⟨array declaration⟩ ::= **array** ⟨array list⟩ | ⟨type⟩ **array** ⟨array list⟩ |
own array ⟨array list⟩ | **own** ⟨type⟩ **array** ⟨array list⟩

By these changes we have merely converted the Algol 60 grammar into an R/D grammar. The resulting grammar, however, is not a precedence grammar. The main source of conflicting precedence relations is careless use of the semicolon, both in the above changes and in the productions which we have left untouched. In [4], R. W. Floyd avoids this difficulty by relaxing the requirement that there should be only a finite number of productions in the grammar. His approach also gives more meaningful parsings of the phrases, from the semantic point of view, than the original Algol 60 syntax.

The grammars for several languages in use can be fitted to the model of operator/operand grammars by using methods similar to those indicated above. There may, however, be instances where this is not possible. But one can incorporate into the algorithm for syntactic analysis, a special device for the exceptional cases. For example, we could modify the grammar to include some »null operators» or »null operands» which do not occur in the sentences but can be added in the course of a preliminary processing.

III. The translation problem

1. The concept of translation

Let there be given two languages L_1 and L_2 both containing infinitely many sentences. Then there exist infinitely many one-to-one mappings from L_1 onto L_2. Under what conditions is it adequate to call such a mapping a *translation*?

The basic requirement is, of course, that the *meaning* of the translated sentence be the same as the meaning of the original sentence. So we have the problem of defining the meanings of sentences. This is a very profound question, even in the case of computer programming languages, and it is not our purpose to discuss it here to a great extent. Basically any attempt to give a formal definition of the meaning of the sentences of a language is an only attempt to push the problem to some other branch of knowledge where the semantics is assumed to be better understood (though not necessarily formally defined).

In order to be able to treat this question by the methods of the formal theory of languages, we must assume that the grammar of a given language has been constructed with primary emphasis on the semantics. This assumption is a nontrivial one, because a language, if regarded as a mere set of sequences of characters, can be generated by infinitely many grammars. In fact, the only virtue of a grammar from a practical point of view is that being able to perform the syntactic analysis of a given sentence helps us understand the meaning of the sentence and, conversely, given the meaning, the grammar tells us how to express it. Thus every grammatical category must have a semantic function.

So our approach to the problem is that in addition to the languages L_1 and L_2 having the same *universe of discourse* we introduce a third formal system H by means of which we can express exactly the same things as by means of the languages L_1 and L_2. The system H need not be a language in the same sense as L_1 and L_2 (e.g. a context-free language). Instead we assume that H in some sense more directly represents the meaning of the sentences than L_1 and L_2.

Now, if the grammatical categories of L_1, as well as those of L_2, have their counterparts in H the order of magnitude of the problem is reduced so as to give us some hope of success in the attempt to handle it by formal methods.

In the foregoing we have discussed the translation from the whole language L_1 onto the whole language L_2. It would be more general to consider a mapping from a subset of L_1 onto some subset of L_2. We must, however, in some way restrict the choice of these subsets from the non-

denumerable collection of all subsets of the languages L_1 and L_2. It is intuitively clear that the subsets should be context-free languages. Then we may restrict our discussion to those sub-languages and disregard the original languages.

2. The translation of R/D languages

In the case of R/D grammars we formalize the »meaning» of the language by means of the R/D structure tree. Thus we regard the meaning of the compound operators and the terminal operands as something which requires no further explanation.

Let G_1 and G_2 be two R/D grammars. In both grammars every sentence has its own R/D structure tree. We assume that the grammars are unambiguous so that the tree is uniquely determined by the sentence. We denote the set of the structure trees of all phrases of the grammar G_1 by $M(G_1)$ and that of the grammar G_2 by $M(G_2)$.

The mapping

$$f: M(G_1) \to M(G_2)$$

is a *translation* if

1) f is bijective,

2) the (unordered) labeled tree fX is isomorphic to the labeled tree X whenever $X \in M(G_1)$.

These two conditions imply that f maps the labels of the nodes of the trees belonging to $M(G_1)$ onto the labels of the nodes of the trees belonging to $M(G_2)$ in such a way that

1) the mapping from the set of terminal operands of G_1 onto the set of terminal operands of G_2 is bijective,

2) the mapping from the set of compound operators of G_1 onto the set of compound operators of G_2 is bijective,

3) to each compound operator there corresponds a unique permutation which determines how the mapping f changes the order of the branches defined by the successors of a node labeled by the compound operator in question.

Now if f is a translation from $M(G_1)$ onto $M(G_2)$, it induces a mapping from the language $L(G_1)$ generated by G_1 onto the language $L(G_2)$ generated by G_2. We call this mapping a *translation from* $L(G_1)$ *onto* $L(G_2)$. We may denote this translation by the same letter f because there is no danger of confusion.

3. The prefix and postfix grammars associated with an R/D grammar

Earlier (Ch. II, Sect. 3) we mentioned the method of forming a sentence in a prefix or postfix language by starting from the R/D structure tree of the sentence. Thus we can construct for any unambiguous R/D grammar, a postfix and a prefix grammar in such a way that the R/D structure trees of corresponding sentences in the three grammars are isomorphic as *ordered* labeled trees. We have here complete freedom in choosing the characters forming the terminal and nonterminal alphabets of the prefix and postfix grammars. Apart from this fact, the prefix and postfix grammars are uniquely determined by the original R/D grammar and the partitioning of its alphabet into operators and operands. (There may be some degree of freedom in this partitioning and the decisions are best done on semantic grounds. If the semantics do not favor either class, it may be better from the point of view of translatability to classify a neutral character as an operator.)

The process of constructing the prefix and postfix grammars associated with a given R/D grammar can be mechanized. To find out whether or not a given grammar is an R/D grammar we can examine all partitionings of the total alphabet into two disjoint classes; the procedure can easily be refined to exclude most of the impossible partitionings so that it becomes feasible in practice. By inspecting first all productions of length 1 and 2, we can form »clusters» of characters $A_1, \ldots, A_i, B_1, \ldots, B_j$ such that all A-characters belong to one class and all B-characters belong to its complement. After that we can iteratively scan the longer productions and possibly combine some of the clusters into bigger ones. If the partitioning is not uniquely determined by this process we can include additional constraints based on semantic grounds, and then reiterate the process. The next step is to take an inventory of the compound operators. After that the productions are transformed one by one to prefix or postfix form. Derivations having a nonterminal operator as the left-hand side are ignored in this last phase, because they have been accounted for by the compound operators.

4. Translation from an R/D language into the associated prefix or postfix language and vice versa

The prefix or postfix language associated with a given R/D language can be regarded as a convenient way of representing the R/D structure trees of the sentences of the given language. Thus it is relatively easy to modify an algorithm for syntactic analysis of an R/D language to turn it into an algorithm for translation from the given language into the associated prefix or postfix language. It is somewhat simpler to generate the prefix representation of the R/D structure tree from right to left, and the postfix

representation from left to right, because the tree will otherwise split into a forest in the course of the generation process (Cf. Section II.3).

There are no particular difficulties in the translation into a given R/D language from the corresponding prefix or postfix language, because the phrases are easily recognized and transformed into their target language representations.

In the foregoing we never made mention of the use of parentheses and punctuation marks occurring in higher level programming languages. If their special nature is ignored, they must be treated just like the other operators. Thus for example the expression written in the usual arithmetic notation as

$$x + f(h\ ,\ k\ ,\ l) - y/z$$

will, according to our formal theory, have the postfix representation

$$x\ f\ h\ k\ ,\ l\ ,\ ap\ +\ y\ z/-$$

(*ap* is a special operator denoting functional application). The comma is here understood to be an operator which operates on a parameter list and a parameter to combine them into a new parameter list. — If the special nature of parentheses and punctuation marks is taken into account, there must be a special device for them in the translation algorithm. — Related topics have been discussed, for example, by Landin [7] and Čulík [3].

Let us finally take a quick look at the concept of translation as defined in Section III.2. We can now reduce the problem of translatability between two R/D languages to the problem of translatability between their associated prefix (or postfix) languages. The definition of translation given in Section III.2 allows only the permutation of operands and the transliteration of characters as permissible translations between the prefix (or postfix) languages.

Summary and comments

This paper is a theoretical study of a special class of context-free grammars, characterized by the property that all phrases of length > 1 are operator/operand combinations. We call such grammars operator/operand (or R/D) grammars. The languages generated by such grammars are called operator/operand (or R/D) languages. These languages are interesting because most programming languages currently in use are R/D languages or nearly R/D languages, insofar as the context-free properties are concerned.

In Chapter I we have striven toward as accurate definitions as possible. In the definition of a tree we have included finiteness because we have use only for finite trees. A more general approach would be to postulate the relation between predecessor and successor, instead of that between ancestor and descendant. We also give a formal definition of the equivalence of two derivations, and prove that this definition agrees with the conventional one, which is based on derivation trees. In the last section we discuss the bottom-up method of syntactic analysis. The reason why we restrict ourselves to this method is that it enables us to profit most from the operator/operand property of the grammar.

In Chapter II we discuss the syntactic properties of operator/operand languages. We have tried to keep as close as possible to the intuitive notion of operators and operands. The idea of subdividing the alphabet into operators and operands was present in many of the early papers on programming languages and their translation; cf. for example the article of Bauer and Samelson [1]. Colmerauer [2] explicitly uses the terms »operator» and »operand» and applies the theory of context-free languages; however, his definition of these terms differs from ours because of different goals. We have put more emphasis on the semantic interpretability than on the efficiency of syntactic analysis. Also the discussion about Algol 60 as an R/D language is intended to be just an example — in this case our method does not result in an algorithm for syntactic analysis essentially different from that of Floyd [4].

In Chapter III we give the rules for transforming a given R/D language into »Polish notation». The discussion about the translation from one R/D language into another R/D language is based on the idea of R/D structure trees, which reflect the operator/operand structure of the phrases of the R/D grammars. We have felt that the question about translation between two context-free languages is too general for a formal treatment, but the R/D languages constitute an interesting subclass of languages which is more amenable to such treatment.

Computing Centre,
University of Helsinki,
Finland

References

[1] BAUER, F. L., and K. SAMELSON: Maschinelle Verarbeitung von Programmsprachen. - Digitale Informationswandler, edited by WALTER HOFFMANN, Friedr. Vieweg & Sohn, Braunschweig, 1962, pp. 227–268.

[2] COLMERAUER, A.: Notions d'opérateurs dans une grammaire »context-free». - Revue Française d'Informatique et de Recherche Opérationnelle 1, 1967, pp. 55–97.

[3] CULÍK, K.: Well-translatable grammars and Algol-like languages. - Formal language description languages for computer programming, edited by T. B. STEEL, JR., North-Holland Publishing Company, Amsterdam, 1966, pp. 76–85.

[4] FLOYD, R. W.: Syntactic analysis and operator precedence. - J. Assoc. Comput. Mach. 10, 1963, pp. 316–333.

[5] –»– Bounded context syntactic analysis. - Comm. ACM 7, 1964, pp. 62–65.

[6] KNUTH, D. E.: On the translation of languages from left to right. - Information and Control 8, 1965, pp. 607–639.

[7] LANDIN, P. J.: The mechanical evaluation of expressions. - Comput. J. 6, 1964, pp. 308–320.

[8] Revised report on the algorithmic language Algol 60, edited by P. NAUR. - Comm. ACM 6, 1963, pp. 1–17. / Numer. Math. 4, 1963, pp. 420–453.

Printed June 1968