AUTOMATIC SYNTACTIC ANALYSIS AND THE PUSHDOWN STORE[1]

BY

ANTHONY G. OETTINGER

1. Introduction. The problems of syntactic analysis have received considerable attention in recent years from three types of investigators, namely: mathematical logicians interested in the structure of formal "artificial" languages, applied mathematicians concerned with the design and translation of languages suitable for programming automatic information-processing machines, and mathematical linguists seeking algorithms for automatic translation among "natural" languages or for automatic information retrieval. Although these three types of investigators have different central objectives, a perusal of the works listed in the bibliography should convince the reader that there is a strong underlying kinship not only in the problems under investigation, but also in the methods of study and in the kinds of solutions sought or obtained.

One important common problem is that of obtaining an algorithm for distinguishing sentences from nonsentences or, in more formal terms, well-formed strings from not well-formed strings. Although the pure form of this problem has received more attention from logicians than from the others, it is of equal importance in automatic programming, where a fail-safe translator or compiler capable of detecting and rejecting not well-formed input data could eliminate much tedious debugging effort, and in automatic translation, where the problem of guaranteeing that each sentence is well-formed and has been correctly analyzed still looms large.

A second common problem is that of obtaining the simplest, in some sense, of a set of otherwise equivalent algorithms. Since this is a problem of practicality and economy rather than of existence, it has been of more vital concern to applied mathematicians and to mathematical linguists seeking algorithms that can be executed in a reasonable time at a reasonable cost than to logicians for whom this matter may have only aesthetic significance, since an exhaustive treatment of all cases is theoretically sufficient whenever "all" is finite.

In many cases of interest both of these problems seem amenable to solution by the application of techniques based on the use of what some computer people have come to call a "pushdown" store, namely a linear array of storage locations in which information is entered or removed from one end only, in accordance with a "last-in-first-out" principle. Although the full range of applicability of pushdown store techniques still remains to be determined, the cases studied so far all have structures that can be described in terms of trees or graphs akin to trees.

The work detailed in this paper and sketched at an earlier conference (Oettinger [19; 20]) grew out of reflections on a syntactic analysis technique devised

[1] This work has been supported in part by the National Science Foundation and by the Rome Air Development Center of the United States Air Force.

by Rhodes [22; 23; 24] at the National Bureau of Standards and adopted with modifications by Sherry [29] and others at Harvard, on the formalization of the syntax of Łukasiewicz's parenthesis-free notation given by Burks, Warren, and Wright [3] and applied by Miehle [16] and Oettinger [18], on the analytic and explanatory linguistic models of Chomsky [5] and Yngve [32; 33], and the related psychological model of Miller [17], on the syntactic analysis theories of Wundheiler and Wundheiler [31], Bar Hillel [1; 2], Lambek [14], and Riguet [25], and on such theoretical studies of automatic programming as those by Rutishauser [27], Janov [12], and Ljapunov [15]. Several authors, particularly Kanner [13], Ingerman [9], and Gorn [8], have thought along similar directions. The importance and the simplicity of the pushdown store have been clearly and independently recognized by Samelson and Bauer [28], whose paper appeared just as this one was being completed.

The Rhodes method of "predictive" syntactic analysis is based on the observation that in scanning through a Russian sentence from left to right it is possible, on the one hand, to make predictions about the syntactic structures to be met further to the right and, on the other hand, to determine the syntactic role of the word currently being scanned by testing what previously made predictions it fulfills. The predictions are stored in a linear array called the "prediction pool" which behaves approximately as a pushdown store. Before a new sentence is scanned, a set of initial predictions is entered in the pool. The first word of the sentence is then admitted, and a test is made to see if the topmost prediction in the pool will accept it. If so, the successful prediction is erased from the prediction pool, and new predictions based both on lexical data about the word obtained from a dictionary and on syntactic rules embodied in the predictive analysis system are entered into the prediction pool on top of whatever earlier predictions may have remained there. The system is then ready to process the next word in the sentence. To a fair degree of approximation, this technique may be regarded as the inverse of the system for sentence generation outlined by Yngve [33], in which a pushdown store is clearly used.

The foregoing is necessarily a grossly simplified account of the predictive analysis method. Somewhat more detailed descriptions have been given by Rhodes [22; 23; 24], Sherry [29], and Oettinger [19]. Empirical results obtained for Russian at the National Bureau of Standards and at Harvard are extremely encouraging, but it must be strongly emphasized that no claim is made of any final solution of the problems of automatic translation. Recent work at Harvard by Bossert, Giuliano, and Grant on the application of predictive analysis to English is equally encouraging. Detailed reports on both Russian and English are in preparation.

Predictive analysis yields a description of the syntactic structure of a sentence in terms consonant, although not identical, with old-fashioned parsing, immediate constituent theory (e.g., Wells [30]), or phrase-structure theory (Chomsky [5]). It remains to analyze the exact relation of the predictive method to these theories, as well as to those of Bar Hillel, Lambek, and Riguet, and to those of the logicians who, like Ajdukiewicz, inspired the latter investigators. A model by Sherry, described in a report in preparation, that

extends some of the results given later in this paper to account for significant properties of predictive analysis, may help to shed more light on this question.

For present purposes, the most important properties of predictive analysis are (1) that in the ideal case and, indeed, in many simple practical cases, a correct syntactic analysis of a sentence is obtained after a single scan of the sentence from left to right, each word being used once and only once, and (2) that the method incorporates very natural checks for well-formation, and therefore has desirable fail-safe properties. As applied to natural languages in the general case, the technique of predictive analysis has an empirical, approximative and, in some instances, iterative character. It seemed natural to ask whether an exact and interesting theoretical counterpart exists for some suitable simple artificial languages. The remainder of this paper answers this question in the affirmative. Familiarity with the work of Burks, Warren, and Wright [3] is assumed in what follows; references to their article will henceforth be made with the abbreviation "BWW". It should be noted that the *register* in the Evaluator of BWW (p. 56) is essentially a pushdown store.

2. The languages P_1, P_2, P_3, and L. In the following definitions "δ_{jk}" designates the kth member of a set of elements of degree j. Elements δ_{0i} are usually called variables, and elements for which $j \geqq 1$ are called functors or operators of degree j. Discussion will be confined to $0 \leqq j \leqq 2$, since the "parenthetic" languages to be defined do not admit of functors of degree >2. The symbol "Δ_j" denotes an arbitrary formula in L, while "Δ_j^i" denotes an arbitrary formula in P_i. Left and right parentheses designate themselves. In all cases, "Λ" designates the null formula, and for any Δ, if $\Delta \neq \Lambda$, Δ is of finite length.

Well-formed formulas in the languages P_1, P_2, P_3, and L are defined as follows:

DEFINITION 1 (P_1). (a) δ_{0i}; and (b) if Δ_1^1 and Δ_2^1, then $(\delta_{1j}^1\Delta_1^1$, and also $(\Delta_1^1\delta_{2k}^1\Delta_2^1$.

DEFINITION 2 (P_2). (a) $\not\delta_{0i}$; and (b) if Δ_1^2 and Δ_1^2, then $\delta_{1j}^1\Delta_1^2)$, and also $\Delta_1^2\delta_{2k}^1\Delta_2^2)$.

DEFINITION 3 (P_3). (a) δ_{0i}; and (b) if Δ_1^3 and Δ_1^3, then $(\delta_{1j}^1\Delta_1^3)$, and also $(\Delta_1^3\delta_{2k}^1\Delta_2^3)$.

DEFINITION 4 (L). (a) δ_{0i}; and (b) if Δ_1 and Δ_2, then $\delta_{1j}\Delta_1$, and also $\delta_{2k}\Delta_2\Delta_1$.

P_1, P_2, and P_3 will be referred to respectively as left-parenthetic, right-parenthetic, and simple full-parenthetic languages. Except for the restriction to functors of degree $\leqq 2$, L is the Łukasiewicz parenthesis-free language, as described in BWW and in Rosenbloom [26], Chapter IV.

Mappings among P_1, P_2, P_3, and L are defined as follows:

DEFINITION 5. (a) $\Lambda \longleftrightarrow \Lambda$, $\delta_{0i} \longleftrightarrow \delta_{0i}$, that is, the null formula and the variables are, for the sake of simplicity, assumed to be the same in all four languages.

(b) if $\Delta_j^1 \longleftrightarrow \Delta_j^2 \longleftrightarrow \Delta_j^3 \longleftrightarrow \Delta_j$

and $\delta_{jk}^1 \longleftrightarrow \delta_{jk}$, $j > 0$, then

$$(\delta_{1j}^1\Delta_1^1 \longleftrightarrow \delta_{1j}^1\Delta_1^2) \longleftrightarrow (\delta_{1j}^1\Delta_1^3) \longleftrightarrow \delta_{1j}\Delta_1 ,$$

and also

$$(\Delta_1^1\delta_{2k}^1\Delta_2^1 \longleftrightarrow \Delta_1^2\delta_{2k}^1\Delta_2^2) \longrightarrow (\Delta_1^3\delta_{2k}^1\Delta_2^3) \longrightarrow \delta_{2k}\Delta_2\Delta_1 .$$

EXAMPLE 1. Let $\delta_{0i} = x_i$, $\delta_{11}^1 = \sim$, $\delta_{21}^1 = +$, $\delta_{22}^1 = \cdot$, $\delta_{11} = N$, $\delta_{21} = A$, $\delta_{22} = M$; then

$$(\sim((x_1 + x_2 \cdot x_3 \longleftrightarrow \sim x_1 + x_2) \cdot x_3)) \longrightarrow (\sim((x_1 + x_2) \cdot x_3)) \longrightarrow NMx_3Ax_2x_1 .$$

Algorithms for effecting the mappings of Definition 5 can easily be devised with the aid of a pushdown store. Let p be a pushdown store. Let the input formula be scanned character-by-character from left to right, and let the output formula be produced by adjoining each new character to the left of those previously generated. With these conventions, the following simple translation algorithms may be defined:

DEFINITION 6. Translation from P_2 to L.

If the current input character is

(1) a variable, adjoin it to the output formula;

(2) a right parenthesis, adjoin the character currently at the top of p to the output formula, then remove it from p;

(3) a functor, put its image (Definition 5) at the top of p.

DEFINITION 7. Translation from P_1 to L.

If the current input character is

(1) a left parenthesis, put a "v" at the top of p;

(2) a functor, replace the character currently at the top of p by the image (Definition 5) of the functor;

(3) a variable

(a) adjoin it to the output formula; then

(b) check p; if it is empty or has a "v" on top, proceed to the next input character; otherwise adjoin the character currently at the top of p to the output formula, then remove it from p, and repeat step (b).

EXAMPLE 2. (For ease in printing, the pushdown store is laid on its side, opening toward the left; viewed this way, it is obviously analogous to a tape on a Turing machine or on a real computer.) The formulas are those of Example 1. On each line, the current character, the pushdown store p, and the output formula are shown after application of the rules of Definition 6 or Definition 7.

$P_2 \rightarrow L$			$P_1 \rightarrow L$		
$\sim$	N	Λ	(	v	Λ
x_1	N	x_1	$\sim$	N	Λ
$+$	AN	x_1	(	$v\ N$	Λ
x_2	AN	x_2x_1	(	$v\ v\ N$	Λ
)	N	Ax_2x_1	x_1	$v\ v\ N$	x_1
$\cdot$	MN	Ax_2x_1	$+$	$Av\ N$	x_1
x_3	MN	$x_3Ax_2x_1$	x_2	$v\ N$	Ax_2x_1
)	N	$Mx_3Ax_2x_1$	$\cdot$	MN	Ax_2x_1
)	Λ	$NMx_3Ax_2x_1$	x_3	Λ	$NMx_3Ax_2x_1$

Note that in both Definition 6 and Definition 7 the characters adjoined to the output obviously depend on p as well as on the current character, but that in Definition 6 the sequence of operations depends solely on the current input character. In Definition 7, "v" clearly "marks the place" of a left parenthesis occurrence, and "v" is used instead of (simply to avoid compounding homography and autonymy. Example 2 also clearly shows that P_1, P_2, and P_3 are equivalent in the sense that grouping remains unambiguous even when either all left parentheses or all right parentheses are removed from an expression in P_3.

3. A notation for algorithms. Definitions 6 and 7 are simple enough to be readily understood as given. More complex algorithms require more formal and precise definitions, especially if it is of importance to establish certain of their characteristics by formal proofs. The notation of BWW is adequate for formal proofs, but it does not lend itself readily to a rapid grasp of the essence of a given algorithm nor to experimentation with the definition of new ones. Conventional flow chart notations used in computer programming lack in rigor and universality, since they often rely on *ad hoc* devices, or on devices tailored too closely to the characteristics of specific real machines. The notation adopted for this paper is a simplified version of a new notation recently devised by Iverson [10] which shows great promise of lucidity, precision, and universality. A detailed definition of the notation and of its varied applications is given by Iverson and Brooks [11].

BWW use juxtaposition in the syntax language to denote juxtaposition in the object language. It proves more convenient for our purposes to use a "vector" notation, for example "$[(, \delta'_{1j}, \Delta^1_i]$" instead of "$(\delta'_{1j}\Delta^1_i$" or "$[(, \sim, (, (, x_1, +, x_2,), \cdot, x_3,),)]$" for "$(\sim((x_1 + x_2) \cdot x_3))$". The dimension of a vector is equivalent to length in the sense of BWW. Thus, if $y = [(, \delta_{01}, \delta'_{11}, \delta_{02},)]$, $L(y) = 5$, and if $y = [(, \Delta^3_1, \delta'_{11}, \Delta^3_2,)]$, $L(y) = 3 + L(\Delta^3_1) + L(\Delta^3_2)$. No distinction is made between a vector of dimension 1 and a scalar.

Vectors whose components are taken from the set $\{0, 1\}$ are called *logical* vectors. If, in the logical vector $[a_1, a_2, \cdots, a_n]$, the first k components $= 1$ and the remaining components $= 0$, the vector is designated by "h^k"; if the last k components $= 1$ and the remaining components $= 0$, the vector is designated by "t^k"; if only the kth component $= 1$, the vector is designated by "e^k"; finally, if all components $= 1$, this "unit" vector is designated by "e" without any superscript.

The following operations, among those defined by Iverson, will be used in this paper:

DEFINITION 8. (a) *Scalar multiplication.*

If c is a scalar, and w is a logical vector, then $(cw)_i = cw_i$. For example, if $c = \delta_{01}$ and $w = [0, 1, 1, 0, 1, 1]$, then $cw = [0, \delta_{01}, \delta_{01}, 0, \delta_{01}, \delta_{01}]$.

(b) *Compression.*

If w is an arbitrary vector, and u a logical vector of the same dimension as w, then the result u/w of a compression of w by u is a vector whose sole components are the components w_i of w for which $u_i = 1$. For example, if $w = [(, \delta_{01}, \delta'_{21}, \delta_{02},)]$ and $u = [1, 0, 0, 0, 1]$, $u/w = [(,)]$.

(c) *Complementation.*

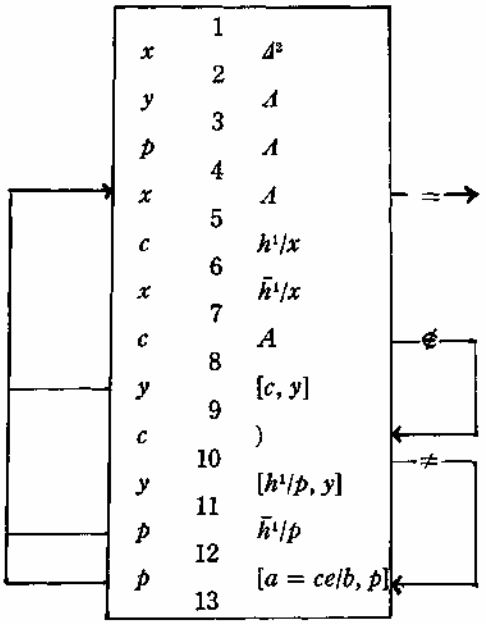
If u is a logical vector, then $v = \bar{u}$ is the complement of u if $v_i = 0$ when $u_i = 1$, and vice versa. In the example of (b), we have $\bar{u} = [0, 1, 1, 1, 0]$, hence $\bar{u}/w = [\delta_{01}, \delta'_{21}, \delta_{02}]$. Note that if $h^1 = [1, 0, 0, 0, 0]$ then $h^1/w = [(]$, and $\bar{h}^1/w = t^1/w = t^{L(w)-1}/w = [\delta_{01}, \delta'_{21}, \delta_{02})]$.

(d) *Reduction.*

If x and y are arbitrary vectors of equal dimension, and $\mathscr{R}$ is a binary relation defined on their components, the reduction $x\mathscr{R}y$ is a logical vector u of the same dimension such that $u_i = 1$ if $x_i \mathscr{R} y_i$ holds and $u_i = 0$ otherwise. For example, let $x = [1, 2, 3, 4, 5]$, $y = [5, 4, 3, 2, 1]$, and $\mathscr{R} = >$. Then $x > y = [0, 0, 0, 1, 1]$.

EXAMPLE 3. Let $a = [(, (,), \delta_{01}, \delta'_{1j}, \delta'_{1k}, \delta'_{2k}, \delta'_{2k}]$ and $b = [s, v, u, u, t, \delta_{1j}, t, \delta_{2k}]$. Then if $c = \delta'_{1j}$ and e is the unit vector we have by reduction, $(a = ce) = [0, 0, 0, 0, 1, 1, 0, 0]$, and by a subsequent compression we obtain $(a = ce)/b = [t, \delta_{1j}]$. Similarly, if $c =)$, $(a = ce)/b = u$.

In Iverson's notation, as in conventional computer programming, an algorithm is specified by a sequence of statements, as illustrated in Figure 1, where a formal expression of Definition 6 is shown. The expression "$x\ \Delta^2$", for example,



S_2

$A = \{\delta_{0i}\}$

$a = [\delta'_{11}, \cdots, \delta'_{1j}, \delta'_{21}, \cdots, \delta'_{2k}]$

$b = [\delta_{11}, \cdots, \delta_{1j}, \delta_{21}, \cdots, \delta_{2k}]$

Translation from P_2 to L

Figure 1

may be read as "x is specified to be Δ^2", and "y $[c, y]$" is equivalent to "y is specified to be $[c, y]$" or to "the new y is the result of juxtaposing c to the left of the old y". Equivalent expressions in many common flow chart notations would be "$\Delta^2 \rightarrow y$" and "$[c, y] \rightarrow y$", but arrows are superfluous within elementary statements.

The diagram of Figure 1 is normally read from top to bottom, and the order of execution of the statements is normally the order of listing. A line directed outward from a statement indicates a branch point, namely a break in the normal sequence. If the line is unlabeled, the statement next to it is read according to the definition of the preceding paragraph, but the next statement to be executed is never the one listed next, but always that to which the directed line leads. The unlabeled line thus marks an unconditional branch. For example, "x A" and not "c)" follows "y $[c, y]$". If the directed line is labeled, the statement next to it is interpreted as a *comparison*, not as a specification, and the branch is a conditional one. Thus, "c A" is equivalent to "$c \in A$?" or to "does c belong to the set A?", and the mark "$\notin$" on the line indicates that "c A" is followed by "c)" if "$c \notin A$ and by "y $[c, y]$" if $c \in A$. The reader may now easily verify, with the aid of Definition 8, that the diagram of Figure 1 is indeed a precise version of Definition 6.

A notation for paths through a diagram will be useful. For purposes of reference to statements, each statement is associated with the number immediately above it. For purposes of reference to paths, each number is associated with the *interval* in which the numeral appears. The expression "(m, n)" designates any path starting at interval m and terminating at interval n, without restriction as to allowable intermediate subpaths. For example "$(1, 4)$" designates a path encompassing the first three statements of the algorithm S_2 in Figure 1, and "$(4, 4)$" designates any nontrivial path starting at interval 4 and terminating at the same interval after one or more cycles with arbitrary subpaths.

At a branch point, two intervals are coalesced into one. The expression "r/s" indicates that interval r is to be treated as equivalent to the interval s. The expression "$(4, 9/4)$" designates a path starting at 4 and returning via the unconditional branch which identifies interval 9 with interval 4. The path $(4, 8/9, 12/4)$ is one taken when $c =$), while $(4, 8/9, 10/12, 13/4)$ is followed when c is a functor. No confusion results if the latter path is simply labeled "$(4, 13/4)$".

The expressions "$n : y = a$" or "$(m, n) : y = a$" indicate that at interval n (possibly following a prior passage through interval m) y has the value a before the statement following the interval is executed. Thus, in S_2, $(1, 4) : x = \Delta^2, y = \Lambda$, and $p = \Lambda$.

4. Δ_M-theorems; fail-safe translation. An algorithm T_3, for translation from L to P_3, and an algorithm S_3, for translation from P_3 to L, are given in §§5 and 6, respectively. These algorithms have the following interesting properties:

(1) Internal storage consists essentially of a single pushdown store.

(2) The input formula is scanned in one direction only.

Each character in the input formula is used once and only once and in sequence, eliminating the need either for storing the input formula in internal memory or else for repeated backward and forward reading of tapes, as is required in Rutishauser's method and its derivatives.

(3) The amount of internal storage is independent of the length of the input formula, and depends only on the depth of the deepest nest in the formula.

(4) The characters of the output formula are generated practically simultaneously with the scanning of the input characters, so that translation is completed almost as soon as the last input symbol is read. In translation from P_3 to L, the characters of the output formula are generated in proper sequence either for immediate evaluation by a pushdown evaluator in the manner of BWW or for further translation into a computer program by a device quite analogous to the BWW pushdown evaluator, which is essentially what Samelson and Bauer propose.

(5) It will be shown that each algorithm operates successfully if and only if the input formula is well-formed. The algorithms are therefore ideally fail-safe.

(6) For each algorithm, a theorem of the following type (Δ_M-theorem) can be proved:

Let $\Delta = [\Delta_H, \Delta_M, \Delta_T]$ be any formula of the domain of translation, split into a head Δ_H, a middle Δ_M, and a tail Δ_T. The formula Δ_M is assumed to be well-formed in the domain, while Δ_H and Δ_T are arbitrary residues, possibly null, determined by the choice of Δ_M. At a certain point in the execution of the algorithm, the remaining input formula will be $[\Delta_M, \Delta_T]$, an image Δ'_H of Δ_H will have been generated, and p (as defined in Definition 6 or Definition 7) will be $p(\Delta_H)$, namely a function of Δ_H only. While the characters of Δ_M are being scanned, p naturally becomes a function of Δ_M as well as of Δ_H, but all contributions to p due to Δ_M will be "above" those due to Δ_H in the pushdown store. The Δ_M-theorem effectively guarantees that, once the remaining input formula has become Δ_T,

(a) p will again be $p(\Delta_H)$, that is, no contributions due to Δ_M remain at the top of the pushdown store; and

(b) the image Δ'_M of Δ_M, which is well-formed in the range of translation, will have been adjoined to Δ'_H.

5. Translation from L to P_3. An algorithm T_3 for translating from L to P_3 is given in Figure 2. The properties of this algorithm are presented in a series of theorems. In the following, all paths are paths in T_3.

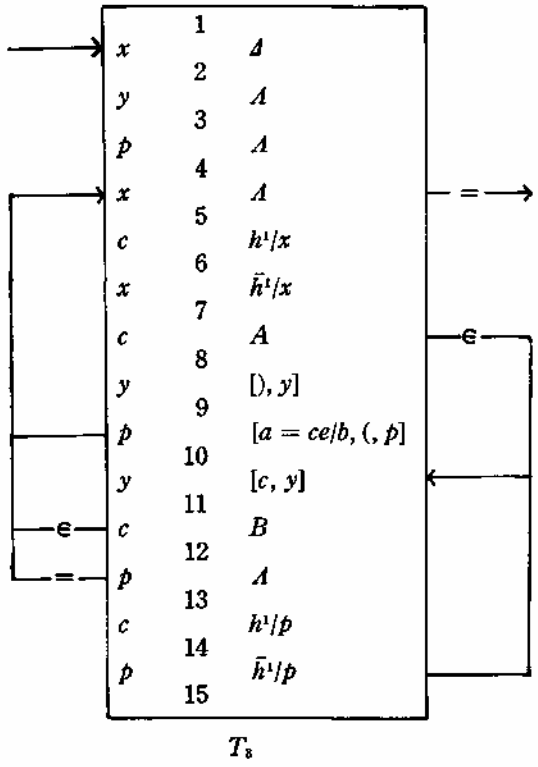
DEFINITION 9. Any path (4, 13/4) such that at no time $4 : x = \Lambda$ and $p \neq \Lambda$, is called a *formula cycle* of T_3.

DEFINITION 10. T_3 is *strictly effective* for a formula Δ of L if and only if:

(a) $\Delta = \Lambda$; or

(b) for $\Delta \neq \Lambda$, a formula cycle of T_3 is traversed once and only once to obtain $(4, 13/4) : x = \Lambda$.

DEFINITION 11. T_3 is *effective* for a formula Δ of L if and only if, for $\Delta \neq \Lambda$, a formula cycle of T_3 is traversed a finite number of time to obtain



T_3

$A = \{\delta_{01}\}$

$B = \{\delta'_{2k}\}$

$a = [\delta_{11}, \cdots, \delta_{1j}, \delta_{21}, \cdots, \delta_{2k}]$

$b = [\delta'_{11}, \cdots, \delta'_{1j}, \delta'_{21}, \cdots, \delta'_{2k}]$

Translation from L to P_3
Figure 2

$(4, 13/4) : x = \Lambda$.

DEFINITION 12. When $4 : x = \Lambda, 4 : y = T_3\Delta = \Delta^3$ is the *T_3-image* of Δ in P_3.

LEMMA 1. (a) $(1, 4) : x = \Delta, y = \Lambda, p = \Lambda$,
(b) $(4, 13/4) : p = \Lambda$.

The proof is obvious.

LEMMA 2. *$T_3\Delta = \Lambda$ and T_3 is strictly effective or effective for Δ if and only if $\Delta = \Lambda$.*

PROOF. Obviously if $\Delta = \Lambda$, $T_3\Delta = \Lambda$, and T_3 is strictly effective. If $T_3\Delta = \Lambda$ and T_3 is strictly effective or effective for Δ, but $\Delta \neq \Lambda$, either Δ contains an element of A, hence when $4 : x = \Lambda, y \neq \Lambda$, or it contains no element of A, hence when $4 : x = \Lambda, p \neq \Lambda$, hence T_3 can be neither effective nor strictly effective. Therefore $\Delta = \Lambda$.

The definition of well-formation given by BWW (Definition 3F) is shown to be equivalent to Definition 4 by the following lemma:

LEMMA 3. *$\Delta \neq \Lambda$ is a well-formed formula if and only if it is of the form $\Delta = [\delta, \Delta_{D(\delta)}, \cdots, \Delta_1]$, where each Δ_j is well-formed.*

PROOF. (a) If $\Delta = [\delta, \Delta_{D(\delta)}, \cdots, \Delta_1]$, then either

(i) $D(\delta) = 0$, hence $W(\delta) = 1$, and $\Delta = \delta$ is positive and well-formed; or

(ii) $D(\delta) > 0$; then $\Phi = [\Delta_{D(\delta)}, \cdots, \Delta_1]$ is a positive formula (BWW, Theorem IA) and $W(\Phi) = D(\delta) > 0$. Since $W(\delta) = 1 - D(\delta)$, we have $W(\Delta) = W(\delta) + W(\Phi) = 1 - D(\delta) + D(\delta) = 1$. Hence Δ is positive and of weight 1, that is, well-formed.

(b) If Δ is well-formed, then $\Delta = [\delta, \Phi]$, where either $\Phi = \Lambda$ or Φ, being the tail of a positive formula, is positive. If $\Phi = \Lambda$, the result follows at once that $\Delta = \delta$, and $D(\delta) = 1 - W(\delta) = 0$. If $\Phi \neq \Lambda$, then since $1 = W(\Delta) = W(\delta) + W(\Phi) = 1 - D(\delta) + W(\Phi)$, we have $W(\Phi) = D(\delta)$, and by Theorem IA (BWW) $\Phi = [\Phi_{D(\delta)}, \cdots, \Phi_1]$ whence the conclusion follows at once that Δ is of the form $[\delta, \Delta_{D(\delta)}, \cdots, \Delta_1]$.

LEMMA 4. *Every well-formed formula can be written in the form $\Delta = [\Delta_H, \Delta_M, \Delta_T]$, where Δ_M is some well-formed formula embedded in Δ, and where Δ_H and Δ_T are respectively the residual (possibly null) head and tail of Δ.*

PROOF. Note that any Δ_j of Lemma 3 can serve as Δ_M and that, since Δ_j itself has a decomposition $\Delta_j = [\delta, \Delta_{jD(\delta)}, \cdots, \Delta_{j1}]$ any Δ_{jk} likewise, and so on.

LEMMA 5. *For arbitrary $\Delta = [\Delta_H, \Delta_M, \Delta_T]$, there exists a path* (1, 4) *such that*

$$(1,4): x = [\Delta_M, \Delta_T],$$
$$y = y_I(\Delta_H) = y_I,$$
$$p = p_I(\Delta_H) = p_I.$$

PROOF. Clearly $(1,4): x = [\Delta_M, \Delta_T]$ sometime. Consider a path leading to 4 at this time. Since $(4,6): c = h^1/x = h^1/\Delta_M$ marks the first appearance of a character of Δ not belonging to Δ_H, $(1,4): y = y(\Delta_H), p = p(\Delta_H)$.

THEOREM 1 (Δ_M-THEOREM). *For arbitrary $\Delta = [\Delta_H, \Delta_M, \Delta_T] \neq \Lambda$, where Δ_M is well-formed, there exists a path* (4, 13) *starting at*

$$4: x_s = [\Delta_M, \Delta_T],$$
$$y_s = y_I,$$
$$p_s = p_I,$$

such that

(A) $13: x_f = \Delta_T,$
$$y_f = [y_f(\Delta_M), y_I],$$
$$p_f = p_I;$$

(B) *$y_f(\Delta_M) = T_s\Delta_M = \Delta_M^3$ and is unique, and T_s is strictly effective for Δ_M.*

The proof is by induction on the length $L(\Delta_M) > 0$ of Δ_M.

(a) $L(\Delta_M) = 1$. Since Δ_M is well-formed by hypothesis, $\Delta_M = \delta_{0t}$. The path (4, 8/10, 13) is followed, and

(A) $13: x_f = \bar{h}^1/x_s = \Delta_T$,
$y_f = [c, y_I] = [h^1/x_s, y_I] = [\delta_{0t}, y_I]$,
$p_f = p_I$;

(B) Let $\Delta_H = \Delta_T = \Lambda$: then $y_I = p_I = \Lambda$ (Lemma 1), and

$$13: x_f = \Lambda,$$
$$y_f = \delta_{0t} = y(\Delta_M),$$
$$p_f = \Lambda;$$

hence the path continues to 4. Since $4: x = \Lambda, y_f(\Delta_M) = \delta_{0t} = \Delta_M^3$, and is unique by (A). Since 4: $p = \Lambda$, (4, 8/10, 13/4) is a formula cycle traversed once and only once, and T_3 is strictly effective for Δ_M.

(b) $L(\Delta_M) > 1$.

Case I. $\Delta_M = [\delta_{1j}, \Delta_1]$.

(A) The path (4, 10/4) is followed and

$$(4, 10/4): x_a = [\Delta_1, \Delta_T],$$
$$y_a = [), y_I],$$
$$p_a = [\delta'_{1j}, (, p_I].$$

Applying the induction hypothesis to Δ_1, we obtain a path (4, 10/4, 13) such that

$$(4, 13): x_b = \Delta_T,$$
$$y_b = [\Delta_1^3,), y_I],$$
$$p_b = p_a.$$

Since $p_b = [\delta'_{1j}, (, p_I] \neq \Lambda$, the path (13, 15/10, 13) is traversed twice yielding

$$(4, 13): x_f = \Delta_T,$$
$$y_f = [(, \delta'_{1j}, \Delta_1^3,), y_I],$$
$$p_f = p_I.$$

(B) Let $\Delta_H = \Delta_T = \Lambda$; then $y_I = p_I = \Lambda$ (Lemma 1), and

$$13: x_f = \Lambda,$$
$$y_f = [(, \delta'_{1j}, \Delta_1^3,)] = y_f(\Delta_M),$$
$$p_f = \Lambda;$$

hence the path continues to 4.

Since $4: x = \Lambda, y_f(\Delta_M) = [(, \delta'_{1j}, \Delta_1^3,)] = \Delta_M^3$, and is unique by (A). Since $4: p = \Lambda$, (4, 13/4) is a formula cycle traversed once and only once, and T_3 is strictly effective for Δ_M.

Case II. $\Delta_M = [\delta_{2k}, \Delta_2, \Delta_1]$.

(A) The path (4, 10/4) is followed and

$$(4, 10/4): x_a = [\Delta_2, \Delta_1, \Delta_T],$$
$$y_a = [), y_I],$$
$$p_a = [\delta'_{2k}, (, p_I].$$

Applying the induction hypothesis to Δ_2, we obtain a path (4, 13) such that

$$(4, 13): x_b = [\Delta_1, \Delta_T],$$
$$y_b = [\Delta^3_2,), y_I],$$
$$p_b = [\delta'_{2k}, (, p_I].$$

The path (13, 15/10, 12/4) is followed next, yielding

$$(4, 4): x_c = [\Delta_1, \Delta_T],$$
$$y_c = [\delta'_{2k}, \Delta^3_2,), y_I],$$
$$p_c = [(, p_I].$$

Applying the induction hypothesis to Δ_1, we obtain a path (4, 13) such that

$$(4, 13): x_d = \Delta_T,$$
$$y_d = [\Delta^3_1, \delta'_{2k}, \Delta^3_2,), y_I],$$
$$p_d = [(, p_I].$$

Finally, the path (13, 15/10, 13) is followed, yielding

$$(4, 13): x_f = \Delta_T,$$
$$y_f = [(, \Delta^3_1, \delta'_{2k}, \Delta^3_2,), y_I],$$
$$p_f = p_I.$$

(B) Proof parallel to that for Case I.

THEOREM 2. *For every* POSITIVE $\Delta_M = [\Delta_{W(\Delta M)}, \cdots, \Delta_1] \neq \Lambda$ (*BWW, Theorem* 1) *there exists a path* (4, 13/4) *starting at*

$$4: x_s = [\Delta_M, \Delta_T],$$
$$y_s = y_I,$$
$$p_s = p_I,$$

such that if $p_I = \Lambda$, *then*

(A) $$(4, 13/4): x_f = \Delta_T,$$
$$y_f = [y_f(\Delta_M), y_I],$$
$$p_f = p_I = \Lambda,$$

(B) $y_f(\Delta_M) = T_3\Delta_M = [\Delta^3_1, \cdots, \Delta^3_{W(\Delta M)}] = \Delta^3_M$ *and is unique,* T_3 *is effective for* Δ_M, *and the formula cycle is traversed* $W(\Delta_M)$ *times.*

The proof is by induction on $W(\Delta_M)$.

Applying Theorem 1 to $\Delta_{W(\Delta_M)}$ we obtain

$$(4, 13): x_a = [\Delta_{T'}, \Delta_T], \Delta_{T'} = [\Delta_{W(\Delta_M)-1}, \cdots, \Delta_1],$$
$$y_a = [\Delta^3_{W(\Delta_M)}, y_I],$$
$$p_a = p_I = \Lambda.$$

Since $p_a = \Lambda$, the path may be continued to (4, 13/4) with the same values of x, y, and $\check{p}$, and constitutes a formula cycle, the first traversed. Since T_3 is strictly effective for $\Delta_{W(\Delta_M)}$ (Theorem 1) it is also effective (Definitions 10 and 11) and $\Delta^3_{W(\Delta_M)}$ is unique (Theorem 1).

If $W(\Delta_M) = 1$, $\Delta_{W(\Delta_M)} = \Delta_M$, $\Delta_{T'} = \Lambda$, hence these results provide the basis for the induction. If $W(\Delta_M) > 1$, the same results give the first part of the induction step.

Applying the induction hypothesis to $\Delta_{T'}$, $W(\Delta_{T'}) = W(\Delta_M) - 1$, we obtain at once

(A) $4 : x_f = \Delta_T$,

$$y_f = [\Delta^3_{T'}, \Delta^3_{W(\Delta_M)}, y_I] = [\Delta^3_1, \cdots, \Delta^3_{W(\Delta_M)-1}, \Delta^3_{W(\Delta_M)}, y_I],$$

$$p_f = p_I = \Lambda;$$

(B) $y_f(\Delta_M)$ clearly has the requisite form and is unique, T_3 is effective for Δ_M, and the formula cycle has been traversed $1 + W(\Delta_{T'}) = W(\Delta_M)$ times.

THEOREM 3. *If $\Delta = [\Delta_{W(\Delta)}, \cdots, \Delta_1] \neq \Lambda$ is positive (BWW Theorem 1), then T_3 is effective, $T_3\Delta = [T_3\Delta_1, \cdots, T_3\Delta_{W(\Delta)}] = [\Delta^3_1, \cdots, \Delta^3_{W(\Delta)}] = \Delta^3$ is the unique T_3-image of Δ, and the formula cycle is traversed precisely $W(\Delta)$ times.*

PROOF. Let $\Delta = [\Lambda, \Delta, \Lambda]$, and the proof follows directly from Lemma 1 and Theorem 2.

THEOREM 4. *If $\Delta \neq \Lambda$ is well-formed, then T_3 is strictly effective and $T_3\Delta = \Delta^3$ is the unique T_3-image of Δ.*

PROOF. This result follows directly from Definitions 10 and 11 and the specialization of Theorem 3 to the case $W(\Delta) = 1$.

THEOREM 5. *T_3 is effective only if $\Delta \neq \Lambda$ is positive.*

PROOF. If Δ is not positive, then for some integer $i \leqq L(\Delta)$ there exists a $\Phi = t^i/\Delta$ such that $W(\Phi) < 1$. Let $j + 1$ be the smallest such integer. Then $\psi = t^j/\Delta$ is either null or positive, and $e^{j+1}/\Delta = \delta$ is such that $-1 \leqq W(\delta) < 1$, i.e., $\delta = \delta_{1j}$ or $\delta = \delta_{2k}$. Let $\Delta = [\Delta_H, \delta, \psi] = [\Delta_H, \Delta_M, \Delta_T]$, where $\Delta_H = h^{L(\Delta)-j-1}/\Delta$.

With the aid of Lemma 5 we obtain, at some time

$$4 : x_a = [\delta, \psi],$$
$$y_a = y_a(\Delta_H),$$
$$p_a = p_a(\Delta_H).$$

Since $\delta \notin A$, there is a path (4, 10/4) such that

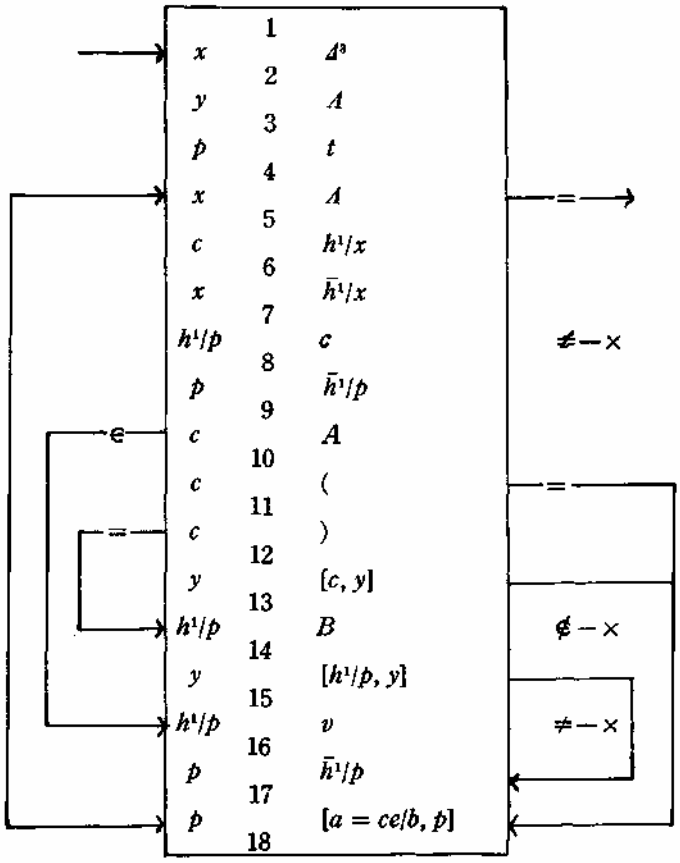
$$(4, 10/4) : x_s = \psi,$$
$$y_s = [\,), y_a],$$
$$p_s = [\delta', (, p_a].$$

If $\psi = \Lambda$, then, since $p \neq \Lambda$, no formula cycle is possible and T_3 cannot be effective. If $\psi \neq \Lambda$, $W(\psi) = 1$ since for $W(\psi) > 1$ and $W(\Phi) = W(\delta) + W(\psi) < 1$ we would require $W(\delta) < -1$ which is impossible in L as defined. Therefore

ψ is well-formed. In this case, $\delta = \delta_{2k}$, since if $\delta = \delta_{1k}$ we would have $W(\emptyset) = W(\delta_{1k}) + W(\psi) = 0 + 1 = 1$ contrary to hypothesis. Applying Theorem 1 to $\psi = [\psi, A]$

$$13: x_f = A\,,$$
$$y_f = [\psi^s,), y_a]\,,$$
$$p_f = [\delta'_{2k}, (, p_a]\,.$$

The path (13, 15/10, 12/4) is followed next, yielding



S_3

$A = \{\delta'_{1j}, \delta'_{2k}\}$
$B = \{\delta_{1j}, \delta_{2k}\}$
$a = [(, (,), \delta_{04}, \delta'_{1j}, \delta'_{1j}, \delta'_{2k}, \delta'_{2k}]$
$b = [s, v, u, u, t, \delta_{1j}, t, \delta_{1k}]$

Translation from P_3 to L
Figure 3

$$4: x = \Lambda ,$$
$$y = [\delta'_{2k}, \psi^3,), y_a] ,$$
$$p = [(, p_a] \neq \Lambda .$$

No formula cycle is possible, hence T_3 cannot be effective, and the proof is complete.

THEOREM 6. *T_3 is effective if and only if $\Delta \neq \Lambda$ is positive, and strictly effective if and only if Δ is well-formed or $\Delta = \Lambda$.*

PROOF. Theorems 3 and 5 together account for the case of positive Δ. By Theorem 5, T_3 is effective only if $\Delta \neq \Lambda$ is positive, but the number of traversals of the formula cycle will be precisely 1 only if Δ is also well-formed. Hence T_3 is strictly effective only if Δ is well-formed; this result, together with Theorem 4, accounts for the case of well-formed Δ. The case of $\Delta = \Lambda$ is covered by Definition 10.

6. Translation from P_3 to L. An algorithm S_3 for translating from P_3 to L is given in Figure 3. It is characteristic of L that there are no restrictions on the order in which characters may appear in well-formed formulas, a property reflected in the absence of any test for order in T_3 and in the dependence of the proof of Theorem 5 on arguments based on tail weight only. The situation is different in S_3. Figure 4 shows that occurrence of a character of a given type influences the type of character that may occur next, either immediately or after intervening characters. The compatibility of adjacent characters is tested by the statement on line 7 of S_3, where "$\doteqdot$" is to be interpreted as "is compatible with". This feature of S_3 is analogous to the pair test of Perlis (Carr, [4, pp. 2–223]). The conditions under which $h'p \doteqdot c$ are given in Figure 5. Compatibility across intervening characters is tested by the statements on lines 13 and 15. For the elementary language P_3 at least, these tests, analogous to the use of predictions in the syntactic analysis of Russian or English, achieve in a very simple manner what Carr attempts to do with production trees (Carr, [4, pp. 2–233ff]). While the mode of operation of the tests is accounted for in the theorems that follow, the reader may find it illuminating at this point to get an intuitive feeling for their operation by applying S_3 to a few short formulas, both well-formed and not.

Character	Predictors	Predicts
(	$[s, v]$	$(, \delta_{0i}, \delta'_{1j}$
δ'_{1j}	$[t, \delta_{1j}]$	$(, \delta_{0i}$
δ'_{2k}	$[t, \delta_{2k}]$	$(, \delta_{0i}$
δ_{0i}	u	$\delta'_{2k},)$
)	u	$\delta'_{2k},)$

Predictions
Figure 4

Character	Accepted by
(	s, t
δ'_{1j}	s
δ'_{2k}	u
δ_{0i}	s, t
)	u

Compatibility Conditions
Figure 5

DEFINITION 13. S_3 is *strictly effective* for a formula Δ^3 of P_3 if and only if a) $\Delta^3 = \Lambda$; or (b) for $\Delta^3 \neq \Lambda$

(i) at no time $4: x = \Lambda$ and $p \neq u$; and
(ii) at no time $7: c \neq h^1/p$,
(iii) at no time $13: h^1/p \notin B$,
(iv) at no time $15: h^1/p \neq v$.

A path in which all the conditions of (b) are met is an *effective* path; otherwise it is *ineffective*.

DEFINITION 14. When $4: x = \Lambda, 4: y = S_3\Delta^3 = \Delta$ is the S_3-image of Δ^3 in L.

LEMMA 6. $(1, 4): x = \Delta, y = \Lambda, p = t$.

The proof is obvious.

LEMMA 7. *$S_3\Delta^3 = \Lambda$ and S_3 is strictly effective for Δ^3 if and only if $\Delta^3 = \Lambda$.*

PROOF. Obviously, if $\Delta^3 = \Lambda$, $S_3\Delta^3 = \Lambda$ and S_3 is strictly effective. If $S_3\Delta^3 = \Lambda$ and S_3 is strictly effective but $\Delta^3 \neq \Lambda$, then either Δ^3 contains a variable, hence when $4: x = \Lambda, y \neq \Lambda$, or it contains no variable and ends:

(i) with a right parenthesis; then either $13: h^1/p = \Lambda \notin B$ and S_3 is not strictly effective, or $13: h^1/p \neq \Lambda$, hence when $4: x = \Lambda, y \neq \Lambda$; or

(ii) with a left parenthesis or a functor; then, when $4: x = \Lambda, p \neq u$ and S_3 is not strictly effective.

Therefore $\Delta^3 = \Lambda$.

LEMMA 8. *Every well-formed formula can be written in the form $\Delta^3 = [\Delta^3_H, \Delta^3_M, \Delta^3_T]$, where Δ_M is some well-formed formula nested in Δ^3, and where Δ^3_H and Δ^3_T are respectively the residual (possibly null) head and tail of Δ^3.*

The proof is a direct consequence of the definition of well-formation in P_3 (Definition 3).

LEMMA 9. *For every $\Delta^3 = [\Delta^3_H, \Delta^3_M, \Delta^3_T] \neq \Lambda$, there exists a path* $(1, 4, 18/4)$ such that

$$4: x = [\Delta^3_M, \Delta^3_T],$$
$$y = y_I(\Delta^3_H) = y_I,$$
$$p = \begin{bmatrix} s \\ t, p_I \\ u \end{bmatrix}, p_I = \bar{h}^1/p,$$

where $p_I = p_I(\Delta_H)$ either $= \Lambda$ or contains only instances of "v" or of members of B.

PROOF. Showing that x and y have the stated properties and that $p_I = p_I(\Delta^3_H)$ is trivial (see Lemma 5). The remainder may be proved by induction on $L(\Delta_H)$.

If $L(\Delta^3_H) = 1$ we have

$$(1, 4) : p = t, (4, 9) : p = \Lambda .$$

A path (16, 17) cannot add characters to p. Eventually, therefore, the path (1, 4, 9, 18/4) is followed resulting in $4 : h^1/p = s, u,$ or $t,$ and $4 : \bar{h}^1/p = v, \delta_{1j}, \delta_{2k},$ or $\Lambda,$ as a consequence of the definition of the vectors a and b (Figure 3).

For $L(\Delta^3_H) > 1$, applying the induction hypothesis to $\bar{t}^1/\Delta^3_H$ yields

$$4 : p = \begin{bmatrix} s \\ t, p_I \\ u \end{bmatrix},$$

where p_I has the desired properties. Applying an argument similar to that for $L(\Delta^3_H) = 1$ to the remainder t^1/Δ^3_H of Δ^3_H is then sufficient to complete the proof.

DEFINITION 15. If, in Lemma 9, $p = \begin{bmatrix} s \\ t \end{bmatrix}, p_I]$, Δ_M is *properly nested.*

THEOREM 7 (Δ^3_M-THEOREM). *For every $\Delta^3 = [\Delta^3_H, \Delta^3_M, \Delta^3_T]$, where Δ^3_M is well-formed and properly nested, there exists a path* (4, 18/4) *starting at*

$$4 : x_s = [\Delta^3_M, \Delta^3_T] ,$$
$$y_s = y_I ,$$
$$p_s = \begin{bmatrix} s \\ t \end{bmatrix}, p_I] ,$$

such that

(A)
$$(4, 18/4) : x_f = \Delta^3_T ,$$
$$y_f = [y_f(\Delta^3_M), y_I] ,$$
$$p_f = [u, p_I] ,$$

(B)
$$y_f(\Delta^3_M) = S_3\Delta^3_M = \Delta_M$$

and is unique, and S_3 is strictly effective for Δ^3_M.

COROLLARY. *If, in Theorem 7, Δ^3_M is not properly nested, S_3 is not strictly effective for Δ^3_M.*

The proof of the corollary follows directly from Definitions 13 and 15, and the compatibility conditions (Figure 5). If $p = [u, p_I]$, then since every well-formed formula begins either with (or with δ_{0i}, it follows at once that $7 : c \neq h^1/p$, hence S_3 cannot be strictly effective.

The proof of Theorem 7 is by induction on the length $L(\Delta^3_M)$ of Δ^3_M.

(a) $L(\Delta^3_M) = 1$. Since Δ^3_M is well-formed by hypothesis, $\Delta^3_M = \delta_{0i}$. The path (4, 13/17, 18/4) is followed; hence we have

(A)
$$4 : x_f = \Delta^3_T ,$$
$$y_f = [\delta_{0i}, y_I] ,$$
$$p_f = [u, p_I] .$$

(B) Let $\Delta_H = \Delta_T = \Lambda$. Then $y_I = \Lambda$ and $p_s = t$ (Lemma 6), hence $p_s = [t, \Lambda]$,

that is, $p_I = \Lambda$. We have, therefore

$$4: x_f = \Lambda ,$$
$$y_f = \delta_{0i} ,$$
$$p_f = u .$$

Clearly $y_f(\Delta_M) = \delta_{0i}$ is the S_3-image of Δ_H^3 and is unique. Since $4: x = \Lambda$ and $p = u$, and since, in (A), $7: c \fallingdotseq h^1/p$ and there are no subpaths (13, 14) or (15, 16), S_3 is strictly effetive for Δ_M^3.

(b) $L(\Delta_M^3) > 1$.

CASE I. $\Delta_M^3 = [(, \delta_{1j}', \Delta_1^3,)]$.

(A) The path (4, 7, 11/17, 18/4) yields

$$4: x_a = [\delta_{1j}', \Delta_1^3,), \Delta_T^3] ,$$
$$y_a = y_I ,$$
$$p_a = [s, v, p_I] .$$

We note that 7: $c \fallingdotseq h^1/p = s$ or t. Continuing with the path (4, 7, 10/15, 18/4), we obtain

$$4: x_b = [\Delta_1^3,), \Delta_T^3] ,$$
$$y_b = y_I ,$$
$$p_b = [t, \delta_{1j}, p_I] ,$$

noting that $7: c \fallingdotseq h^1/p = s$, and $15: h^1/p = v$. Now, applying the induction hypothesis to Δ_1^3, we obtain

$$4: x_c = [), \Delta_T^3] ,$$
$$y_c = [\Delta_1, y_I] ,$$
$$p_c = [u, \delta_{1j}, p_I] .$$

Finally, the path (4, 7, 12/13, 15/16, 18/4) is followed, yielding

$$4: x_f = \Delta_T^3 ,$$
$$y_f = [\delta_{1j}, \Delta_1, y_I] ,$$
$$p_f = [u, p_I] .$$

(B) Let $\Delta_H = \Delta_T = \Lambda$. Then $y_I = p_I = \Lambda$. It follows that $y_f(\Delta_M^3) = [\delta_{1j}, \Delta_1] = \Delta_M$. The process of (A) shows that Δ_M is unique and that S_3 is strictly effective.

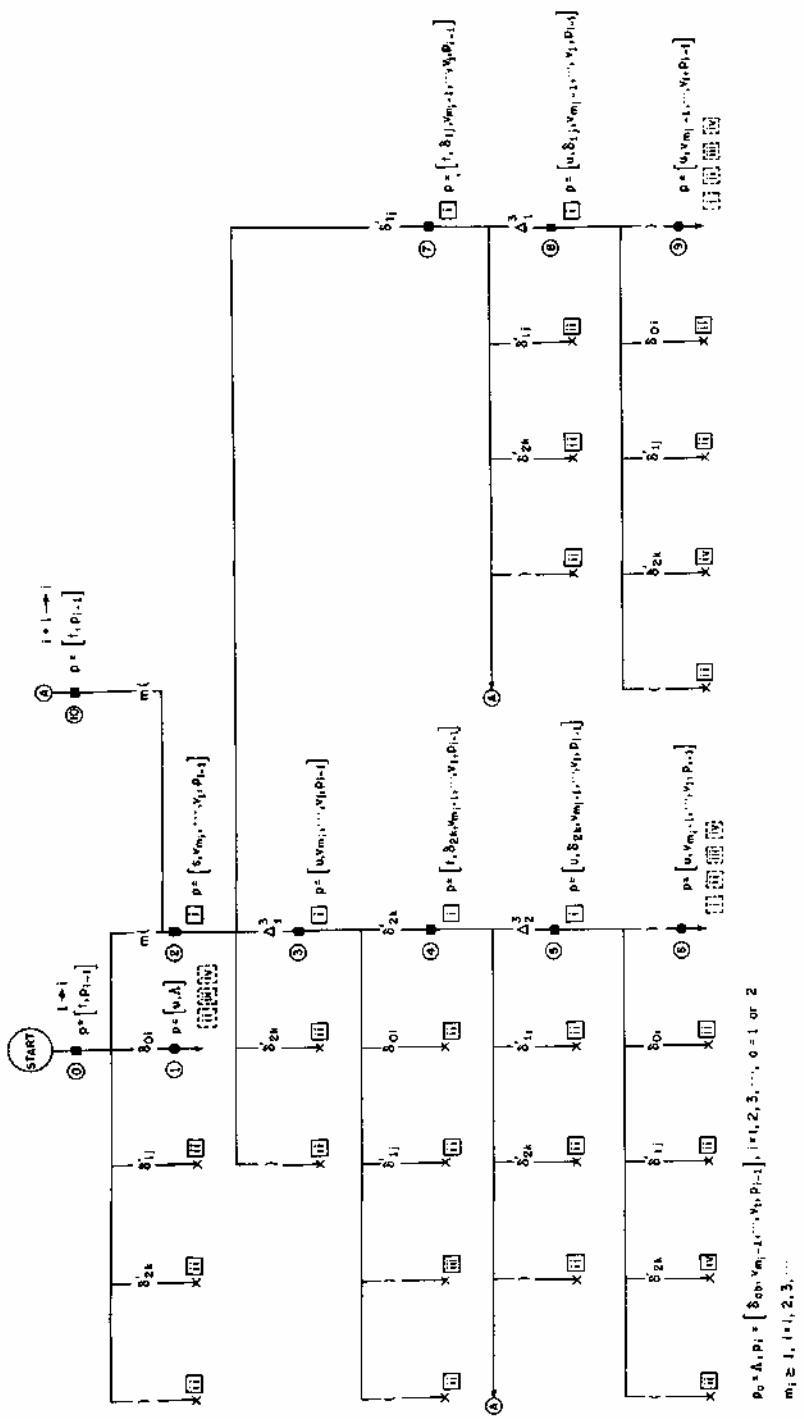
CASE II. $\Delta_M^3 = [(, \Delta_1^3, \delta_{2k}', \Delta_2^3,)]$.

(A) As in Case I, the path (4, 7, 11/17, 18/4) yields

$$4: x_a = [\Delta_1^3, \delta_{2k}', \Delta_2^3,), \Delta_T^3] ,$$
$$y_a = y_I ,$$
$$p_a = [s, v, p_I] ,$$

and 7: $c \fallingdotseq h^1/p = s$ or t. Applying the induction hypothesis to Δ_1^3, we obtain

$$4: x_b = [\delta_{2k}', \Delta_2^3,), \Delta_T^3] ,$$
$$y_b = [\Delta_1, y_I] ,$$
$$p_b = [u, v, p_I] .$$



Effective and Ineffective Paths
Figure 6

The path (4, 7, 10/15, 18/4) then yields

$$4 : x_c = [\Delta_2^3,), \Delta_T^3] ,$$
$$y_c = [\Delta_1 , y_I] ,$$
$$p_c = [t, \delta_{2k} , p_I] ,$$

and $7 : c \doteqdot h^1/p = u$, and $15 : h^1/p = v$. Now, applying the induction hypothesis to Δ_2^3, we obtain

$$4 : x_d = [), \Delta_T^3] ,$$
$$y_d = [\Delta_2 , \Delta_1 , y_I] ,$$
$$p_d = [u, \delta_{2k} , p_I] .$$

Finally, the path (4, 7, 12/13, 15/16, 18/4) is followed, yielding

$$4 : x_f = \Delta_T^3 ,$$
$$y_f = [\delta_{2k} , \Delta_2 , \Delta_1 , y_I] ,$$
$$p_f = [u, p_I] .$$

(B) As in Case I.

THEOREM 8. *If $\Delta^3 \neq \Lambda$ is well-formed, then S_3 is strictly effective, and $S_3\Delta^3 = \Delta$ is the unique S_3-image of Δ^3.*

PROOF. Let $\Delta^3 = [\Lambda, \Delta^3, \Lambda]$, and the proof follows directly from Lemma 6 and Theorem 7.

THEOREM 9. *S_3 is strictly effective only if $\Delta^3 \neq \Lambda$ is well-formed.*

PROOF. The proof is by induction, following a pattern illustrated by Figure 6. The crosses mark the ends of ineffective paths, the Roman numeral in the square box indicating which of the conditions of Definition 13(b) is violated. The heavy squares and dots each mark a return to line 4 of S_3; the circled numerals next to these are for reference. The value $4 : p$ is given in each case, and the Roman numeral in the square box denotes a violation of the conditions of Definition 13(b) in case $4 : x = \Lambda$. Dots are associated with possible terminals of effective paths, while squares denote that a path terminating at the point is ineffective. Connectors are used in the conventional flow chart fashion. The notation "$m($" indicates the presence of m left parentheses in Δ^3, $m \geqq 1$.

(A) $L(\Delta^3) = 1$. Reference to S_3 and to Definition 13 shows that for $L(\Delta^3) = 1$, only $\Delta^3 = \delta_{0i}$ leads to an effective path.

(B) $L(\Delta^3) > 1$.

POINT 0 (Figure 6). Clearly formulas beginning with), δ'_{2k}, or δ_{1j} lead at once to ineffective paths in S_3.

If $h^1/\Delta^3 = \delta_{0i}$, point 1 is reached. If $\Delta^3 = \delta_{0i}$, it is well-formed and $L(\Delta^3) = 1$, a case already covered. If $\Delta^3 = [\delta_{0i} , \Delta_T^3]$, $\Delta_T^3 \neq \Lambda$, then, since $p = [u, \Lambda]$ at point 1, all characters $\delta = h^1/\Delta_T^3$, exeept) and δ'_{2k}, lead to ineffective paths in S_3 by virtue of condition (ii) of Definition 13(b). The characters) and δ'_{2k} in turn lead to ineffective paths by virtue of conditions (iii) and (iv) of Definition 13(b),

respectively. Hence Δ^3 can begin with δ_{0i} and S_3 be strictly effective only if $\Delta^3 = \delta_{0i}$ and therefore is well-formed.

The remaining case is $h^1/\Delta^3 = ($. Here the situation is complicated (a) by the possibility that $m > 1$ initially, namely that several left parentheses occur in succession, and (b) by the possibility that $m > 1$ also after the points leading to the connectors, in which case $i > 1, m_i \geqq 1$.

When $i = 1, m_1 = 1$, it is easy to verify that, if Δ^3 is well-formed, the paths from point 0 to points 9 and 6 correspond respectively to Case I and Case II in the proof of Theorem 7.

(B1) Consider now $i = 1, m_1 = 1, \Delta^3 \neq \Lambda$ arbitrary except $L(\Delta^3) > 1$. Since $h^1/\Delta^3 = ($ by hypothesis, a path leads to point 2, where $p = [s, v]$. Since $L(\Delta^3) > 1$ by hypothesis, we have $\Delta^3 = [(, \Delta^3_T], \Delta^3_T \neq \Lambda$.

Point 2. If $h^1/\Delta^3_T =)$ or δ'_{2k}, there exists no further effective path in S_3. Consideration of the case $h^1/\Delta^3_T = \delta'_{1j}$ will be deferred. If $h^1/\Delta^3_T = ($, and $($ is not the head of a well-formed formula, then $m_1 > 1$, contrary to hypothesis. The only remaining cases are $h^1/\Delta^3_T = \delta_{0i}$ or $h^1/\Delta^3_T = ($, where $($ is the head of a well-formed head of Δ^3_T. In either case, Δ^3_T has a well-formed head Δ^3_1. Since Δ^3_1 is well-formed, Theorem 7 guarantees that there is a path to point 3, where $p = [u, v]$. If $\Delta^3 = [(, \Delta^3_1]$, the corresponding path in S_3 is ineffective, since $p \neq u$. Consider therefore $\Delta^3 = [(, \Delta^3_1, \Delta^3_T], \Delta^3_T \neq \Lambda$.

Point 3. There is a further effective path in S_3 only if $h^1/\Delta^3_T = \delta'_{2k}$. Such a path corresponds to one leading to point 4, where $p = [t, \delta_{2k}]$. If $\Delta^3 = [(, \Delta^3_1, \delta'_{2k}]$, the corresponding path in S_3 is ineffective, since $p \neq u$. Consider therefore $\Delta^3 = [(, \Delta^3_1, \delta'_{2k}, \Delta^3_T], \Delta^3_T \neq \Lambda$.

Point 4. If $h^1/\Delta^3_T =)$, δ'_{2k}, or δ'_{1j}, there exists no further effective path in S_3. If $h^1/\Delta^3_T = ($, and $($ is not the head of a well-formed formula, then $i > 1$, contrary to hypothesis. The only remaining cases are $h^1/\Delta^3_T = \delta_{0i}$ or $h^1/\Delta^3_T = ($, where $($ is the head of a well-formed head of Δ^3_T. In either case, Δ^3_T has a well-formed head Δ^3_1. Since Δ^3_2 is well-formed, Theorem 7 guarantees that there is a path to point 5, where $p = [u, \delta_{2k}]$. If $\Delta^3 = [(, \Delta^3_1, \delta'_{2k}, \Delta^3_2]$, the corresponding path in S_3 is ineffective, since $p \neq u$. Consider therefore $\Delta^3 = [(, \Delta^3_1, \delta'_{2k}, \Delta^3_2, \Delta^3_T], \Delta^3_T \neq \Lambda$.

Point 5. There is a further effective path in S_3 only if $h^1/\Delta^3_T =)$. Such a path corresponds to one leading to point 6, where $p = [u, \Lambda] = u$. If $\Delta^3 = [(, \Delta^3_1, \delta'_{2k}, \Delta^3_2,)]$, the corresponding path in S_3 is effective, but Δ^3 is also clearly well-formed. It remains only to consider $\Delta^3 = [(, \Delta^3_1, \delta'_{2k}, \Delta^3_2,), \Delta^3_T], \Delta^3_T \neq \Lambda$.

Point 6. There is a further effective path in S_3 only if $h^1/\Delta^3_T =)$ or δ'_{2k}. Since $p = [u, \Lambda]$, $($ and δ'_{2k} in turn lead to ineffective paths by virtue of conditions (iii) and (iv) of Definition 13(b), respectively.

The conclusion is that for $i = 1, m_i = 1$, there is an effective path in S_3 only if $\Delta^3 = [(, \Delta^3_1, \delta'_{2k}, \Delta^3_2,)]$, namely only if Δ^3 is well-formed, provided that, at point 2, $h^1/\Delta^3_T \neq \delta'_{1j}$. It remains now to consider the case where $h^1/\Delta^3_T = \delta'_{1j}$ at point 2.

Point 2 $(h^1/\Delta_T = \delta'_{1j})$. Since $h^1/\Delta^3_T = \delta'_{1j}$, a path leads to point 7, where $p = [t, \delta_{1j}]$. If $\Delta^3 = [(, \delta'_{1j}]$, the corresponding path in S_3 is ineffective, since $p \neq u$. Consider therefore $\Delta^3 = [(, \delta'_{1j}, \Delta^3_T], \Delta^3_T \neq \Lambda$.

It can easily be verified that the transitions from point 7 to points 8 and 9

are completely analogous to those from point 4 to points 5 and 6, and that, with obvious substitutions, identical arguments apply. This, together with the result for $L(\Delta^3) = 1$, demonstrates that for $i = 1, m_i = 1, L(\Delta^3) \geqq 1$, there is an effective path in S_3 only if Δ^3 is well-formed.

(B2) Consider now $i > 1, m_j \geqq 1$ for all $j \leqq i, \Delta^3 \neq \Lambda$ arbitrary except $L(\Delta^3) > 1$.

We note that termination of the formula at all points except 6 or 9 never corresponds to an effective path in S_3, since either $h^1/p \neq u$, or, if $h^1/p = u$, then $\bar{h}^1/p \neq \Lambda$, since $m_i \geqq 1$. Furthermore, since the path through S_3 for any given $c = h^1/x$ depends only on p, it is evident from Figure 6 that points 6 and 9 may be treated as equivalent.

Since Δ^3 is finite, it will have a rightmost left parenthesis, which may (a) lead to an ineffective path in S_3 and terminate the process; or (b) increase m_i, for a given i; or (c) increase i and set $m_i = 1$; or (d) be the head of a well-formed head of a tail Δ^3_T of Δ^3. Case (a) obviously need not be considered further. In either case (b) or case (c) this left parenthesis leads to point 2. In case (d), one of the points 3, 5, or 8 will be reached from points 2, 4, or 7, respectively. Let k be the value of i at point 2 in cases (b) and (c), and at the immediately preceding passage through point 2 in case (d).

It can easily be verified, by arguments parallel to those in (B1), that paths leaving these points must either correspond to ineffective paths in S_3, or lead to point 6 (or 9), where $p = [u, v_{m_k-1}, \cdots, v_1, p_{k-1}]$. Since $p \neq u$, if Δ^3 is exhausted by the right parenthesis leading to point 6 (or 9), the corresponding path in S_3 cannot be effective. It remains only to consider the case where point 6 (or 9) is reached, but the remaining $\Delta^3_T \neq \Lambda$.

Assume that for $i = k - 1$, any path leaving point 6 (or 9) is effective only if $\Delta^3 = [\Delta^3_H, \Delta^3_T]$ is well-formed, where Δ^3_H is the head of Δ^3 processed by S_3, prior to arrival at point 6 (or 9) and $\Delta^3_T \neq \Lambda$ is the tail remaining to be processed by S_3.

If $m_k > 1$, then $m_k - 1 \geqq 1$. There is a further effective path in S_3 only if $h^1/\Delta^3_T =)$ or δ'_{2k}. Since $m_k - 1 \geqq 1$,) leads to an ineffective path by virtue of condition (iii) of Definition 13(b). Note that except for a difference in the subscripts of v, point 3 and point 6 (or 9) have, in this case, equivalent configurations of p. An occurrence of δ'_{2k} therefore leads to point 4 with $p = [t, \delta_{2k}, v_{m_k-2}, \cdots, v_1, p_{k-1}]$. Since there are no further left parentheses by hypothesis, any path leaving point 4 either corresponds to an ineffective path in S_3, or leads to point 6 with $p = [u, v_{m_k-2}, \cdots, v_1, p_{k-1}]$. Repetition of this process, if feasible, must eventually lead back to point 6 with $p = [u, p_{k-1}] = [u, \delta_{ab}, v_{m_{k-1}-1}, \cdots, v_1, p_{(k-1)-1}]$. If $m_k = 1$, this condition is the original one at point 6 (or 9).

The condition $p = [u, \delta_{ab}, v_{m_{k-1}-1}, \cdots, v_1, p_{(k-1)-1}]$ is associated with point 5 if $\delta_{ab} = \delta_{2k}$ and with point 8 if $\delta_{ab} = \delta_{1j}$. There is a further effective path in S_3 only if $h^1/\Delta^3_T =)$, corresponding to a transition to point 6 (or 9) where $p = [u, v_{m_{k-1}-1}, \cdots, v_1, p_{(k-1)-1}]$. By the induction hypothesis, any further path is effective only if Δ^3 is well-formed, which completes the proof.

THEOREM 10. *S_3 is strictly effective if and only if Δ^3 is well-formed or $\Delta^3 = \Lambda$.*

The proof follows directly from Theorems 8 and 9, and Definition 13.

7. Some conclusions. The central role of Δ_M-theorems (Theorems 1 and 7) in translation both from L to P_3 and from P_3 to L is evident in §§5 and 6. Intuitively, this role may be explained by pointing out that any algorithm for which a Δ_M-theorem holds treats any nested well-formed subformula independently of the rest of the formula. As a consequence, such algorithms, if fail-safe, are fail-safe in a particularly satisfactory way: as one example, taken from natural languages, prepositional phrases or subordinate clauses can emerge unscathed, even though the sentence in which they are embedded may not be well-formed as a whole; as another example, taken from automatic programming, all the well-formed statements or subroutines of a program could be found at a single pass through a compiler, even though the program as a whole might not be well-formed. Debugging could therefore be considerably easier than it is in contemporary practice. Since L, P_1, P_2, and P_3 may all be regarded as representations of trees, the foregoing is equivalent to saying that any branch of a tree can be satisfactorily analyzed even if it has been broken off its parent branch.

In view of the underlying similarity of L and P_3 as tree representations, it may seem surprising that the proof of Theorem 9 for P_3 is so much less elegant than the proof of its counterpart for L, Theorem 5. There is, of course, the possibility that this may simply be due to a deficiency in the perception or ingenuity of the writer, who would gladly learn of a simpler proof. There may, however, be a deeper reason. The nonrecursive definition of well-formation or positiveness in L in terms of tail weight readily yields a nonrecursive definition of nonpositiveness, which is the essence of the proof of Theorem 5. An equally simple nonrecursive definition is lacking at present for P_3, hence the complexity of the proof of Theorem 9. In this light, it is gratifying that the complexity of S_3 is hardly greater than that of T_3.

Numerous extensions of the present study suggest themselves. For example, P_3 is a rather restrictive language for practical applications, and an extension to languages where parentheses may be omitted either in the presence of associative operators or in the presence of operators with a hierarchy, would be desirable. Such an extension has been given by Fischer [7] and turns out to be equivalent to that devised independently by Samelson and Bauer. In neither case has a check for well-formation been incorporated as yet, but the prospects for doing so are good. Samelson and Bauer have made it abundantly clear that the pushdown store technique, as they have applied it to the language ALGOL, is of unquestionable practical importance and, furthermore, that it can perform a great variety of functions not considered in the present paper. Their finding that "sequential treatment is not feasible in the case of certain optimizing processes such as recursive address calculation" is only one facet of the general problem of precisely characterizing the scope and the limitations of pushdown techniques, and of defining the place of these techniques in relation to syntactic theory, the theory of automata, algebra, and the topology of graphs.

The system described by Samelson and Bauer in Table 1 and Example 1 of their paper is essentially a translator from a parenthetic language with associative operators and an operator hierachy, feeding into an evaluator of the

BWW type. The control of transitions "by admissible state-symbol pairs" is not an essential feature of pushdown translators, since it is obviously absent in the algorithms of Figure 1 and 2; it is a feature introduced to account for operator hierarchy, much as state-symbol pairs are used to check for well-formation in the algorithm of Figure 3. Achievement of a deeper understanding of the rules whereby distinct pushdown algorithms may be combined and of the relation between the syntactic structures of languages and the features that algorithms must have to account for these structures is an important goal.

The development of a theory of pushdown algorithms should hopefully lead to systematic techniques for generating algorithms satisfying given requirements to replace the *ad hoc* invention of each new algorithm. It may be noted, for example, that Theorem 2 for T_3 applied to L has no equivalent for S_3 applied to P_3, due to the fact that the sequence ")(" is rejected by S_3. An appropriate theory would indicate, among other things, what modifications to S_3 should be made to accept the parenthetic equivalent of positive formulas in parenthesis-free notation. Such a theory would not only aid in devising appropriate algorithms for dealing with such languages as ALGOL, but also in the synthesis of languages which may be translated or analyzed by algorithms of the maximum simplicity and safe-failing characteristics consonant with other constraints such as the desired richness of expression and simplicity of the languages.

The results described in this paper provide a theoretical model that explains at least one essential feature of the predictive analysis technique for natural languages. Preliminary results obtained by Sherry suggest that this model can easily be extended to account for other important features of practical predictive analysis which, once fully understood, might find applications to the analysis and synthesis of artificial languages as well.

8. Acknowledgments. I am indebted for valuable criticisms and suggestions to my colleagues at the Computation Laboratory, to Professor V. H. Yngve of the Massachusetts Institute of Technology, and to several students in my course on Mathematical Linguistics, where much of the material in this paper was presented in the fall of 1959, especially to Patrick C. Fischer, who simplified an earlier version of T_3 essentially to the form given in §5.

Bibliography

1. Y. Bar Hillel, *A quasi-arithmetical notation for syntactic description*, Language vol. 29, no. 1 (1953) pp. 47–58.

2. ———, *Some linguistic obstacles to machine translation*, Appendix II in "Report on the state of machine translation in the United States and Great Britain," Technical Report no. 1, Hebrew University, Jerusalem, 1959.

3. A. W. Burks, D. W. Warren, and J. B. Wright, *An analysis of a logical machine using parenthesis-free notation*, Mathematical tables and other aids to computation, vol. VIII, no. 46 (1954) pp. 53–57.

4. J. W. Carr, *Programming and coding*, Chapter 2 in E. M. Grabbe, S. Ramo, and D. E. Wooldridge, "Handbook of automation, computation, and control," vol. 2, New York, Wiley, 1959.

5. N. Chomsky, *Syntactic structures*, 's-Gravenhage, Mouton, 1957.

6. Dartmouth Mathematics Project, *Symbolic work on high speed computers*, Project Report no. 4, Dartmouth, New Hampshire, 1959.

7. P. C. Fischer, *A proposal for a term project for Applied Mathematics* 205, manuscript, 1959.

8. S. Gorn, *On the logical design of formal mixed languages*, Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, 1959.

9. P. Z. Ingerman, *A new algorithm for algebraic translation*, Preprints of papers presented at the 14th National Meeting of the Association for Computing Machinery, 1959, pp. 22-1-22-2.

10. K. E. Iverson, *The description of finite sequential processes*, "Theory of switching," Report no. BL-23, section III, Harvard Computation Laboratory, Cambridge, Massachusetts, 1959.

11. K. E. Iverson, and F. P. Brooks, Jr. *Automatic data processing*, draft manuscript (to be published by Wiley, New York, 1960).

12. Ju. I. Janov, *O logicheskix szemax algoritmov*, "Problemy kibernetiki," no. 1, Gosudarstv. Izdat. Fiz.-Mat. Lit., Moscow (1958) pp. 75-127.

13. J. Kanner, *An algebraic translator*, Communications of the Association for Computing Machinery vol. 2 no. 10 (1959) pp. 19-22.

14. J. Lambek, *The mathematics of sentence structure*, Amer. Math. Monthly vol. 65 no. 3 (1958) pp. 154-170.

15. A. A. Ljapunov, *O logicheskix szemax programm*, "Problemy kibernetiki," no. 1, Gosudarstv. Izdat. Fiz.-Mat. Lit., Moscow (1958) pp. 46-74.

16. W. Miehle, *Burroughs truth function evaluator*, Journal of the Association for Computing Machinery vol. 4, no. 2 (1957) pp. 189-192.

17. G. A. Miller, *Human memory and the storage of information*, I.R.E. Transactions on Information Theory vol. IT-2, no. 3 (1956) pp. 129-137.

18. A. G. Oettinger, *A new basic approach to automatic data processing*, Progress Report no. 3 by the Staff of the Computation Laboratory to the American Gas Association and Edison Electric Institute, section III, Harvard University, Cambridge, Massachusetts, 1956.

19. ———, *Current research on automatic translation at Harvard University*, National Symposium on Machine Translation, Los Angeles, February, 1960 (to appear in Proceedings of the Symposium).

20. ———, *A new theory of translation and its applications*, National Symposium on Machine Translation, Los Angeles, February, 1960 (to appear in Proceedings of the Symposium).

21. ———, *Automatic language translation: lexical and technical aspects*, Harvard Monographs in Applied Science, no. 8, Cambridge, Massachusetts, Harvard University Press, 1960.

22. I. Rhodes, *A new approach to the mechanical translation of Russian*, National Bureau of Standards, Washington, D. C., unpublished report, 1959.

23. ———, *A new approach to the mechanical syntactic analysis of Russian*, National Bureau of Standards, Washington, D. C., unpublished report, 1959.

24. ———, *The NBS method of syntactic integration*, National Symposium on Machine Translation, Los Angeles, February, 1960 (to appear in Proceedings of the Symposium).

25. J. Riguet, private communication, 1957.

26. P. Rosenbloom, *The elements of mathematical logic*, Dover, New York, 1950.

27. H. Rutishauser, *Automatische Rechenplanfertigung bei Programmgesteuerten Rechemaschinen*, Mitt. Inst. Angew. Math. Zürich, no. 3, Basel, Birkhaüser, 1952.

28. K. Samelson and F. L. Bauer, *Sequential formula translation*, Communications of the Association for Computing Machinery vol. 3, no. 2 (1960) pp. 76–83.

29. M. E. Sherry, *Predictive syntatic analysis*, National Symposium on Machine Translation, Los Angeles, February, 1960 (to appear in Proceedings of the Symposium).

30. R. Wells, *Immediate constituents*, Language vol. 23 (1947) pp. 81–117.

31. L. Wundheiler and A. Wundheiler, *Some logical concepts for syntax*, Chapter 13 in W. N. Locke and A. D. Booth, "Machine Translation of Languages," Cambridge, Massachusetts, Technology Press, and New York, Wiley, 1955.

32. V. H. Yngve, *Left-to-right sentence generation*, draft manuscript, 1959.

33. ———, *A model and an hypothesis for language structure*, draft manuscript, 1959 (to appear in Proc. Amer. Philos. Soc.).

HARVARD UNIVERSITY,
CAMBRIDGE, MASSACHUSETTS