# THE MATHEMATICS OF SENTENCE STRUCTURE*

JOACHIM LAMBEK, McGill University

*The definitions* [of the parts of speech] *are very far from having attained the degree of exactitude found in Euclidean geometry.*

—Otto Jespersen, 1924.

**1. Introduction.** The aim of this paper is to obtain an effective rule (or algorithm) for distinguishing sentences from nonsentences, which works not only for the formal languages of interest to the mathematical logician, but also for natural languages such as English, or at least for fragments of such languages. An attempt to formulate such an algorithm is implicit in the work of Ajdukiewicz.† His method, later elaborated by Bar-Hillel [2], depends on a kind of arithmetization of the so-called *parts of speech*, here called *syntactic types*.‡

The present paper begins with a new exposition of the theory of syntactic types. It is addressed to mathematicians with at most an amateur interest in linguistics. The choice of sample languages is therefore restricted to English and mathematical logic. For the same reason, technical terms have been borrowed from the field of high school grammar.

Only a fragmentary treatment of English grammar is presented here. This should not be taken too seriously, but is meant to provide familiar illustrations for our general methods. The reader should not be surprised if he discovers considerable leakage across the line dividing sentences from nonsentences. It is only fair to warn him that some authorities think that such difficulties are inherent in the present methods.§ We take consolation in the words of Sapir: "All grammars leak."

The second part of this paper is concerned with a development of the technique of Ajdukiewicz and Bar-Hillel in a mathematical direction. We introduce a calculus of types, which is related to the well-known calculus of residuals.**

* This paper was written while the author held a Summer Research Associateship from the National Research Council of Canada. The present discussion of English grammar, in its final form, owes much to the careful reading and helpful criticism of earlier versions by Bar-Hillel and Chomsky.

† An English translation of his paper [1] is available in mimeographed form at the University of Chicago.

‡ Historically, these types can be traced back to the *semantic types* attributed by Tarski [21, p. 215] to E. Husserl and S. Lesniewski. A similar technique for logical systems was developed independently by Church [8]. Closely related is also the work by Curry [11] on *functional characters*. These correspond approximately to syntactic types for languages in which functors are always written on the left of their arguments.

§ Chomsky [6; 7] believes that such methods can describe only a small proportion of the sentences of a natural language and that other sentences should be obtained from these by certain transformations.

** See [3, XIII]. The calculus presented here is formally identical with a calculus constructed by G. D. Findlay and the present author for a discussion of canonical mappings in linear and multilinear algebra.

The decision problem for this system is solved affirmatively, following a procedure first proposed by Gentzen for the intuitionistic propositional calculus.††

The methods described here may be applied in several fields. For the teaching of English they provide a rigorous version of the traditional activity known as *diagramming* and *parsing*. For introductory logic courses they offer an effective definition of *well-formed formulas*. For the mechanical translation of languages [**16**], they may help with the syntactic analysis of the input material and indicate how to arrange the output into grammatical sentences of the target language. For the construction of an auxiliary language, they tell how to achieve a completely regular syntax; this is of special importance when the auxiliary is to act as an intermediate language in mechanical translation.

**2. Syntactic types.** While linguists are primarily interested in speech rather than in written texts, we shall here confine attention to the latter, if only to escape the difficult task of breaking up continuous discourse into discrete words. By a *word* we shall understand a word-form: Such forms as *work*, *works*, *worked* and *working* are different words; but the word *work* occurs twice in *we work best when we like our work*, although it functions as a verb in the first place and as a noun in the second. To describe the function of a word or expression we ascribe to it a certain *syntactic type*. This concept will now be defined; it corresponds approximately to the traditional *part of speech*.

We begin by introducing two *primitive* types: $s$, the type of sentences, and $n$, the type of names. For the sake of simplicity, we here restrict *sentence* to denote complete declarative sentences, ruling out requests and questions (as well as most replies, which are usually incomplete). By a *name* we understand primarily a proper name, such as *John* or *Napoleon*. But we shall also assign type $n$ to all expressions which can occur in any context in which all proper names can occur. Thus type $n$ is ascribed to the so-called class-nouns *milk*, *rice*, $\cdots$, which can occur without article, and to compound expressions such as *poor John*, *fresh milk*, $\cdots$.‡‡ We do not need to assign type $n$ to the so-called count-nouns *king*, *chair*, $\cdots$, which require an article, nor to the pronoun *he*, as it cannot replace *John* in *poor John works* or *milk* in *John likes milk*.

From the primitive types we form compound types, by the recursive definition: If $x$ and $y$ are types, then so are $x/y$ (read *x over y*) and $y\backslash x$ (read *y under x*). The meaning of these two kinds of division will be made clear by two examples.

The adjective *poor* modifies the name *John* from the left, producing the noun-phrase *poor John*. We assign to it type $n/n$.

The predicate (intransitive verb) *works* transforms the name *John* from the right into the sentence *John works*. We assign to it type $n\backslash s$.

In general, an expression of type $x/y$ when followed by an expression of type

†† See [**13**; **10**, II; **15**, XV]. Curry [**11**, appendix] has also observed the close analogy between the theory of functional characters and the propositional calculus.

‡‡ There is a difficulty here: Of course we cannot check *all* admissible name contexts (whose number is infinite) to see whether *poor John* can be fitted in. Our assignment of types is tentative and subject to future revision.

$y$ produces an expression of type $x$, and so does an expression of type $y\backslash x$ when preceded by an expression of type $y$. We write symbolically

(I) $$(x/y)y \to x, \qquad y(y\backslash x) \to x.$$

**3. Type list for a fragment of English.** We shall illustrate the assignment of types to English words by considering a number of sample sentences.

(1) $$\begin{array}{cc} \textit{John} & \textit{works} \\ n & n\backslash s \end{array}$$

This remains a sentence if *John* is replaced by any other name, hence *works* has type $n\backslash s$.

(2) $$\begin{array}{ccc} (\textit{poor} & \textit{John}) & \textit{works} \\ n/n & n & n\backslash s \end{array}$$

Here *poor John* takes the place of the name in (1); in fact *poor John* can occur in any context in which all names can occur, hence it has type $n$. Moreover, so has *poor Tom*, *poor Jane*, $\cdots$, thus *poor* has type $n/n$.

(3) $$\begin{array}{ccc} (\textit{John} & \textit{works}) & \textit{here} \\ n & n\backslash s & s\backslash s \end{array}$$

The word *here* transforms (1), or any other sentence, into a new sentence, hence it has type $s\backslash s$. The question may be raised whether *here* can be attached to a sentence such as (3) itself. While *John works here here* is open to stylistic objections, we shall consider it grammatically well-formed.

(4) $$\begin{array}{ccc} \textit{John} & (\textit{never} & \textit{works}) \\ n & (n\backslash s)/(n\backslash s) & n\backslash s \end{array}$$

Since *John* can be replaced by any name here, *never works* has type $n\backslash s$, and so has *never sleeps*, $\cdots$; hence *never* has type $(n\backslash s)/(n\backslash s)$. It may be argued that (3) could also have been grouped *John* (*works here*) suggesting the type $(n\backslash s)\backslash(n\backslash s)$ for *here*. It will be shown later that every adverbial expression of type $s\backslash s$ also has type $(n\backslash s)\backslash(n\backslash s)$.

(5) $$\begin{array}{cccc} (\textit{John} & \textit{works}) & (\textit{for} & \textit{Jane}) \\ n & n\backslash s & (s\backslash s)/n & n \end{array}$$

This indicates that *for Jane* should have the same type as *here* in (3), namely $s\backslash s$, and since *Jane* can be replaced by any other name *for* has type $(s\backslash s)/n$.

(6) $$\begin{array}{ccccc} (\textit{John} & \textit{works}) & (\textit{and} & (\textit{Jane} & \textit{rests})) \\ n & n\backslash s & (s\backslash s)/s & n & n\backslash s \end{array}$$

This illustrates how *and* can join two arbitrary sentences to form a new sentence; its type is therefore $(s\backslash s)/s$.

(7)
$$\begin{matrix} \textit{John} & (\textit{likes} & \textit{Jane}) \\ n & (n\backslash s)/n & n \end{matrix}$$

Here *likes Jane* has the same type as *works* in (1), hence *likes* has type $(n\backslash s)/n$. Similarly we may write *John* (*likes milk*) and even *milk* (*likes John*). The latter is a grammatical sentence, though open to semantic objections.

Example (7) raises an important point. Let us group this sentence

(7′)
$$\begin{matrix} (\textit{John} & \textit{likes}) & \textit{Jane} \\ n & n\backslash(s/n) & n \end{matrix}$$

Here *John likes* has type $s/n$, hence *likes* must be given the new type $n\backslash(s/n)$. We would regard the two types of *likes* in (7) and (7′) as in some sense equivalent. Abstracting from this particular situation, we write symbolically

(II)
$$(x\backslash y)/z \rightleftarrows x\backslash(y/z).$$

In practical applications it is often tedious to distinguish between equivalent types, we then write $x\backslash y/z$ for either side of (II). Further examples of this convention are afforded by the types of *never*, *for* and *and* [see Table I]. To avoid multiplication of parentheses, we may also abbreviate $(x/y)/z$ as $x/y/z$, and, symmetrically, $z\backslash(y\backslash x)$ as $z\backslash y\backslash x$. However, parentheses must not be omitted in such compounds as $x/(y/z)$, $(z\backslash y)\backslash x$, $(x/y)\backslash z$ and $z/(y\backslash x)$.

Table I compares the syntactic types of the words discussed above with the traditional parts of speech and the recent classification of Fries [12].

TABLE I

| | Word | Type | Part of Speech | Fries Class |
|---|---|---|---|---|
| (1) | *works* | $n\backslash s$ | intransitive verb | 2C |
| (2) | *poor* | $n/n$ | adjective | 3 |
| (3) | *here* | $s\backslash s$ | adverb | 4 |
| (4) | *never* | $n\backslash s/(n\backslash s)$ | adverb | |
| (5) | *for* | $s\backslash s/n$ | preposition | F |
| (6) | *and* | $s\backslash s/s$ | conjunction | E, J |
| (7) | *likes* | $n\backslash s/n$ | transitive verb | 2B |

It is fairly clear that in this manner we can build up a type list for a gradually increasing portion of English vocabulary. This should be subject to possible revision, as more information becomes available.

To distinguish between different forms such as *works* and *work*, usually represented by a single dictionary entry, it is necessary to allow for more than two primitive types. Thus we might assign the type $n^*$ to all noun-plurals, such as *men*, *chairs*, · · · . In contrast to examples (1), (2), (5), (7) we then have

(1*)
$$\begin{matrix} \textit{men} & \textit{work} \\ n^* & n^*\backslash s \end{matrix}$$

(2*) *poor men work*

$$n^*/n^* \quad n^* \quad n^*\backslash s$$

(5*) *John works for men*

$$n \quad n\backslash s \quad s\backslash s/n^* \quad n^*$$

(7*) *John likes girls, men like Jane, men like girls*

$$n \quad n\backslash s/n^* \quad n^* \qquad n^* \quad n^*\backslash s/n \quad n \qquad n^* \quad n^*\backslash s/n^* \quad n^*$$

This assignment successfully distinguishes between the forms *work* and *works*, *like* and *likes*, but it introduces an undesirable multiplicity of types for *poor*, *for*, *like*, and *likes*. While French distinguishes the forms *pauvre* and *pauvres*, English fails to make a corresponding distinction.

A more thorough analysis of the English verb phrase would compel us to introduce further primitive types for the infinitive and the two kinds of participles of intransitive verbs. This would lead to some revision of the type list embodied in Table I. While giving a more adequate treatment of English grammar, such a program would not directly serve the purpose of the present paper.

**4. Formal systems.** Suppose we have before us a string of words whose types are given. Then we can compute the type of the entire expression, provided its so-called *phrase structure* has been made visible by some device such as brackets. Consider for example

*John (likes (fresh milk))*

$$\underbrace{n \quad \underbrace{n\backslash s/n \quad \underbrace{n/n \quad n}_{n}}_{n\backslash s}}_{s}$$

The indicated computation can also be written in one line:

$$n((n\backslash s/n)((n/n)n)) \rightarrow n((n\backslash s/n)n) \rightarrow n(n\backslash s) \rightarrow s.$$

In the formal languages studied by logicians, this process offers an effective test whether a given grouped string of symbols is a well-formed formula. For in these languages each word (usually consisting of a single sign) has just one preassigned type, and the use of brackets is obligatory. Let us call expressions with built-in brackets *formulas*; then formulas may be defined recursively: Each word is a formula, and if $A$ and $B$ are formulas, so is $(AB)$.

Brackets are usually omitted when this can be done without introducing ambiguity. Brackets are regularly omitted in accordance with Rule (II). Thus logicians write

$$\begin{array}{ccc} p & \rightarrow & q \\ s & s\backslash s/s & s \end{array}$$

rather than

$$\begin{array}{ccc} p & ( \rightarrow & q) \\ s & (s\backslash s)/s & s \end{array}$$

Allowance being made for this convention, the sentence structure of a formalized language is completely determined by its type list. A number of examples will illustrate this.

1. The propositional calculus, according to one of its formulations, possesses an infinite sequence of propositional variables of type $s$, and two signs for negation and implication of types $s/s$ and $s\backslash s/s$ respectively.

The Polish school of logicians prefer to write all functors on the left of their arguments; it is well-known [18, IV] that all brackets can then be omitted without introducing ambiguity. The implication sign in the Polish notation is therefore of type $s/s/s$.

2. Boolean algebra, rather redundantly formulated, contains an infinite sequence of individual variables, as well as the signs 0 and 1, all of type $n$, an accent (for complementation) of type $n\backslash n$, cap and cup of type $n\backslash n/n$, equality and inclusion signs of type $n\backslash s/n$.

3. Quine's mathematical logic [17], into which we here introduce a special sign for universal quantification, contains an infinite sequence of individual variables of type $n$, and signs for joint denial, universal quantification and membership of types $s\backslash s/s$, $s/s/n$ and $n\backslash s/n$ respectively.

4. The calculus of lambda conversion due to Church, with a special sign of type $n/n/n$ for application [18, p. 111], contains also an infinite sequence of individual variables $x_i$ $(i=1, 2, \cdots)$ of type $n$, together with a parallel sequence $\lambda x_i$ of type $n/n/n$.

5. The syntactic calculus to be introduced in this paper contains a number of symbols for primitive types of type $n$, three connectives $\cdot$, $\backslash$, $/$ of type $n\backslash n/n$, and the sign $\rightarrow$ of type $n\backslash s/n$.

In the interpretation of formal languages [21, XVIII, Section 4] one usually assumes that expressions of type $s$ denote truth values, expressions of type $n$ denote members of a given domain of individuals, and expressions of type $x/y$ or $y\backslash x$ denote functions from the class of entities denoted by expressions of type $y$ into the class of entities denoted by expressions of type $x$.

The above discussion of formal systems is somewhat oversimplified. Thus in Quine's formulation of mathematical logic, no special symbol is used for universal quantification, and in Church's formulation of the calculus of lambda conversion the sign for application is not written. The syntactic description of these languages in terms of types would be more complicated without the special symbols introduced here. In some languages it is important to distinguish between constants and variables of apparently the same type [see, *e.g.*, **1**]. A description in terms of two primitive types is then no longer adequate.

**5. Type computations in English.** Suppose we wish to compute the type of a string of English words, which are taken from a given type list. We cannot proceed quite as directly as in the formal systems discussed above, for two reasons, which we shall pause to discuss.

First, brackets do not usually occur in English texts, unless we regard punctuation as a half-hearted attempt to indicate grouping. Two ways of inserting brackets into an expression such as *the daughter of the woman whom he loved* may lead to essentially different syntactic resolutions, which may be accompanied by different meanings.

Secondly, English words usually possess more than one type. We have seen some examples of this in Section 3; others are easily found: The adverbial expression *today* has type $s/s$ or $s\backslash s$, depending on whether it precedes or follows the sentence modified. The word *sound* may be a noun, an adjective, or a verb, either transitive or intransitive, depending on the context. Some "chameleon" words possess a type which is systematically ambiguous, allowing them to blend into many different contexts. Thus *only*, of type $x/x$, can probably modify expressions of any type $x$, and *and*, of type $x\backslash x/x$, will join together expressions of almost any type $x$ to form a compound of the same type.

A mechanical procedure for analyzing English sentences would consist of four steps:

I. Insert brackets in all admissible ways.

II. To each word assign all types permitted by a given finite type list. (We ignore for the moment the difficulty arising from words which possess a potentially infinite number of types, as do the chameleons *and* and *only*).

III. For each grouping and type assignment compute the type of the total expression.

IV. Select that method of grouping and that type assignment which yields the desired type $s$.

A simple example, in which the problem of grouping does not arise, is

| *time* | *flies* |
|---|---|
| $n$ | $n^*$ |
| $n^*\backslash s/n$ | $n\backslash s$ |
| $n^*\backslash s/n^*$ | $n\backslash s/n$ |
|  | $n\backslash s/n^*$ |

Only the assignment

$$\begin{array}{cc} \textit{time} & \textit{flies} \\ n & n\backslash s \end{array}$$

produces a declarative sentence. This may be contrasted with

$$\begin{array}{ccccc} (\textit{spiders} & \textit{time} & \textit{flies}) & \textit{without} & \textit{clocks}, \\ n^* & n^*\backslash s/n^* & n^* & s\backslash s/n^* & n^* \end{array}$$

and

$$\begin{array}{cccccc} (\textit{TIME} & \textit{flies} & (10{,}000 & \textit{copies})) & \textit{to} & \textit{Montreal}. \\ n & n\backslash s/n^* & n^*/n^* & n^* & s\backslash s/n & n \end{array}$$

6. **Pronouns.** So far we have confined attention to the computation Rules (I) and (II). We have had one indication that other rules may play a role: the discussion of Example (4) suggests the rule

$$s\backslash s \rightarrow (n\backslash s)\backslash(n\backslash s).$$

To give a heuristic introduction for the consideration of further rules, we enter into a short discussion of English pronouns.

$$\begin{array}{ccccc} \textit{he} & \textit{works}, & \textit{he} & \textit{likes} & \textit{Jane} \\ s/(n\backslash s) & n\backslash s & s/(n\backslash s) & n\backslash s/n & n \end{array} \tag{8}$$

Since *he* transforms such expressions as *works, likes Jane,* · · · , of type $n\backslash s$ into sentences, we assign to it type $s/(n\backslash s)$. We could of course enlarge the class of names to include pronouns, but then we should be hard put to explain why *poor he works* and *Jane likes he* are not sentences. At any rate, the assignment of type $s/(n\backslash s)$ to *he* is valid, irrespective of whether we regard pronouns as names. In fact, by the same argument, the name *John* also has type $s/(n\backslash s)$. This point will be discussed later.

$$\begin{array}{ccccc} \textit{that's} & \textit{him}, & \textit{Jane} & \textit{likes} & \textit{him}, \\ s/n & (s/n)\backslash s & n & n\backslash s/n & (s/n)\backslash s \end{array} \tag{9}$$

$$\begin{array}{cccc} \textit{Jane} & \textit{works} & \textit{for} & \textit{him} \\ n & n\backslash s & s\backslash s/n & (s/n)\backslash s \end{array}$$

The expressions *that's*, *Jane likes* and *Jane works for* all have type $s/n$, hence we have ascribed type $(s/n)\backslash s$ to *him*. (This assignment is not quite correct:* The example *Jane likes poor John* indicates that the expression *Jane likes poor* also has type $s/n$, yet *Jane likes poor him* is not a sentence. Moreover the present assignment does not explain why *that's he* is a sentence in the speech of some people. We shall overlook these defects here.) We observe that the difference

* This was kindly pointed out to the author by N. Chomsky.

in form between *he* and *him* is reflected by a difference in type, indicating that the former operates from the left, while the latter operates from the right. Sapir [19, VII] has called these two forms the *pre-verbal* and *post-verbal* case of the pronoun respectively.

A difficulty arises when we try to show the sentencehood of

$$\begin{matrix} he & likes & him \;; \\ s/(n\backslash s) & n\backslash s/n & (s/n)\backslash s \end{matrix} \tag{10}$$

for

$$(s/(n\backslash s))(n\backslash s/n)((s/n)\backslash s)$$

cannot be simplified any further by the Rules (I) and (II). We introduce two new rules

$$(x/y)(y/z) \to x/z, \qquad (x\backslash y)(y\backslash z) \to x\backslash z. \tag{III}$$

We may then assign type

$$(s/(n\backslash s))(n\backslash s/n) \to s/n$$

to *he likes* and type

$$(n\backslash s/n)((s/n)\backslash s) \to n\backslash s$$

to *likes him*, permitting two equivalent resolutions

$$\underbrace{(he\ likes)}_{s/n}\ \underset{(s/n)\backslash s}{him,} \qquad \underset{s/(n\backslash s)}{he}\ \underbrace{(likes\ him).}_{n\backslash s}$$

Rules (III) also allow alternative, though equivalent, resolutions of expressions considered earlier; *e.g.*, the sentence

$$\begin{matrix} (John & works) & for & Jane \\ n & n\backslash s & s\backslash s/n & n \end{matrix}$$

can now also be grouped *John* (*works* (*for Jane*)), where the predicate has type

$$(n\backslash s)((s\backslash s/n)n) \to (n\backslash s)(s\backslash s) \to n\backslash s.$$

We have seen above that the name *John* also has the type of the pronoun *he*. For the same reason, it also has the type of the pronoun *him*. We symbolize the situation by writing

$$n \to s/(n\backslash s), \qquad n \to (s/n)\backslash s$$

and more generally

$$x \to y/(x\backslash y), \qquad x \to (y/x)\backslash x. \tag{IV}$$

These new rules may actually be required for computations. Suppose that from sample sentences such as *books by him bore* we arrived at the type $n^*\backslash s/n'$ for *by*, where $n'$ is short for $(s/n)\backslash s$. The phrase *books by John* then requires the computation

$$n^*(n^*\backslash s/n')n \rightarrow (s/n')n \rightarrow (s/n')n' \rightarrow s$$

which utilizes rules (I), (IV) and (I) in this order.

While Ajdukiewicz [1] makes use of (III), Rules (IV) suggest that the mathematical apparatus used hitherto may have to be expanded.

**7. Syntactic calculus.** By an *expression* we shall mean a string of words. Let us suppose that to certain expressions there have been assigned certain primitive types. If $A$ has type $x$ and $B$ has type $y$, we assign to the expression $AB$ the type $xy$, also written $x\cdot y$. We assign type $z/y$ to all expressions $A$ such that $AB$ has type $z$ for any $B$ of type $y$. We assign type $x\backslash z$ to all expressions $B$ such that $AB$ has type $z$ for any $A$ of type $x$. We write $x\rightarrow y$ to mean that any expression of type $x$ also has type $y$. We write $x\rightleftarrows y$ to mean that $x\rightarrow y$ and $y\rightarrow x$.

The following rules are now valid:

(a) $x \rightarrow x$

(b) $(xy)z \rightarrow x(yz)$ (b′) $x(yz) \rightarrow (xy)z$

(c) *if* $xy \rightarrow z$ *then* $x \rightarrow z/y$ (c′) *if* $xy \rightarrow z$ *then* $y \rightarrow x\backslash z$

(d) *if* $x \rightarrow z/y$ *then* $xy \rightarrow z$ (d′) *if* $y \rightarrow x\backslash z$ *then* $xy \rightarrow z$

(e) *if* $x \rightarrow y$ *and* $y \rightarrow z$ *then* $x \rightarrow z$

Rules (a), (b), (b′), (e) hold trivially. Rules (c′) and (d′) are symmetric duals of (c) and (d), hence it suffices to prove the latter.

Assume $xy\rightarrow z$, and let $A$ have type $x$. Then for any $B$ of type $y$, $AB$ has type $z$; hence $A$ has type $z/y$. Thus $x\rightarrow z/y$.

Conversely, assume $x\rightarrow z/y$, and let $A$, $B$ have types $x$, $y$, respectively; then $AB$ has type $z$. Thus $xy\rightarrow z$.

The system presented above may be viewed abstractly as a formal language with a number of primitive type symbols of type $n$, three connectives $\cdot$, $/$, $\backslash$ of type $n\backslash n/n$, and a relation symbol $\rightarrow$ of type $n\backslash s/n$. If we furthermore regard (a), (b) and (b′) as axiom schemes and (c) to (e) as rules of inference, we obtain a deductive system which may be called *syntactic calculus*. A number of rules are provable in the system; for example,

(f) $x \rightarrow (xy)/y,$

(g) $(z/y)y \rightarrow z,$

(h) $y\rightarrow (z/y)\backslash z,$

(i) $(z/y)(y/x) \to z/x,$

(j) $z/y \to (z/x)/(y/x),$

(k) $(x\backslash y)/z \rightleftarrows x\backslash(y/z),$

(l) $(x/y)/z \rightleftarrows x/(zy),$

(m) *if* $x \to x'$ *and* $y \to y'$ *then* $xy \to x'y',$

(n) *if* $x \to x'$ *and* $y \to y'$ *then* $x/y' \to x'/y.$

Here (f) follows from $xy \to xy$ by (c), (g) follows from $z/y \to z/y$ by (d), (h) follows from (g) by (c′), (j) follows from (i) by (c). Proofs of (i), (k) and (l) are a bit longer; we omit them in view of the decision procedure established in Section 8. Proofs of (m) and (n) are arranged in tree form.

Proof of (m).

$$\dfrac{\dfrac{\dfrac{x \to x' \quad \dfrac{x'y \to x'y}{x' \to (x'y)/y}\,(\mathrm{c})}{x \to (x'y)/y}\,(\mathrm{e})}{xy \to x'y}\,(\mathrm{d}) \qquad \dfrac{\dfrac{y \to y' \quad \dfrac{x'y' \to x'y'}{y' \to x'\backslash(x'y')}\,(\mathrm{c}')}{y \to x'\backslash(x'y')}\,(\mathrm{e})}{x'y \to x'y'}\,(\mathrm{d}')}{xy \to x'y'}\,(\mathrm{e})$$

Proof of (n).

$$\dfrac{\dfrac{\dfrac{\dfrac{y \to y' \quad \dfrac{\dfrac{x/y' \to x/y'}{(x/y')y' \to x}\,(\mathrm{d})}{y' \to (x/y')\backslash x}\,(\mathrm{c}')}{y \to (x/y')\backslash x}\,(\mathrm{e})}{(x/y')y \to x}\,(\mathrm{d}')}{x/y' \to x/y}\,(\mathrm{c}) \qquad \dfrac{\dfrac{\dfrac{x/y \to x/y}{(x/y)y \to x}\,(\mathrm{d}) \quad x \to x'}{(x/y)y \to x'}\,(\mathrm{e})}{x/y \to x'/y}\,(\mathrm{d})}{x/y' \to x'/y}\,(\mathrm{e})$$

The syntactic theorems (g), (h), (i), and (k) coincide with the Rules (I), (IV), (III), and (II), respectively. An illustration of (j), or rather its symmetric dual, appeared in Section 3, where it was pointed out that every sentence-modifying adverb is also a predicate-modifying adverb, symbolically,

$$s\backslash s \to (n\backslash s)\backslash(n\backslash s).$$

Rule (l) is due to Schönfinkel [20], who observed that a function of two variables may be regarded as an ordinary function of one variable whose value is again an ordinary function, so that

$$f(a, b) = (fa)b.$$

If $a$, $b$ and $f(a, b)$ have types $x$, $y$ and $z$ respectively, then $f$ occurs in $f(a, b)$ with type $z/(xy)$ and in $(fa)b$ with type $(z/y)/x$, these two types being equivalent by (1).

**8. Decision procedure.** Is there an effective method for testing whether a sentence $x \to y$ of the syntactic calculus is deducible from rules (a) to (e)? This is the so-called decision problem for the syntactic calculus. It turns out that the decision procedure discovered by Gentzen [**15**, XV] for the intuitionistic propositional calculus can be adapted for the present purpose.

Following Gentzen, we define the *sequent*

$$x_1, x_2, \cdots x_n \to y$$

to stand for

$$(\cdots((x_1x_2)x_3)\cdots x_n) \to y,$$

where $x_1, \cdots, x_n, y$ are types. Now let $x$ be any of the possible products of the $x_i$ obtained from some way of grouping the string $x_1x_2 \cdots x_n$. Then it follows by repeated application of rules (b), (b′), (m) and (e) that

$$x \rightleftarrows (\cdots((x_1x_2)x_3)\cdots x_n).$$

Hence the above sequent is also equivalent to the formula $x \to y$.

Let capitals denote sequences of types, possibly empty sequences. By "$U$, $V$" we mean the sequence obtained by juxtaposing $U$ and $V$; if $U$ is empty it means $V$, and if $V$ is empty it means $U$. The following rules are consequences of (a) to (e), provided $T$, $P$ and $Q$ are not empty.

(1) $x \to x$

(2) *if* $T, y \to x$ *then* $T \to x/y$

(2′) *if* $y, T \to x$ *then* $T \to y\backslash x$

(3) *if* $T \to y$ *and* $U, x, V \to z$ *then* $U, x/y, T, V \to z$

(3′) *if* $T \to y$ *and* $U, x, V \to z$ *then* $U, T, y\backslash x, V \to z$

(4) *if* $U, x, y, V \to z$ *then* $U, xy, V \to z$

(5) *if* $P \to x$ *and* $Q \to y$ *then* $P, Q \to xy$

Note that each of Rules (2) to (5) introduces an occurrence of one of the connectives $\cdot$, $/$, $\backslash$ into the conclusion.

To derive Rules (1) to (5) from (a) to (e), we observe that (1) is the same as (a), (2) becomes (c), (2′) becomes (c′), (4) is immediate, and (5) becomes (m), if the sequences $T$, $U$, $V$, $P$, and $Q$ are replaced by the products of the terms in them. It remains only to prove (3), since (3′) is its symmetric dual.

First let us take the case where $U$ and $V$ are empty sequences. We replace $T$ by some product $t$ of its terms. Then (3) takes the form: *if* $t \to y$ *and* $x \to z$ *then* $(x/y)t \to z$. This may be shown thus:

$$\dfrac{\dfrac{x \to z \quad t \to y}{x/y \to z/t}\,(\mathrm{n})}{(x/y)t \to z}\,(\mathrm{d})$$

Next suppose $U$ is empty but $V$ is not. Replace the latter by a product $v$ of its terms. Then (3) takes the form: *if* $t \to y$ *and* $xv \to z$ *then* $((x/y)t)v \to z$. This is established thus:

$$\dfrac{\dfrac{\dfrac{xv \to z}{x \to z/v}\,(\mathrm{c}) \quad t \to y}{(x/y)t \to z/v}\,(\text{as above})}{((x/y)t)v \to z}\,(\mathrm{d})$$

Similarly we deal with the remaining two cases in which $U$ is not empty.

Conversely, we shall deduce rules (a) to (e) from (1) to (5), so that the two sets of rules are equivalent. For the moment we assume one additional rule, the so-called *cut,*

(6) *if* $T \to x$ *and* $U, x, V \to y$ *then* $U, T, V \to y$

It will appear later (Gentzen's theorem) that this new rule does not increase the set of theorems deducible from (1) to (5).

Now (a) coincides with (1), and (e) is a special case of (6), hence it suffices to prove (b), (c) and (d). Proofs are arranged in tree form.

Proof of (b).

$$\dfrac{\dfrac{\dfrac{x \to x \quad \dfrac{y \to y \quad z \to z}{y, z \to yz}\,(5)}{x, y, z \to x(yz)}\,(5)}{xy, z \to x(yz)}\,(4)}{(xy)z \to x(yz)}\,(4)$$

Proof of (c).

$$\dfrac{\dfrac{\dfrac{\dfrac{x \to x \quad y \to y}{x, y \to xy}\,(5)}{x \to (xy)/y}\,(2) \qquad \dfrac{y \to y \quad xy \to z}{(xy)/y, y \to z}\,(3)}{x, y \to z}\,(6)}{x \to z/y}\,(2)$$

Proof of (d).

$$\cfrac{\cfrac{\cfrac{x \to z/v \quad v \to v}{x, v \to (z/v)v}\,(5) \qquad \cfrac{\cfrac{v \to y \quad z \to z}{z/y, v \to z}\,(3)}{(z/v)y \to z}\,(4)}{x, v \to z}\,(6)}{xy \to z}\,(4)$$

Let us verify that we have, in fact, a decision procedure. Given a sequent $U \to x$, we attempt to construct a proof in tree form, working from the bottom up, using Rules (1) to (5), but not (6). Every upward step eliminates an occurrence of one of the connectives $\cdot$, $/$, $\backslash$, and there are only a finite number of ways of making this step. Therefore the total number of proofs that can be attempted is finite. The sequent $U \to x$ is deducible if and only if one of the attempted proofs is successful.

**9. Proof of Gentzen's theorem.** If $T \to x$ and $U, x, V \to y$ are both provable according to Rules (1) to (5), we will show that $U, T, V \to y$ is also provable, so that we may adopt as a new rule of inference the cut

$$\frac{T \to x \quad U, x, V \to v}{U, T, V \to y}\,(6).$$

We prove this by reduction on the *degree* of the cut, which is defined thus: Let $d(x)$ be the number of separate occurrences of the connectives $\cdot$, $/$, $\backslash$ in the type formula $x$, and let

$$d(x_1, x_2, \cdots x_n) = d(x_1) + d(x_2) + \cdots d(x_n),$$

then the degree of the above cut is

$$d(T) + d(U) + d(V) + d(x) + d(y).$$

We will now show that in any cut, whose premises have been proved without cut, the conclusion is either identical with one of the premises, or else the cut can be replaced by one or two such cuts of smaller degree. Since no degree is negative, this will establish Gentzen's theorem. We consider seven cases, which need not be mutually exclusive.

*Case* 1. $T \to x$ is an instance of (1); then $T = x$ and the conclusion coincides with the other premise.

*Case* 2. $U, x, V \to y$ is an instance of (1); then $U$ and $V$ are empty and $x = y$. Hence the conclusion coincides with the premise $T \to x$.

*Case* 3. The last step in the proof of $T \to x$ uses one of Rules (2) to (5), but does not introduce the main connective of $x$. Then $T \to x$ is inferred by Rule (3), (3′) or (4) from one or two sequents, one of which has the form $T' \to x$ with

$d(T') < d(T)$. The cut

$$\frac{T' \to x \quad U, x, V \to y}{U, T', V \to y}\,(6)$$

has smaller degree than the given cut. Moreover the rule which led from $T' \to x$ to $T \to x$ will also lead from $U, T', V \to x$ to $U, T, V \to x$, as may be easily verified in the different subcases.

*Case* 4. The last step in the proof of $U, x, V \to y$ uses one of Rules (2) to (5), but does not introduce the main connective of $x$. Then $U, x, V \to y$ is inferred from one or two sequents, one of which has the form $U', x, V' \to y'$. Since the inference introduces an occurrence of one connective,

$$d(U') + d(V') + d(y') < d(U) + d(V) + d(y).$$

Therefore the cut

$$\frac{T \to x \quad U', x, V' \to y'}{U', T, V' \to y'}\,(6)$$

has smaller degree than the given cut. Moreover, the same rule which led from $U', x, V' \to y'$ to $U, x, V \to y$ will lead from $U', T, V' \to y'$ to $U, T, V \to y$, as is easily verified in the different subcases.

*Case* 5. The last steps in the proofs of both premises introduce the main connective of $x = x'x'' = x' \cdot x''$. We may replace

$$\frac{\dfrac{T' \to x' \quad T'' \to x''}{T', T'' \to x'x''}\,(5) \qquad \dfrac{U, x', x'', V \to y}{U, x'x'', V \to y}\,(4)}{U, T', T'', V \to y}\,(6)$$

by

$$\frac{T'' \to x'' \qquad \dfrac{T' \to x' \quad U, x', x'', V \to y}{U, T', x'', V \to y}\,(6)}{U, T', T'', V \to y}\,(6)$$

where both new cuts have smaller degree.

*Case* 6. The last steps in the proofs of both premises introduce the main connective of $x = x'/x''$. We may replace

$$\frac{\dfrac{T, x'' \to x'}{T \to x'/x''}\,(2) \qquad \dfrac{V' \to x'' \quad U, x', V'' \to y}{U, x'/x'', V', V'' \to y}\,(3)}{U, T, V', V'' \to y}\,(6)$$

by

$$\dfrac{V' \to x'' \qquad \dfrac{T, x'' \to x' \quad U, x', V'' \to y}{U, T, x'', V'' \to y}\,(6)}{U, T, V', V'' \to y}\,(6)$$

where both new cuts have smaller degree.

*Case* 7. This last case is like Case 6, except that $x = x''\backslash x'$, and is treated symmetrically.

**10. Algebraic remarks.** The following remarks may be of mathematical interest. If we write $=$ instead of $\rightleftarrows$, the deductive system studied here becomes a partially ordered system which resembles a residuated lattice [**3**, XIII]. It may be mapped homomorphically onto a free group by mapping each element $x$ onto its congruence class modulo $\equiv$, where $x \equiv y$ means that there exists a sequence $x = x_1, \cdots, x_n = y$ $(n \geqq 1)$, such that $x_i \to x_{i+1}$ or $x_{i+1} \to x_i$ $(1 \leqq i < n)$. If this group is abelianized, we obtain something very much like the group of dimension symbols, which plays an important role in the grammar of physics.

It turns out that $x \equiv y$ if and only if

(1) $\qquad x \to t$ *and* $y \to t$ *for some* $t$,

or equivalently

(2) $\qquad z \to x$ *and* $z \to y$ *for some* $z$.

This result is easily proved by induction on the length of the given sequence connecting $x$ with $y$, once the equivalence of (1) and (2) has been established.

Assuming (1), we put

$$z = (x/((t/t)\backslash t))((t/t)\backslash y)$$

and verify (2) by computation; assuming (2), we put

$$t = (x(z\backslash z))/(y\backslash(z(z\backslash z)))$$

and verify (1) by computation.

## References

**1.** K. Ajdukiewicz, Die Syntaktische Konnexität, Studia Philosophica, vol. 1, 1935, pp. 1–27.

**2.** Y. Bar-Hillel, A quasiarithmetical notation for syntactic description, Language, vol. 29, 1953, pp. 47–58.

**3.** G. Birkhoff, Lattice Theory, New York, 1948.

**4.** L. Bloomfield, Language, New York, 1933.

**5.** R. Carnap, Logical Syntax of Language, New York, 1937.

**6.** N. Chomsky, Three models for the description of language, I.R.E. Transactions on Information Theory, vol. IT-2, 1956, pp. 113–124.

**7.** ———, Syntactic Structures, The Hague, 1957.

**8.** A. Church, A formulation of the simple theory of types, J. Symb. Logic, vol. 5, 1940, pp. 56–68.

**9.** ———, Introduction to Mathematical Logic, vol. 1, Princeton, 1956.

**10.** H. B. Curry, A Theory of Formal Deducibility, University of Notre Dame, 1950.

**11.** ———, Leçons de Logique Algébrique, Paris, 1952.
**12.** C. C. Fries, The Structure of English, New York, 1952.
**13.** G. Gentzen, Untersuchungen über das logische Schliessen, Math. Z., vol. 39, 1934, pp. 176–210, 405–431.
**14.** O. Jespersen, The Philosophy of Grammar, New York, 1924.
**15.** S. C. Kleene, Introduction to Metamathematics, New York, 1952.
**16.** W. N. Locke and A. Booth (editors), Machine translation of languages, Massachusetts Institute of Technology, 1955.
**17.** W. V. O. Quine, Mathematical Logic (rev. ed.), Cambridge, 1951.
**18.** P. Rosenbloom, The Elements of Mathematical Logic, New York, 1950.
**19.** E. Sapir, Language, New York, 1949.
**20.** M. Schönfinkel, Über die Bausteine der Mathematischen Logik, Math. Ann., vol. 92, 1924, pp. 305–316.
**21.** A. Tarski, Logic, Semantics, Metamathematics, Oxford, 1956.

# VARIABLE MATRIX SUBSTITUTION IN ALGEBRAIC CRYPTOGRAPHY

JACK LEVINE, North Carolina State College

**1. Introduction.** The use of algebraic methods in cryptography is well-known through two important papers by Hill [1], [2]. Briefly, the basic idea can be formulated in the following way. Consider the system of simultaneous congruences

$$y_i = \sum_{j=1}^{n} a_{ij}x_j \pmod{26}, \qquad i = 1, \cdots, n, \tag{1.1}$$

where the constants $a_{ij}$ are chosen so that the determinant $|a_{ij}|$ is *prime* to 26. By means of (1.1) the set of $n$ variables $(x_1, \cdots, x_n)$ is transformed to the set $(y_1, \cdots, y_n)$ and, conversely, the set $(y_1, \cdots, y_n)$ will be transformed to the unique set $(x_1, \cdots, x_n)$ by means of the inverse transformation which exists by the assumption on $|a_{ij}|$.

To each of the 26 letters of the alphabet we associate an integer from the set 0, 1, $\cdots$, 25, so that no two letters correspond to the same integer. For simplicity we illustrate with the correspondence (used throughout this paper)

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (1.2) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 0 |

Now to encipher a message, or plain text, by means of (1.1), first replace each letter of the text by means of its numerical equivalent, using for illustration, (1.2). Then divide the resulting sequence of numbers into groups containing $n$ numbers each. Call these

$$p_{11}p_{12} \cdots p_{1n} \qquad p_{21}p_{22} \cdots p_{2n} \quad \cdots \quad p_{i1}p_{i2} \cdots p_{in} \quad \cdots. \tag{1.3}$$

Each group of (1.3) is then used in (1.1) for $x_1 \cdots x_n$, and the transformed set