L. T. F. Gamut

LOGIC, LANGUAGE, AND MEANING

VOLUME I
Introduction to Logic

The University of Chicago Press
Chicago and London

The University of Chicago Press, Chicago 60637
The University of Chicago Press, Ltd., London

11 10 09 9 8 7

First published as *Logica, Taal en Betekenis*, (2 vols.) by Uitgeverij Het Spectrum, De Meern, The Netherlands. Vol. 1: *Inleiding in de logica*, vol. 2: *Intensionele logica en logische grammatica*, both © 1982 by Het Spectrum B. V.

Library of Congress Cataloging in Publication Data

Gamut, L. T. F.
[Logica, taal en betekenis. English]
Logic, language, and meaning / L. T. F. Gamut.
p. cm.
Translation of: Logica, taal en betekenis.
Includes bibliographical references.
Contents: v. 1. Introduction to logic — v. 2. Intensional logic and logical grammar.
ISBN 0-226-28084-5 (v. 1). — ISBN 0-226-28085-3 (v. 1, pbk.). — ISBN 0-226-28086-1 (v. 2) — ISBN 0-226-28088-8 (v. 2, pbk.)
1. Logic. 2. Semantics (Philosophy) 3. Languages—Philosophy. I. Title.
BC71.G33513 1991
160—dc20 90-10912
CIP

♾ The paper used in this publication meets the minimum requirements of the American National Standard for Information Sciences—Permanence of Paper for Printed Library Materials, ANSI Z39.48-1992.

L. T. F. Gamut is a collective pseudonym for **J. F. A. K. van Benthem**, **J. A. G. Groenendijk**, **D. H. J. de Jongh**, **M. J. B. Stokhof**, all senior staff members in the Institute for Logic, Language & Computation at the University of Amsterdam, and **H. J. Verkuyl**, emeritus professor at the University of Utrecht.

The University of Chicago Press
www.press.uchicago.edu

7 Formal Syntax

In this book the emphasis has been on the logical study of semantic questions. Nevertheless, the pure syntax of natural and formal languages also has an interesting structure which is accessible to treatment by mathematical methods. In this chapter we shall attempt to sketch some central notions and themes in this area, pointing out some connections with the rest of our text. There is no pretense at completeness here: for a more thorough study, the reader is referred to, e.g., Hopcroft and Ullman 1979.

7.1 The Hierarchy of Rewrite Rules

We shall be considering a finite alphabet A of symbols $a_1, \ldots, a_n$. Corresponding to this is the set A* of all finite sequences of symbols taken from A (including the 'empty sequence' ⟨⟩). A *language* L can now be seen as a subset of A* (the 'grammatical expressions of L'). If this abstract idea is applied to natural language, then, for example, words, or even whole parsed expressions, would correspond to symbols in alphabet A.

Description of a language L now amounts to finding a *grammar* G for L. Grammars are usually thought of as sets of *rewrite rules* of the form:

$X \Rightarrow E$ (Rewrite symbol X as expression E.)

Example: Let G consist of the following two rules (in the alphabet {a, b}):

$S \Rightarrow \langle\rangle$
$S \Rightarrow aSb$

The symbol S is called the 'start symbol' (which often refers to the category 'sentence'). The class of expressions generated by G consists of all sequences of the form:

$a^i b^i$ (i letters a, followed by the same number of letters b)

The sequence aabb, for example, may be obtained by means of the following rewrite steps:

S, aSb, aaSbb, aa⟨⟩bb(=aabb)

Thus, more generally, besides the *terminal symbols* in A, rewrite rules also involve *auxiliary symbols* which can be rewritten as expressions formed out of terminal and auxiliary symbols. We say that the grammar G generates the language L(G) of all strings E composed from terminal symbols only which are *derivable* from G, that is, such that there is a finite sequence of expressions starting with S and ending with E, in which every expression can be obtained from its predecessor by rewriting a single auxiliary symbol with the aid of one of the rules in G. Here is another illustration.

Example: The following grammar describes the formulas of propositional logic, with the alphabet {p, ', ¬, ∧,), (} (where propositional letters are of the form p, p', p'', . . .):

$A \Rightarrow p$
$A \Rightarrow A'$
$S \Rightarrow A$
$S \Rightarrow \neg S$
$S \Rightarrow (S \wedge S)$

Here the auxiliary symbol A stands for propositional letters or 'atomic formulas'. In fact, auxiliary symbols often correspond to grammatical categories which are also useful by themselves.

Rewrite grammars can be classified according to the kinds of rules used in them. Notably, the grammars which we have introduced so far are said to be *context-free*, which means that their rules allow the rewriting of single auxiliary symbols independently of the context in which they occur. Context-free grammars are very common and are very important.

A simpler, but still useful subspecies of this class is formed by the *regular* grammars, in which an additional requirement is placed on the expression E to the right of the arrow: it must consist of either (i) a single terminal symbol (or the empty sequence ⟨⟩) or (ii) a single terminal symbol and a single auxiliary symbol. In the latter case, all of the rules in G must have the same order: the terminal symbol must be in front (the grammar is 'left-regular') or at the end ('right-regular').

Example: Consider the alphabet {a, b} and the grammar G with the rules:

$S \Rightarrow aX$
$X \Rightarrow b$
$X \Rightarrow bS$

L(G) consists of all sequences of the form ab . . . ab.

A more realistic example of a 'language' with a regular description would be the decimal notation of numerals, like 123.654.

On the other hand there are also more complex kinds of grammars, with 'conditional' rewrite rules of the form:

$(E_1)X(E_2) \Rightarrow E$ (Rewrite X as E if it appears in the context E_1XE_2.)

One well-known example of such a *context-sensitive* grammar is the following:

$S \Rightarrow aSBC$
$S \Rightarrow abC$
$(C)B \Rightarrow D$
$C(D) \Rightarrow B$
$(B)D \Rightarrow C$
$(b)B \Rightarrow b$
$C \Rightarrow c$

The language L(G) produced by this grammar consists of all sequences of terminal symbols with equal numbers of a's, b's and c's (in that order). A derivation of aabbcc, for example, goes like this:

S, aSBC, aabCBC, aabCDC, aabBDC, aabBCC, aabbCC, aabbcC, aabbcc.

Finally, the most complex variety, *type-0* grammars, admits rules in which any expression formed out of auxiliary and terminal symbols may be rewritten as any other:

$E_1 \Rightarrow E_2$

A gradient of grammar models results for linguistic description:

regular, context-free, context-sensitive, type-0.

This is often called the *Chomsky hierarchy* after the originator of this fundamental categorization.

7.2 Grammars and Automata

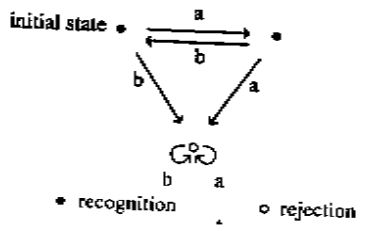
Intuitively, a grammar is a system of rules by means of which a language can be produced. But besides the 'generative' aspect of language, there is also the question of recognition: i.e., deciding whether a given sequence of symbols is an expression in the language in question or not. The latter function is often given a mathematical description in terms of machine models. Parallel to the above hierarchy of grammars, then, we have a hierarchy of recognizing machines ordered according to their 'engine power'.

The simplest recognizing machines are the *finite state automata*. These can read expressions, symbol by symbol (say, encoded on a linear tape), while always being in one of a finite number of internal states. So the behavior of such a machine is wholly determined by the following features:

(i) its 'initial state'
(ii) its 'transition function', which says which state the machine will go into, given any present state and the symbol last read

(iii) a classification of all the states as 'recognizing' or 'rejecting' (for the string read so far)

Example: The regular language consisting of all sequences of pairs ab is recognized by the following finite-state automaton:



Here exactly those strings count as 'recognized' whose processing brings the machine to an accepting state.

The correlation exhibited in this example is not a fluke. It can be proved that, given any language with a regular grammar, there is some finite-state automaton which precisely recognizes that language. And the converse also holds: for any such machine a regular grammar can be constructed generating precisely the language consisting of the expressions recognized by that machine. (For detailed definitions and arguments, the reader is referred to the literature.)

Now it could be argued that any physically realizable machine must be a (perhaps rather large) finite-state automaton. But there are other natural notions of computation too. In particular, if we are prepared to idealize away all restrictions of memory or computational cost, considering only what a human of mechanical computer could do *in principle*, then we arrive at the notion of a *Turing machine*, which realizes the most general idea of an effective procedure, or *algorithm*. Compared to a finite-state machine, a Turing machine has two extra capacities: it has a memory which is in principle unlimited, and it can apply transformations to the memory. A more concrete description is the following. The machine works on an infinitely long tape with symbols on it (initially just the string which is to be investigated). It scans this tape with its read/write head, and depending on its internal state and the symbol it has just read, it may:

(i) replace that symbol with another
(ii) shift its read/write head one position to the left or to the right
(iii) assume a different state.

Turing machines provide a very powerful and elegant analysis of effective computability in the foundations of mathematics and computer science. Even so, it is generally assumed that they are too powerful for the description of natural languages. This is connected with the following fact: the languages recognized by Turing machines are precisely those for which a type-0 grammar can be written.

There is also an intermediate kind of machine which corresponds to the above-mentioned context-free grammars between these two extremes, namely, the *push-down automaton*. This is a finite-state machine which is also capable of maintaining and using a 'stack' containing information about symbols already read in. While it is reading in symbols, and depending on its present state and whatever symbol has just been read in, a push-down automaton has the following options: it can remove the top symbol in its memory stack, it can leave this symbol untouched, or it can replace it with a new combination of symbols. The result is that linguistically relevant information can be stored in the memory and later retrieved. The following may serve as an illustration.

The language a^ib^i of strings of symbols a followed by an equal number of symbols b was generated earlier on by a context-free grammar. And in fact it cannot be recognized by a finite-state automaton, since any such machine has only a finite number of states in which it can encode the symbol patterns encountered so far. Consequently, there are always sequences whose initial segment a^i gets too long to remember, as a result of which no sufficient comparison can be made with the number of b's which are to come. A push-down automaton, however, solves the problem by storing in its stack all of the a's it reads in and then simply checking these off with the b's it reads in.

Again, a string counts as recognized by some push-down automaton if its processing drives the machine into an accepting state. There is a subtlety here, however. In general, context-free languages may need *nondeterministic* push-down automata for their recognition, which have several options for possible moves at each stage. In the latter case, a string counts as being recognizable if there exists at least one successful sequence of choices on the part of the machine leading to an accepting state after its perusal.

In more linguistic terms, a push-down automaton can deal with one 'coordination' at a distance. More than one coordination, however, cannot be performed: the earlier example of $a^ib^jc^i$ cannot be given a context-free description. It must be described by context-sensitive means. The question of whether the syntax of natural language really has such multiple coordination patterns is still a matter of continuing debate in linguistics.

7.3 The Theory of Formal Languages

The concepts discussed above have given rise to a rich general theory of languages. Once again, the reader is referred to Hopcroft and Ullman (1979), which also contains exact formulations and proofs of the results discussed in this chapter. An interesting up-to-date survey of current discussions is Savitch et al. (1987).

One important question is how specific natural languages (but also, e.g., programming languages) should be fitted into the above hierarchies. An explicit generating grammar or recognizing machine indicates a highest level of complexity at which the language must be placed, but in order to show that it could not be placed farther down we need a way to demonstrate that it cannot be described by means of some simpler kind of grammar or automaton. We have already briefly indicated one refutation method to this effect in §7.2, for regular languages as recognized by finite-state machines. The restriction to a fixed finite number of states gives rise to a result which is called the 'Pumping Lemma':

Lemma: For every regular language L there is a constant N such that if $E_1E_2E_3$ is any expression in L in which the length of E_2 is larger than N, then there are x, y, z such that E_2 is of the form xyz, and every expression of the form $E_1xy^kzE_2$ is also an expression in L (for any number k of repetitions of y).

That the earlier language a^ib^i is not regular is a direct consequence of this lemma ('pump the initial segment a^i', for $i > N$). Subtler pumping results, involving more complex duplication patterns, hold for context-free and higher languages.

A second important matter concerns the complexity of various languages. Just as we can consider the effective decidability of the laws of reasoning valid in logical systems (cf. chap. 4 above), we can also consider the decidability of the grammatical forms of expression of a language. This question can be approached by associating algorithms with the system of rewrite rules; the algorithm will check to see if any given string can be produced by means of some combination of the rules. It turns out that the membership of L(G) is decidable for grammars G up to and including context-sensitive grammars. But the languages produced by type-0 grammars are not necessarily decidable. They are in general only 'effectively enumerable': that is, we have an effective procedure for successively generating all strings belonging to the language. (Essentially one merely traces all possible derivations according to some sensible schema.) Since this will generally be an infinite process, however, it does not allow us to reject any given expression at some finite stage of the procedure: its turn might come later. This situation is analogous to one we encountered before when discussing the complexity of the valid laws in predicate logic (see §4.2). The full class of decidable languages must lie somewhere between that described by context-sensitive grammars and the full type-0 level in the Chomsky hierarchy.

This whole topic has direct practical ramifications in the parsing of linguistic expressions, with an added concern as to the efficiency with which our decision procedures can be implemented. For context-free languages, at least, parsing algorithms can be efficient: these languages can be parsed by means of an algorithm which requires no more than k^3 successive computational steps to parse an expression with k symbols. (In this connection, an independent, more finely structured hierarchy of languages ordered according to their parsing complexity can be drawn up too. The theory of the latter hierarchy is as yet fairly undeveloped.)

A third and last matter concerns investigations into families of languages which are associated with different kinds of grammars. Certain operations on the universe of all expressions in the relevant alphabet are of particular importance in this field of study. The family of regular languages is, for example, closed under all Boolean operations (corresponding to the connectives of propositional logic): intersections, unions, and complements of regular languages are regular languages. But this does not hold for context-free languages, where only closure under unions is guaranteed. One useful 'mixed' result, however, is that the intersection of a context-free and a regular language must always be a context-free language. We shall have an opportunity to apply it in §7.5.

In addition to Boolean operations, which are familiar from logic, there are many other important operations on languages with a more intrinsic syntactic flavor. For instance, given any two languages L_1, L_2, one may form their 'product' consisting of all sequences formed by concatenating a string from L_1 and one from L_2, in that order. Both regular languages and context-free languages are also closed under products of this kind.

7.4 Grammatical Complexity of Natural Languages

One of the most convincing aspects of Chomsky's classical work *Syntactic Structures* (1957) was its discussion of the complexity of natural languages. Regular and context-free grammars were successively considered as grammatical paradigms and then rejected as such. The eventually resulting model of linguistic description was the well-known proposal to make use of a context-free *phrase structure* component generating a relatively perspicuous linguistic base, with another set of rules, *transformations*, which would operate on the latter to get the details of syntax right. But around 1970, Peters and Ritchie proved that the two-stage approach has the same descriptive power as type-0 grammars, or Turing machines: something which was generally seen as combinatorial overkill. Even so, the prevalent linguistic opinion on the matter remained that the complexity of natural languages is higher than that of context-free languages.

The discussion has been revived in the last few years (for a survey, see Gazdar and Pullum 1987). It turns out, for example, that various traditional arguments for non–context freedom are formally incorrect. One favorite mathematical fallacy is that if some sublanguage of the target language L is not context-free, for example, because of the occurrence of ternary or higher patterns of coordination, then L cannot be context-free either. Other at least formally correct arguments turned out to be debatable on empirical grounds. At this moment, only a few plausible candidates are known for natural languages which are not context-free (among them Swiss German, Bambara, and Dutch).