Grammars with Structured Vocabulary: a Model for the ALGOL-68 Definition*

JOHN L. BAKER

Department of Mathematics, Statistics and Computing Science,
University of Calgary, Calgary, Alberta, Canada

Three aspects of the definition of ALGOL 68 (van Wijngaarden et al., 1969) are modelled: (1) A style of grammar (vWg) with infinitely many productions and variables, corresponding to the ALGOL-68 "syntax" is defined; (2) The passage from "strict language" to "representation language" is formalized, essentially as an inverse gsm mapping; (3) Another style of grammar (vWpg), obtained by applying the "property" idea of Stearns and Lewis (1969) to vWg, and corresponding to the ALGOL-68 syntax with "context conditions," is defined.

It is shown that being a representation language of a vW[p]g is characteristic of the recursively enumerable sets. A length-nondecreasing [and nondisappearance of indices] restriction on vW[p]g is given, and it is shown that being a representation language of a vW[p]g satisfying that restriction is characteristic of the λ-free context-sensitive languages.

GENERAL INTRODUCTION

It is not clear what society (the computing community, the sales department, the fellowship of scholars,...) expects of a "definition" of a computer programming language. It may be that the defining document for a language is expected to be an easily readable introduction, a broadside proclaiming the novel "features" of the language, a sketch of the first, best, or latest compiler, a ready reference to the commonest constructions in the language, or a literary amalgam of these and other elements. (Let us call these expectations "informal".) It may, on the other hand, be that the definition is expected to provide complete and unambiguous answers to any questions a user or

* Research sponsored in part by National Research Council of Canada (Grant A7882).
† Supported by IBM fellowship, 1969–70, at University of Washington.

Copyright © 1972 by Academic Press, Inc.
All rights of reproduction in any form reserved.
implementer may have concerning the form or interpretation of the defined
language. (Let us call this expectation "formal").

The present paper offers some tools to be used in programming-language
definition. The tools offered are extensions of the phrase-structure grammars
of Chomsky (1959). Floyd (1964, p. 351) remarks on the unlikelihood of such
extensions' retaining the "explanatory power" of phrase-structure grammars.
The present author does not feel he is contesting these remarks, or even taking
them up as a challenge. He feels rather that Floyd's remarks are based on the
great utility of phrase-structure grammars in satisfying (company policy
permitting) both formal and informal expectations; whereas the present work
is intended to help satisfy only formal expectations.

The technical content of the present paper is stated in the abstract above.
In addition to establishing some mathematical properties of the ALGOL-68
definition, this work is intended to provide substantial ground for some of the
debate on its theoretical merits and technical deficiencies.

The following two subsections give an idea of the historical and systematic
position of the ALGOL-68 definition among existing programming-language
definitions. They are biased to support the opinion that the present work is
accurate as a model for the ALGOL-68 definition. Taken together with the
motivational remarks in the body of the paper, these subsections are intended
to make further knowledge of the ALGOL-68 report superfluous to the reader
interested primarily in the abstract qualities of the definition scheme offered
here.

Syntax in the ALGOL Definitions

In the widely studied and used formulation of Chomsky (1959), the
collection of syntactic entities which are the operands of grammatical
derivations (the vocabulary for a grammar) constitute a set, with no structure.
That is to say, that formulation does not model systematic relations among
the names of syntactic entities. In the ALGOL-60 definition (Naur, 1963),
which is an application of the ideas of that formulation, such systematic
relations, for example,

\[ \langle \text{arithmetic expression} \rangle \quad \langle \text{simple arithmetic expression} \rangle \]
\[ \langle \text{designational expression} \rangle \quad \langle \text{simple designational expression} \rangle, \]

are present. Indeed, use is made of them in simplifying the description of the
meaning of the parts of the ALGOL-60 language designated by the names so
related. For example, Naur (1963, Section 3.5.3) includes "... the principle of
the evaluation [of designational expressions] is entirely analogous to that of
arithmetic expressions." Despite the discrepancy just described, it is clear that the Chomsky formulation adequately models the part of the ALGOL-60 definition labelled "syntax," since no explicit use of the structure in the ALGOL-60 vocabulary is made in its metalinguistic formulae (production rules).

How to exploit the descriptive power of a structured vocabulary, more or less within the Chomsky formulation, has been the subject of some debate in programming-language circles. The extent to which vocabulary structure should be incorporated in formal syntax has attracted particular attention, and particularly within working group 2.1 of the International Federation for Information Processing, the committee which commissioned the ALGOL-68 report (van Wijngaarden et al., 1969). Wirth and Hoare (1966) is an outgrowth of the work of that committee. It includes an avowedly modest incorporation of structured vocabulary into formal syntax, namely, the inclusion of syntactic rules like

\[
\langle T \text{ expression} \rangle ::= \langle \text{simple } T \text{ expression} \rangle,
\]

together with a statement to the effect that such a rule is an abbreviation for a (finite) set of rules

\[
\langle \text{integer expression} \rangle ::= \langle \text{simple integer expression} \rangle,
\]
\[
\langle \text{real expression} \rangle ::= \langle \text{simple real expression} \rangle,
\]
etc., obtained by consistently substituting therein "integer", "real", etc., for "\(T\)". It is again clear that the Chomsky formulation is adequate to model such a grammar, just considering it to model the unabbreviated form. Even the modest amount of structure in the vocabulary of Wirth and Hoare (1966) permits (as a matter of style) some expansion [as compared with Naur (1963)] of the role of formal syntax in the language definition. For example, in Wirth and Hoare (1966), there are only the four (unabbreviated) syntactic rules

\[
\langle \text{simple integer expression} \rangle ::= \langle \text{simple integer expression} \rangle + \langle \text{integer term} \rangle,
\]
\[
\langle \text{simple real expression} \rangle ::= \langle \text{simple integer expression} \rangle + \langle \text{real term} \rangle,
\]
\[
\langle \text{simple real expression} \rangle ::= \langle \text{simple real expression} \rangle + \langle \text{integer term} \rangle,
\]
\[
\langle \text{simple real expression} \rangle ::= \langle \text{simple real expression} \rangle + \langle \text{real term} \rangle,
\]
corresponding to both the metalinguistic formulae

\[
\langle \text{simple arithmetic expression} \rangle ::= \langle \text{simple arithmetic expression} \rangle \langle \text{adding operator} \rangle \langle \text{term} \rangle,
\]
\[
\langle \text{adding operator} \rangle ::= +
\]
and the sentence (in the corresponding “semantics”): “The type of expression will be integer if both of the operands are of integer type, otherwise real”; of Naur (1963).

The mechanism used in Wirth and Hoare (1966) for structuring the vocabulary of the grammar underlying the language definition, namely, systematic substitution within a syntactic rule to form the names of the syntactic entities involved, is used again in the ALGOL-68 report; but with the important difference that the set from which the substitutions are chosen is infinite. The names of ALGOL-68 modes (corresponding to ALGOL-60 types), for example, are formed in this way, and the set from which these names are chosen includes

- procedure-integral (mode corresponding to an integer procedure with no arguments),
- procedure-with-integral-parameter-integral (mode corresponding to an integer procedure with one integer argument),
- procedure-with-integral-parameter-and-integral-parameter-integral (mode corresponding to an integer procedure with two integer arguments).

This additional structure in the vocabulary of the ALGOL-68 grammar permits further expansion of the role of formal syntax in language definition. For example, in the valid ALGOL-68 construct

\[ ((\text{int } x, \text{int } y) \text{ int } : x \uparrow y)(2, 3) \]

(which means \(\lambda x y [x'y'](2, 3)\) and has the value 8), the agreement of actual \((2, 3)\) and formal \((x, y)\) parameters is forced by the presence in the ALGOL-68 grammar of the (unabbreviated) rule

\[
\langle \text{integral call} \rangle ::= \langle \text{firm procedure with integral parameter and integral parameter integral primary} \rangle \langle \text{actual integral parameter and integral parameter pack} \rangle
\]

and the absence of any variant of it in which the number or modes of parameters named in the two syntactic entities on the right-hand side of the rule differ.

To reiterate: the ALGOL-68 grammar consists of finitely many rules (abbreviated rules), for example,

\[
\langle M \text{ call} \rangle ::= \langle \text{firm procedure with PM primary} \rangle \langle \text{actual } P \text{ pack} \rangle
\]
[= van Wijngaarden et al. (1969, 8.6.2.1.a), in different notation]; together with a finite description of (generally infinite) sets from which substitutions are to be chosen, specifying, for example, that "integral" may be substituted for "M", "integral parameter and integral parameter" for "P"; and a statement to the effect that the production rules of the strict language (unabbreviated rules) are obtained by making such substitutions systematically.

One can see in many ways that the Chomsky formulation is not an adequate model for the grammar just described. It is the primary purpose of Section 1 of the present paper to provide an adequate mathematical model (van Wijngaarden grammar) for the ALGOL-68 production-rule syntax.

Adjuncts to Syntax

It is axiomatic in the theory of formal languages that there is no grammatical significance in the symbols in which the language itself is written, that they constitute an arbitrary set, with no structure. In the application of this theory to programming-language definition, however, some such structure does arise, due to a natural confusion between the terminal symbols of the grammar specifying a language, for example, "real" and "integer", and the characters used in preparing programs in that language for machine reading, for example, "A", "E", "G", "I", "L", "N", "R", "T". The transformation from the former to the latter can certainly be included in the grammar, by productions like

\[
\text{real} ::= \text{REAL} \\
\text{integer} ::= \text{INTEGER}.
\]

But this is undesirable for two reasons: First, there is, in fact, no grammatical significance in the transformation, so its inclusion in the grammar is uninformative. Second, since there is noticeable variation among the sets of characters to be used in connection with the machines which might be used to process a language, specification of the characters in which the language is to be written would likely inhibit sensible use of the characters available with some particular machines. On the other hand, the failure to specify this transformation permits ambiguity in the machine-ready version of the language. Consider, for example, that an ill-chosen transformation of this sort might identify the two ALGOL-60 statements

\[
\text{for } i := j \text{ do } p, \\
\text{FOR I } := \text{JDOP}.
\]
In the light of the above discussion, it is reasonable that a careful description of a programming language include, apart from the grammatical definition of the language, a specification of the transformation in question. Because the formal language-definition scheme of the present paper has intended application to the problem of programming-language definition, it includes a model (the notions representation and representation language) for such a specification. The inclusion of a model for this aspect of programming-language definition here is further justified in that the ALGOL-68 definition, which is here being modelled, explicitly distinguishes between the characters in which ALGOL-68 programs are to be written and the terminal symbols of the ALGOL-68 syntax, and uses the distinction to regularize somewhat the final stages of grammatical derivations. This aspect of the ALGOL-68 definition is described in the following paragraph.

The terminal symbols of the ALGOL-68 grammar have exactly the same form as the rest of the vocabulary, being distinguished by the fact that their names end with "symbol". For example,

\[
\langle \text{letter a symbol} \rangle, \quad \langle \text{becomes symbol} \rangle, \quad \langle \text{real symbol} \rangle
\]

are terminal symbols, while

\[
\langle \text{letter a} \rangle \quad \text{and} \quad \langle \text{actual real declarator} \rangle
\]

are not. The language consisting of strings of terminal symbols (in this sense) grammatically derivable from "\langle program \rangle" is called the strict language. The representation language consists of strings obtained from members of the strict language by any of a given set of replacements. For example, "\langle letter a symbol \rangle" must be replaced by "a"; "\langle becomes symbol \rangle" by ":=", "..="; or ".="; "\langle real symbol \rangle" by "\text{real}". Slight variations, for example "'REAL'" instead of "\text{real}" are explicitly permitted. The agreement in form between the terminal symbols and the rest of the vocabulary permits, for example, the orderly presentation

\[
\langle VM\ \text{declarator} \rangle ::= \langle M\ \text{symbol} \rangle
\]

[= van Wijngaarden et al. (1969, 7.1.1.c), in different notation], where "\(V\)" is to be replaced by "virtual", "actual", or "formal"; \(M\) by "integral", "real", etc., of the facts that each "\langle virtual integral declarator \rangle", "\langle actual integral declarator \rangle", or "\langle formal integral declarator \rangle" is to be represented by "\text{int}"; "\langle virtual real declarator \rangle", etc., by "\text{real}"; etc. (This is a "use" of this feature as mentioned in the preceding paragraph.)
Another aspect of the ALGOL-68 definition modelled in the present paper is the requirement that, in valid ALGOL-68 programs [proper programs, in the terminology of van Wijngaarden et al. (1969)], the declaration of identifiers, etc., be complete and unambiguous. This aspect of programming-language definition is discussed in Stearns and Lewis (1969), and an excellent model for it is given there. The avowed (at p. 524) purpose of that paper is to propose a method of programming language definition which places "more reliance on grammatical methods and less reliance on semantic constraints." The model (van Wijngaarden property grammar) given in the present paper continues that effort, using the structured vocabulary of the underlying (van Wijngaarden) grammar to incorporate a construction of a suitable index set [in the sense of Stearns and Lewis (1969)] into the grammar itself. In terms of the ALGOL-68 syntax, this construction corresponds to including in the index set the terminal productions of the metanotion "TAG". These are the nonterminal symbols "<letter a>" , "<letter b>" , ..., "<letter a letter a>" , "<letter a letter b>" , ..., "<letter a digit one>" , .... All ALGOL-68 derivations involving these symbols are essentially like

\[
\text{<real identifier>} \Rightarrow \text{<letter x letter y>}
\]

\[
\Rightarrow \text{<letter x><letter y>}
\]

\[
\Rightarrow \text{<letter x symbol><letter y>}
\]

\[
\Rightarrow \text{<letter x symbol><letter y symbol>},
\]

in which the last three lines yield the only terminal production of "<letter x letter y>" , and could well be regarded as auxiliary to the syntax, and the first line determines the basic properties of the index "<letter x letter y>" , namely, that its representation "xy" is, in a usual programming-language sense, a real identifier. A further justification for taking up, in the present paper, the question of defining restrictions on use of identifiers is that the ALGOL-68 definition treats it (as the context conditions) with rather more formality than is usual in programming-language definitions. In fact, van Wijngaarden et al. (1969) comments (at 4.4) that "one may consider the context conditions as syntax which is not written as production rules" and (at 4) that it "might be possible [to enforce these restrictions by means of production rules] with a more elaborate syntax."

Finally, the reader for whom these "adjuncts to syntax" are conceptually unfamiliar is referred to the description of the recognizer in Cheatham and Sattley (1964) for a portrayal of their application in syntax-directed compiling.
Formal Relations with Previous Work

The inspiration for the present work is, of course, van Wijngaarden et al. (1969). The present work provides formal models for three structures underlying ALGOL 68:

i. Its syntax in the narrowest sense. The model given here is called a van Wijngaarden grammar, and is defined in Section 1.

ii. The context conditions. These are modelled in the definition of van Wijngaarden property grammar, which is given in Section 2.

iii. The relation between the strict language and the representation language. The model given here is a relation, defined in Section 1 and redefined in Section 2, determined by a generalized sequential machine. [See Hopcroft and Ullman (1969, Chapter 9).]

Chastellier and Colmerauer (1969), which describes a natural-language machine-translation project, includes an independent formalization of the syntax of ALGOL 68. The definition given there for a \( w \)-grammar differs slightly from Definition 1, primarily in that the latter is more convenient for the present mathematical formalities.

Theorem 1 of the present work is an extension of the result of Sintzoff (1967). Theorem 2 is in the same direction as the result of Mazurkiewicz (1969), which, while dealing with more general grammars, only states a condition for recursiveness of their languages. The definition of van Wijngaarden property grammar is inspired by the definition of property grammar given by Stearns and Lewis (1969).

Notation

\( \setminus \) denotes set difference: \( A \setminus B = \{ x \in A \mid x \notin B \} \).

If \( x \) is a string, then \( |x| \) denotes its length, the number of its components.

\( \lambda \) is the empty string, the unique string of length zero.

If \( S \) is a set, then \( S^* \) denotes the set of strings whose components are elements of \( S \), and \( S^+ = S^* \setminus \{ \lambda \} \).

If \( S \) is a set and \( n \) a natural number, then \( S^n \) denotes the \( n \)-th cartesian power of \( S \), the set of ordered \( n \)-tuples of elements of \( S \). It is not assumed here that \( S^n = \{ x \in S^* \mid |x| = n \} \).

Notations and definitions not otherwise specified follow Hopcroft and Ullman (1969). Invocation of "familiar" results and techniques without specific reference may also be taken as referring to the same, in particular to Chapters 6:9.
1. van Wijngaarden Grammars

Definition of van Wijngaarden Grammar

Let us begin by stating the point of view to be taken here with respect to the formal definition of grammars with structured vocabulary. In order to remain generally within the framework of Chomsky (1959), we will say that a language is determined by derivations which are sequences of strings. The derivations are governed by a set of rules, which are, more formally, ordered pairs of strings. A point of a derivation (member of the sequence) is obtained by replacing some substring of its predecessor, which substring is the first component of a rule, by the second component of the same rule. We will use the familiar notations "\rightarrow", "\Rightarrow", and "\sim" for rules and derivations. The constituents of the strings just mentioned may be called the vocabulary for the grammar. We introduce structure into the vocabulary by requiring that its members themselves be strings. To recapitulate: Each sentential form in our style of grammar is a string, and each of its constituents is a string.

There now arises the difficulty of distinguishing (notationally) between, for example, (a) the string "wxyz" of length four and (b) the string of length two whose constituents are the strings "wx" and "yz". We meet this difficulty with a notation (already used in the Introduction) derived from Naur (1963). We denote (a) by "wxyz" and (b) by "(wx)(yz)". In general, if the constituents of a string are strings, we enclose them in angle brackets.

As an example of structured vocabulary, consider the following infinite set of rules:

\[
\begin{align*}
\{ & (s) \rightarrow (ar^n)(br^n)(cr^n) \mid n \geq 0 \\
\cup & \{ (xr^{n+1}) \rightarrow (x)(xr^n) \mid n \geq 0 \text{ and } x \in \{a, b, c\} \}. \quad (1)
\end{align*}
\]

It is easy to see that the set of strings derivable from "(s)" using these rules and including no vocabulary element with "r" or "s" as a constituent is

\[
\{ (a)^n(b)^n(c)^n \mid n > 0 \}. \quad (2)
\]

For \( n = 3 \), there is, for example, the derivation

\[
\begin{align*}
&s \\
&\langle arr\rangle \langle brr\rangle \langle crr\rangle \\
&\langle a\rangle \langle ar\rangle \langle brr\rangle \langle crr\rangle \\
&\langle a\rangle \langle a\rangle \langle a\rangle \langle brr\rangle \langle crr\rangle \\
&\ldots \\
&\langle a\rangle \langle a\rangle \langle a\rangle \langle b\rangle \langle b\rangle \langle b\rangle \langle c\rangle \langle c\rangle \langle c\rangle.
\end{align*}
\]

As indicated in the subsection "Adjuncts to Syntax," above, we regard \{<a>, <b>, <c>\} as terminal symbols for this grammar, so that they may participate fully in the structure of its vocabulary (notice, for example, that \langle ar^n \rangle does not require special treatment in case \( n = 0 \)), and we regard a transformation from, for example, "\langle a \rangle" to "a" as a matter of "representation," extrinsic to the grammar. We therefore also regard (1) as an infinite set of rules determining the language

\[
\{a^n b^n c^n \mid n > 0\}.
\]

The definition to be given for grammar with structured vocabulary must, of course, permit the finite specification of (1). We will use the method indicated in the subsection "Syntax in the ALGOL Definitions," above. Our definition will require (1) to be specified by a finite set of rule schemata, such as

\[
\langle a \rangle \rightarrow \langle aR \rangle \langle bR \rangle \langle cR \rangle \\
\langle XrR \rangle \rightarrow \langle X \rangle \langle XR \rangle,
\]

(4)

together with a statement that an actual rule to be used in a derivation (strict rule) be obtained from one of these schemata by replacing each occurrence of "R" therein by some one member of the set \{r^n \mid n \geq 0\} and each occurrence of "X" by some one member of the set \{a, b, c\}. Our definition will require that (as in this example) the left-hand side of each schema consist of a single element of the vocabulary. This requirement is in agreement with the ALGOL-68 definition.

In our definition, we must also give a finite method of specifying the sets of strings to be substituted (for "R" and "X") in the rule schemata. Our definition will require these sets to be specified by a finite set of ordinary grammatical rules, in which we regard as nonterminal symbols those (like "R" and "X") for which we are obtaining substitutions, and as terminal symbols those (like "s", "r", "a", "b", and "c") which form the strings which are constituents of sentential forms for our grammar. We call the former symbols metavariables (because they are the variables of a grammar for a language used to specify the rules of the grammar being defined, that is, variables for a "metalanguage"), and the latter symbols protovariables (because the constituents, generally variables, of our sentential forms are made up of them). We call these auxiliary rules the metarules. The set of substitutions for a given metavariable is to be the set of strings of protovariables derivable from it using
the metarules. A desirable set of metarules for the present is example is

\[ R \rightarrow \lambda \quad X \rightarrow a \]
\[ R \rightarrow R' \quad X \rightarrow b \]
\[ R' \rightarrow r \quad X \rightarrow c. \]  

(5)

A remark at the end of Section 1 of the present paper shows that the metarules may be taken to be of (Chomsky) types 1, 2, or 3 with no effect on the results obtained here, but may not reasonably be permitted to be of type 0. In the formal definition, we require only that they be of type 1. This grammatical specification of the substitutions for metavariables is in exact agreement with van Wijngaarden et al. (1969). (Since a particular language is being specified there, no general requirement on the type of metarules is given. The metarules given there are all of type 2, and some of the languages specified are not of type 3. In particular, the language for PROCEDURE includes

\(((\text{procedure with})^n \text{ real (parameter void)}^n \mid n > 0).\)

The remaining components of the formal definition prescribe starting and finishing conditions on derivations. This is done in the following familiar way: A member of the vocabulary is specified as the starting variable (\(\langle s \rangle\) in the present example), and a finite subset of the vocabulary is specified as terminal (\(\{\langle a \rangle, \langle b \rangle, \langle c \rangle\}\) in the present example). The language determined is then defined to be the set of sentential forms which have only terminal constituents and which are derivable, using the strict rules, from the starting variable [the set (2) in the present example].

In accordance with the general motivation for the present work, we require that the starting variable and the elements of the terminal vocabulary be composed only of protovariables. Although it will be clear that permitting metavariables in the starting variable or permitting a suitably restricted set of starting variables would have no effect on the results obtained here, the restriction given is in agreement with the ALGOL-68 definition, in which all members of the “strict language” are derived from program. Although the requirement that the terminal vocabulary elements be composed only of protovariables is in exact agreement with the ALGOL-68 definition, the requirement that the terminal vocabulary be finite is not. For one thing, the terminal vocabulary for ALGOL 68 is specified structurally [(van Wijngaarden et al., 1969, 1.1.2) and “Adjuncts to Syntax,” above]. For another, although finitely many terminal elements are given and used in the ALGOL-68 report
itself, it is effectively stated there (at 1.1.5.b and c) that there is no fixed terminal vocabulary. Whether this open-endedness in the ALGOL-68 definition can be tamed, and a structurally characterized, possibly infinite terminal vocabulary reasonably incorporated into a model such as ours, are questions we avoid in the present section by our requirement that the terminal vocabulary be finite.

We now proceed to the formal definitions.

First, a remark on \( \lambda \)-rules in context-sensitive grammars: We wish generally to follow Hopcroft and Ullman (1969) in permitting only \( S \rightarrow \lambda \) as a \( \lambda \)-rule in \((V, T, P, S)\), and that only if \( S \) does not occur on the right-hand side of a production in \( P \). However, we also wish to permit the grammars for the metalanguages to be substituted for distinct metavariables to have rules in common. We therefore use the following slightly modified definition: \((V, T, P, S)\) is context-sensitive if and only if

(i) if \((x \rightarrow y) \in P\), then \(|y| \geq |x|\) or \((y = \lambda\) and \(x \in V\)), and

(ii) if \((A \rightarrow \lambda) \in P\), then \(A\) does not occur on the right-hand side of any rule.

It is obvious that the class of languages determined by this definition is exactly the familiar class of context-sensitive languages.

We say \((V, T, P, S)\) is \( \lambda \)-free if and only if it includes no \( \lambda \)-rules.

**Definition 1.** A van Wijngaarden grammar (vWg) is an ordered sextuple \((M, P, Q, R, S, T)\), where

- \(M\) is a finite set, the **metavariables**.
- \(P\) is a finite set, the **protovariables**, \(M \cap P = \emptyset\).
- \(Q\) is a finite set of **metarules**, \(X \rightarrow Y\), where \(\{X, Y\} \subseteq (M \cup P)^*\), such that \(\forall_{w \in M} (M, P, Q, W)\) is a context-sensitive grammar.
- \(R\) is a finite set of **rule schemata**

\[ \langle X_0 \rangle \rightarrow \langle X_1 \rangle \cdots \langle X_k \rangle \quad (k \geq 0), \]

where \(\forall_{i=0}^{k} X_i \in (M \cup P)^*\).

\(S \in P^*\), \(S\) is the **starting variable**.

\(T \subseteq P^*\), \(T\) is a finite set, the **terminal vocabulary**.

In this paper, it is supposed that "#" is not a metavariable or protovariable of any vWg.
DEFINITION 2 (Universal assignment to metavariables). If \( M, P, Q \) are as in Definition 1, then

\[
\hat{C}(M, P, Q) = \{ c : M \to \bigcup \{ L((M, P, Q, W)) \mid W \in M \} \\
\mid \forall W \in M \ c(W) \in L((M, P, Q, W)) \},
\]

the set of all "choice functions" assigning to each metavariable an element of the language determined by it and the metarules.

Consider, for example, \( C = \hat{C}([R, R', X], \{a, b, c, r, s\}, Q_0) \), where \( Q_0 \) is the set of metarules (5), above. We have \( c \in C \) only if

\[
c(R) \in \{ r^n \mid n \geq 0 \},
\]

\[
c(R') \in \{ r^n \mid n > 0 \},
\]

and

\[
c(X) \in \{ a, b, c \}.
\]

DEFINITION 3. If \( G = (M, P, Q, R, S, T) \) is a vWg,

\[
r = ([X_0] \to [X_1] \to \cdots \to [X_k]) \in R \quad \text{and} \quad c \in \hat{C}(M, P, Q),
\]

then the strict rule of \( G \) corresponding to \( (r, c) \) is

\[
\hat{r}(r, c) = ([\hat{X}_0] \to [\hat{X}_1] \to \cdots \to [\hat{X}_k]),
\]

where each \( \hat{X}_i \) is obtained from the corresponding \( X_i \) by replacing all occurrences of each \( W \in M \) by \( c(W) \).

The set of strict rules of \( G \) is

\[
\hat{R}(G) = \{ \hat{r}(r, c) \mid r \in R \text{ and } c \in \hat{C}(M, P, Q) \}.
\]

For example, if \( G_0 = ([R, R', X], \{a, b, c, r, s\}, Q_0, R_0, \langle s \rangle, \{ \langle a \rangle, \langle b \rangle, \langle c \rangle \}) \), where \( Q_0 \) is the set of metarules (5) and \( R_0 \) the set of rule schemata (4), above, then \( \hat{R}(G_0) \) is the set (1).

DEFINITION 4. If \( G = (M, P, Q, R, S, T) \) is a vWg, then the language specified by \( G \) is

\[
L(G) = \{ X \in T^* \mid \langle S \rangle \xrightarrow{G} X \},
\]
where \( \Rightarrow_G \) is the reflexive-transitive closure of the binary relation \( \Rightarrow \) in \((P^*)^*\), which is defined by

\[
X \Rightarrow_G Y \quad \text{if and only if} \quad \exists_{\{U, V\} \subseteq (P^*)^*} X = U X' V \\
\text{and} \quad Y = U Y' V \quad \text{and} \quad (X' \rightarrow Y') \in \hat{R}(G).
\]

Consider, for example, the step

\[
\langle a \rangle \langle a \rangle \langle brr \rangle \langle crr \rangle \Rightarrow_{G_0} \langle a \rangle \langle a \rangle \langle brr \rangle \langle crr \rangle
\]

in the derivation (3). This step follows Definition 4, taking

\[
U = \langle a \rangle, \quad V = \langle brr \rangle \langle crr \rangle, \quad X' = \langle a \rangle, \quad Y' = \langle a \rangle \langle a \rangle,
\]

since

\[
(c(R) \rightarrow \langle a \rangle \langle a \rangle) = r((\langle X R \rangle \rightarrow \langle X \rangle \langle XR \rangle), c),
\]

where

\[
c(X) = a, \quad c(R) = \lambda,
\]

so that \((X' \rightarrow Y') \in \hat{R}(G_0)\). In fact, \(L(G_0)\) is the set (2), as desired.

**Definition of Representation Language**

The author believes that the "right" class of languages to study in connection with van Wijngaarden is not that determined by Definition 4, but rather its image under a suitably chosen class of mappings, to be called *representations*. One reason for this belief is given in the subsection "Adjuncts to Syntax," above, where it is suggested that the representations model a significant aspect of programming-language definition. Another reason is that the members of the languages of Definition 4 are strings of strings, and so are (technically) not comparable with arbitrary languages, about the constituents of whose members no assumption is made. Something like the notion of representation is needed to clear up this difficulty. Another reason, quite important, is that it is not apparent what relation obtains between the languages of Definition 4 and their analogues with respect to *van Wijngaarden property grammars* (determined by Definition 15, in Section 2): whereas we do obtain (in Theorem 4) a simple relation between the analogous languages determined by representations.

We will define representation here essentially as an inverse generalized
sequential machine (gsm) mapping, in the sense of Hopcroft and Ullman (1969, pp. 128 ff). The choice of a mapping defined inversely is appropriate since representation models the inverse of a certain (preliminary) step in syntax-directed compiling. [Again, see the description of the recognizer in Cheatham and Sattley (1964).] The choice of a finite-state machine seems appropriate to model the sort of work done in such a step, which generally consists of a textually local analysis of the input program. Although it is arguable that a gsm is too "powerful" to be a realistic model, the class of inverse gsm mappings is not too inclusive for our technical purposes here (the limiting factor in this direction being that classes of languages, particularly the classes of Chomsky type-0 and -1 languages, be closed under the class of mappings chosen), and we are generally seeking an inclusive definition. A technical lower limit on the "strength" of the mappings chosen is given by the construction of $D'$ in Theorem 4 (Section 2). Roughly speaking, the "strongest" requirement of this construction is, that the class chosen must include, for each integer $k$, some homomorphisms that erase $k$ consecutive symbols.

**Definition 5.** If $G = (M, P, Q, R, S, T)$ is a vWg, then the canonical representation of $G$ is a function

$$c_G : T^* \to (P \cup \{\#\})^* : \lambda \mapsto \lambda,$$

$$\langle x \rangle X \mapsto x\#c_G(X),$$

where $x \in T$ and $X \in T^*$.

For example, if $G_0$ is as above, then

$$c_{G_0}(\langle a \rangle \langle a \rangle \langle b \rangle \langle b \rangle \langle c \rangle \langle c \rangle) = a\#a\#b\#b\#c\#c\#.$$

**Definition 6.** If $G = (M, P, Q, S, T)$ is a vWg, and $E$ is a set, then a representation of $G$ (in $E$) is a gsm $D$ with input alphabet $E$ and output alphabet a subset of $P \cup \{\#\}$.

For example,

$$D_0 = (\{0\}, \{a, b, c\}, \{a, b, c, \#\}, g, 0, \{0\}),$$

where $g(0, a) = \{(0, a\#)\}$, $g(0, b) = \{(0, b\#)\}$, $g(0, c) = \{(0, c\#)\}$, (a gsm) is a representation of $G_0$ in $\{a, b, c\}$.

**Definition 7.** If $G = (M, P, Q, R, S, T)$ is a vWg, and $D$ is a representation of $G$, then the representation language determined by $(G, D)$ is

$$L(G, D) = D^{-1}(c_G(X) \mid X \in L(G)).$$
For example, $L(G_0, D_0) = \{a^n b^n c^n \mid n > 0\}$.

**Definition of Context-Sensitivity, Results**

The results of this section place the class of representation languages determined by vWg in the Chomsky hierarchy. As defined above, this class is exactly the class of type-0 languages. However, a simple structural restriction on the vWg, requiring that all strict rules be length-increasing, defines a class of representation languages coinciding with the class of $\lambda$-free type-1 languages.

**Definition 8.** A rule schema of a vWg is context-sensitive (cs) if and only if

(i) The total number of occurrences of protovariables on its left-hand side (the length of the constituent of the left-hand side, ignoring metavariables) is not greater than the total number of occurrences of protovariables on its right-hand side (the sum of the lengths of the constituents of the right-hand side, ignoring metavariables); and

(ii) The number of occurrences of each metavariable on its left-hand side is not greater than the number of occurrences of that metavariable on the right-hand side.

In symbols, let the rule schema be

$$r = \langle x_{00} W_{01} x_{01} \cdots W_{0n} x_{0n} \rangle$$

$$\rightarrow \langle x_{10} W_{11} x_{11} \cdots W_{1n_1} x_{1n_1} \rangle \cdots \langle x_{k0} W_{k1} x_{k1} \cdots W_{kn_k} x_{kn_k} \rangle,$$

where $\forall_i \forall_j x_{ij} \in P^*$ and $W_{ij} \in M$. Then $r$ is cs if and only if

(i) $| x_{00} x_{01} \cdots x_{0n} | \leq | x_{10} x_{11} \cdots x_{1n_1} \cdots x_{k0} x_{k1} \cdots x_{kn_k} |$, and

(ii) $\forall_{W \in M} \text{card}\{j \mid W_{ij} = W\} \leq \text{card}\{(i,j) \mid i > 0 \text{ and } W_{ij} = W\}$.

**Definition 9.** A vWg is context-sensitive (cs) if and only if each of its rule schemata is cs.

The following is an obvious consequence of the above definitions.

**Lemma 1.** If $G = (M, P, Q, R, S, T)$ is a cs vWg and

$$(\langle X_0 \rangle \rightarrow \langle X_1 \rangle \cdots \langle X_k \rangle) \in \hat{R}(G),$$

then $| X_0 | \leq \sum_{i=1}^{k} | X_i |$. If $\langle Y_1 \rangle \langle Y_2 \rangle \cdots \langle Y_m \rangle \Rightarrow_G \langle Z_1 \rangle \langle Z_2 \rangle \cdots \langle Z_n \rangle$, then

$\sum_{i=1}^{m} | Y_i | \leq \sum_{i=1}^{n} | Z_i |$. 


THEOREM 1. If $G$ is a $\lambda$-free $cs$ grammar, then there is a $cs$ $\forall Wg G'$ and a representation $D$ of $G'$ with $L(G', D) = L(G)$.

Proof. The idea is to construct $G'$ capable of carrying out the work of a derivation in $G$ strictly within its structured vocabulary. For example, if

$$G = (\{s\}, \{a\}, \{s \rightarrow sa, s \rightarrow \lambda\}, s),$$

we include the rule schemata

$$\langle W_1W_2 \rangle \rightarrow \langle W_1saW_2 \rangle,$$
$$\langle W_1W_2 \rangle \rightarrow \langle W_1W_2 \rangle,$$

in $G'$, along with metarules specifying $L(W_1) = L(W_2) = \{s, a\}^*$. In this way we have included (for example) strict rules sufficient to have

$$\langle s \rangle \xrightarrow{G'} \langle sa \rangle \xrightarrow{G'} \langle saa \rangle \xrightarrow{G'} \langle aa \rangle,$$

corresponding to

$$s \xrightarrow{G} sa \xrightarrow{G} saa \xrightarrow{G} aa.$$

Finally, we include in $G'$ the rule schema

$$\langle aW_3 \rangle \rightarrow \langle a \rangle \langle W_3 \rangle,$$

along with metarules specifying $L(W_3) = \{s, a\}^*$, so that we also have

$$\langle aa \rangle \xrightarrow{G'} \langle a \rangle \langle a \rangle,$$

and specify a representation

$$\langle a \rangle \langle a \rangle \rightarrow aa.$$

Formally, let $G = (V, T, R, s)$ be a $\lambda$-free $cs$ grammar. Without loss of generality, suppose that, if $(X \rightarrow Y) \in R$, then $X \notin T^*$. Let

$$\{W_1, W_2, W_3\} \cap (V \cup T) = \emptyset,$$

and define

$$G' = ([W_1, W_2, W_3], V \cup T, Q, R', \langle s \rangle, T'),$$
where

\[ Q = \{ W_3 \to uW_3 \mid u \in V \cup T \} \cup \{ W_3 \to u \mid u \in V \cup T \} \]
\[ \cup \{ W_1 \to W_3, W_2 \to W_3, W_1 \to \lambda, W_2 \to \lambda \}, \]
\[ R' = \{ \langle W_1 XW_2 \rangle \to \langle W_1 YW_2 \rangle \mid (X \to Y) \in R \} \]
\[ \cup \{ \langle tW_3 \rangle \to \langle t\langle W_2 \rangle \mid t \in T \}, \]
\[ T' = \{ \langle t \rangle \mid t \in T \}. \]

Define

\[ D = (\{ 0 \}, T, T \cup \{ \#, g \}, 0, \{ 0 \}) , \]

where

\[ \forall_{t \in T} g(0, t) = \{ (0, t\#) \} . \]

By definition, \( G' \) is a [cs] vWg and \( D \) a representation of \( G' \). It is easy to see, by induction on the length of a derivation, that, if \( X \in (V \cup T)^* \) and \( s \vartriangleright_G X \),
then \( \langle s \rangle \vartriangleright_{G'} \langle X \rangle \). It is easy to see, by induction on \( | X | \), that, if \( X \in T^+ \),
then \( \langle X \rangle \vartriangleright_{G'} X^\sim \), where \( X^\sim \) is defined by: \( \lambda^\sim = \lambda \); if \( t \in T \) and \( Y \in T^* \), then
\( (tY)^\sim = \langle t \rangle Y^\sim \). With the aid of Lemma 2 (following), it is easy to see that,
if \( X \in T^* \) and \( \langle s \rangle \vartriangleright_{G'} X^\sim \), then \( s \vartriangleright_G X \). By definition of \( D \), if \( X \in T^* \), then
\[ D(X) = \{ c_{G'}(X^\sim) \} . \]

Since \( (T^\sim)^* = \{ X^\sim \mid X \in T^* \} \), the proof is complete.

**Lemma 2.** Under the hypotheses of Theorem 1 and in the notation of its proof, if \( \langle s \rangle \vartriangleright_{G'} X \) (\( X \in ((V \cup T)^*)^* \)), then

\[ \exists_{Y \in T} \exists_{Z \in (V \cup T)^*} X = Y^\sim \langle Z \rangle \quad \text{and} \quad s \vartriangleright_{G} YZ. \]

**Proof.** By induction on the length \( n \) of a derivation \( \langle s \rangle \vartriangleright_{G'} X \).

\( n = 0 \): Then \( X = \langle s \rangle = \lambda^\sim \langle s \rangle \) and \( s \vartriangleright_{G} s = \lambda s \).

\( n > 0 \): Then \( X = YZU \) and \( \langle \langle Z_1 \rangle \to Z \rangle \in \tilde{R}(G') \) and \( \langle s \rangle \vartriangleright_{G'} Y \langle Z_1 \rangle U \)
in \( n - 1 \) steps \( \{ Y, Z, U \} \subset ((V \cup T)^*)^* \), \( Z_1 \in (V \cup T)^* \). If \( | Z_1 | = 1 \), then,
by definition of \( R' \), \( \exists_{Z_2} (Z_1 \to Z_2) \in R \), so, by hypothesis, \( Z_1 \notin T^* \). Therefore,
by inductive hypothesis, it must be that \( Y = Y_1^\sim \) (\( Y_1 \in T^* \)), \( U = \lambda \), and
\( s \vartriangleright_{G} Y_1 Z_1 \). Again, by definition of \( R' \), there are only two possibilities:
\( Z = \langle Z_2 \rangle \) (\( Z_2 \in (V \cup T)^* \)) \( \) and \( Z_1 \vartriangleright_{G} Z_3 \), in which case \( X = Y_1^\sim \langle Z_2 \rangle \) and
\( s \vartriangleright_{G} Y_1 Z_2 \); or \( Z_1 = tZ_2 \) \( \) and \( Z = \langle t \rangle \langle Z_3 \rangle \) (\( t \in T \) \( , Z_2 \in (V \cup T)^* \)), in which
case \( X = (Y_1 t)^\sim \langle Z_2 \rangle \) and \( s \vartriangleright_{G} Y_1 tZ_2 \). This completes the proof of Lemma 2.
For the proof of Theorem 1, it was convenient to permit a metavariable to specify a language including \( \lambda \). For the converse, it is not. Accordingly,

**Lemma 3.** If \( G = (M, P, Q, R, S, T) \) is a [cs] vWg, then there is a [cs] vWg \( G' = (M', P', Q', R', S', T') \) with \( \forall W \in M' \lambda \notin L((M', P', Q', W)) \) and \( L(G') = L(G) \).

**Proof.** \( \lambda \in L((M, P, Q, W)) \) if and only if \( (W \rightarrow \lambda) \in Q \). The idea then is just to define \( Q' \) obtained from \( Q \) by omitting all \( \lambda \)-rules, and \( R' \) obtained from \( R \) by adjoining new rule schemata covering all \( \lambda \)-metarule omissions. With respect to \( G_0 \), defined above, for example we define \( Q'_0 = Q_0 \setminus \{(R \rightarrow \lambda)\} \), and \( R'_0 = R_0 \cup \{(s) \rightarrow (a)(b)(c), (Xr) \rightarrow (X)(X)\} \). (\( Q_0, R_0 \) are given at (5), (4), respectively).

Formally, define

\[
M' = M; P' = P; S' = S; T' = T;
Q' = Q \setminus \{(W \rightarrow \lambda) \mid W \in M\}; \text{ and}
R' = \{r(c) \mid r \in R \text{ and } c \in C(M, P, Q)\},
\]

where, if \( r = (\langle X_0 \rangle \rightarrow \langle X_1 \rangle \cdots \langle X_k \rangle) \in R \) and \( c \in C(M, P, Q) \), then \( r(c) = (\langle Y_0 \rangle \rightarrow \langle Y_1 \rangle \cdots \langle Y_k \rangle) \), where each \( Y_i \) is obtained from the corresponding \( X_i \) by omitting all occurrences of each \( W \in M \) for which \( c(W) = \lambda \).

It is easy to see that \( \hat{R}(G') = \hat{R}(G) \), whence \( L(G') = L(G) \). Evidently \( G' \) is cs if \( G \) is.

**Theorem 2.** If \( G \) is a vWg [resp., a cs vWg] and \( D \) is a representation of \( G \), then there is a nondeterministic turing machine (tm) [resp., linear-bounded automaton (lba)] which accepts \( L(G, D) \).

**Proof.** By construction, given \( G = (M, P, Q, R, S, T) \), of a suitable acceptor. The construction is presented here informally, but with no essential ideas omitted. A more formal presentation is given in Baker (1970), using the technique of Knuth and Bigelow (1967).

The construction to be made is of a nondeterministic tm [resp., lba] \( A \), acting, by familiar techniques, on a six-track tape. In presenting the construction, it is convenient to make the following definitions:

\( c \) is a function extending the work of \( c_G \) to strings including metavariables, and to nonterminal strings, by: \( c(\lambda) = \lambda; \) if \( x \in (M \cup P)^* \) and \( X \in ((M \cup P)^*)^* \), then \( c(\langle x \rangle X) = x \# c(X) \).
is a function. If $X \in (P \cup \{\#\})^*$, then $\alpha(X)$ is the six-track string with $X$ on track 1, blanks on tracks 2-6.

$D_\alpha$ is a gsm, identical to $D$, except that $D_\alpha$ outputs $\alpha(X)$ whenever $D$ outputs $X$.

$I$ is a function. If $X$ is a six-track string, then $I(X)$ is the part on track 1.

Immediately from the definitions, $c$ is one-to-one, $I(\alpha(X)) = X$, and $D_\alpha(Y) = \{\alpha(c(X)) \mid c(X) \in D(Y)\}$. In the construction to be given,

$$A$$

accepts $L = \{X \mid I(X) \in c(L(G))\}$. (6)

By definition, $D_\alpha^{-1}(L) = \{Y \mid D_\alpha(Y) \cap L \neq \emptyset\}$. Therefore, by the present remarks, $D_\alpha^{-1}(L) = \{Y \mid \exists Z c(Z) \in c(L(G)) \text{ and } c(Z) \in D(Y)\} = L(G, D)$, and, by the familiar result that the class of languages accepted by tm [resp., lba] is closed under inverse gsm mappings, the construction of $A$ satisfying (6) is sufficient to prove the existence of the required acceptor of $L(G, D)$.

[According to Lemma 3, suppose, without loss of generality,]

$$\forall_{\omega \in M} \lambda \notin L(M, P, Q, W).]$$ (7)

$A$ produces on its track 1, given input $X$, a sequence $X_0, X_1, \ldots$, with $X_0 = I(X)$, $\forall_i c^{-1}(X_{i+1}) \Rightarrow_G c^{-1}(X_i) \text{ [and } |X_{i+1}| \leq |X_i|\]}. A accepts $X$ if and only if $c^{-1}(X_0) \in T^*$ and $\exists_n X_n = c(S)$. Therefore $L \subseteq \{X \mid I(X) \in c(L(G))\}$. $A$ produces $X_{i+1}$ from $X_i$ by selecting (nondeterministically) a rule schema $(Y \rightarrow Z) \in R$; placing $c(Y)$ on track 3 and $c(Z)$ on track 2, each left justified with $X_i$; generating (nondeterministically) a strict rule $(Y' \rightarrow Z') \in \hat{R}(G)$ from $(Y \rightarrow Z)$, with $c(Y')$ on track 3 and $c(Z')$ on track 2, each still left justified with $X_i$; shifting tracks 2 and 3 right (nondeterministically) $k$ squares ($k \geq 0$); and finally, in case $X_i = Uc(Z)V$ and $|U| = k$ and $(U = \lambda$ or $U$ ends with "\#"), replacing $X_i$ on track 1 by $X_{i+1} = Uc(Y')V$.

Tracks 4, 5, and 6 are used in the generation of $(Y' \rightarrow Z')$ from $(Y \rightarrow Z)$. (See Fig. 1.) All strict rules of $G$ which could be used in a derivation $S \Rightarrow_G c^{-1}(X_0)$ are available for selection by $A$ at each stage, since there is no restriction on its choices [resp., since the only restrictions on its choices are

$$|c(Z')| + k \leq |X_0|,$$

$$|c(Y')| + k \leq |X_0|,$$

and

$$|c(Y)| \leq |X_0|,$$

all of which follow (via (7) and Lemma 1), in case $G$ is cs, from the obvious
GRAMMARS WITH STRUCTURED VOCABULARY

START

\( e^{-1}(T.1) \in T \)

REJECT

YES

NO

T.1 = S#?

ACCEPT

YES

SELECT

\((Y \rightarrow Z) \in R\)

T.2 := \(c(Z)\)

T.3 := \(c(Y)\)

REPLACE \(u\) WITH \(T.4\) THROUGHOUT \(T.2\) AND \(T.3\)

YES

PROTO

MAYBE

NO

STRICT

YES

SELECT \(u \in M\)

NO

MATCH 1

YES

REJECT

NO

MATCH 2

NO

T.1 := \(U \cdot T.3 \cdot V\)

T.4 := \(U \cdot T.6 \cdot V\)

T.5 := \(W\)

T.6 := \(X\)

SELECT \(k \geq 0\)

SELECT \(k \geq 0\)

T.4 := \(U \cdot T.6 \cdot V\)

Fig. 1. Operation of \(A\). (See Theorem 2.) \(T.i\) denotes the nonblank contents of track \(i\).

restriction \(|c(Z')| + k \leq |X_i|\) on the applicability of \((Y' \rightarrow Z')\). Therefore, if \(S = Y_0, Y_1, \ldots, Y_n\) is a derivation in \(G\), then \(A\) can produce the sequence \(c(Y_n), c(Y_{n-1}), \ldots, c(Y_0) = c(S)\) on its track 1, given input \(a(c(Y_n))\). That is, \(\{X \mid I(X) \in c(L(G))\} \subseteq L\).

Figure 1 is another outline of the algorithm embodied in \(A\), with some further details.

Figure 2 is an example of the operation of such an algorithm corresponding to the grammar \(G_o\) for (2).
COROLLARY (main result). A language \( L \) is type-0 [resp., \( \lambda \)-free type-1] if and only if there is a \( vWg \) [resp., a cs \( vWg \)] \( G \) and a representation \( D \) of \( G \) with \( L = L(G, D) \).

Proof. By familiar results, directly from Theorems 1 and 2.

---

**Fig. 2.** Tape of \( A \) (See Theorem 2.) Constructed for \( G_0 \), accepting \( \langle a \rangle \langle a \rangle \langle b \rangle \langle b \rangle \langle c \rangle \langle c \rangle \).
Remark. The proofs of Theorems 1 and 2 show that there is no distinction among types \{3, 2, 1\} of their metarules. That is, if Definition 1 required each \((M, P, Q, W)\) to be a regular grammar or to be a context-free grammar, then the proofs given for Theorems 1 and 2 would still be valid, and so the main result still true.

On the other hand, no reasonable restriction could force \(L(G, D)\) to be cs if the metarules of \(G\) were permitted to be type-0, as the following construction shows:

Let \(G = (V, T, P, S)\) be any grammar. Let \(\{s, W\} \cap (V \cup T) = \emptyset\), and suppose, without loss of generality, \((X \rightarrow Y) \in P \Rightarrow X \notin T^*\). Define

\[
G' = (V \cup \{W\}, T \cup \{s\}, Q, R, \langle s \rangle, \{\langle t \rangle \mid t \in T\}),
\]

where

\[
Q = P \cup \{W \rightarrow tW \mid t \in T\} \cup \{W \rightarrow t \mid t \in T\},
\]

\[
R = \{\langle s \rangle \rightarrow \langle S \rangle\} \cup \{\langle tW \rangle \rightarrow \langle t \rangle \langle W \rangle \mid t \in T\}.
\]

The derivations in \(G'\) consist exactly of the sequences

\[
\langle s \rangle \Rightarrow_{G'} \langle t_1 t_2 \ldots t_n \rangle \Rightarrow_{G'} \langle t_1 \rangle \langle t_2 \ldots t_n \rangle \Rightarrow_{G'} \cdots \Rightarrow_{G'} \langle t_1 \rangle \langle t_2 \rangle \cdots \langle t_n \rangle,
\]

where \(t_1 t_2 \ldots t_n \in L(G)\). As usual, define

\[
D = (\{0\}, T, T \cup \{\#\}, g, 0, \{0\}),
\]

where

\[
\forall t \in T \ g(0, t) = (0, t\#).
\]

Then \(L(G', D) = L(G)\).

2. van Wijngaarden Property Grammars

Definition of van Wijngaarden Property Grammar

In outline, this section will be exactly parallel to Section 1. The point of view given at the beginning of Section 1 with respect to grammars with structured vocabulary will be maintained in this section. The constituents of the sentential forms of the grammars to be defined here will, as in Stearns and Lewis (1969), have two components. The first component is a usual sort of constituent of a sentential form, to wit, a grammatical name, like "block" or
“program”, for the portion of a terminal string the constituent dominates in a derivation of which the sentential form is part. The second component is a table giving the properties which the elements of a specified index set have in the string dominated by the constituent, considered as an instance of the grammatical entity named by the first component. For example, such a table might specify that the only identifier “used” in a block considered as such, that is, free in the block, is “C”, and that it is used there as a label.

Let us proceed to an example, given in notation used by Stearns and Lewis (1969). The example, which will be developed and used throughout this section, has to do with a crude sort of assembly language. The opcodes are not distinguished, but are all denoted grammatically by “(o)”, in representation by “op”. The identifiers (“i”) are strings of “x”s, the constants (“c”) strings of “z”s. The grammatical names for identifier-use and -definition are, respectively, “u” and “d”. “(p)" is the starting variable; “(s)" is another variable (“program”; “sequence”).

The rule schemata of what will be called the underlying vWg are

\[
\begin{align*}
(p) & \rightarrow (s) \\
(s) & \rightarrow \lambda \\
(s) & \rightarrow (s)(KU) \\
(d) & \rightarrow (i) \\
(u) & \rightarrow (o)(i) \\
(cu) & \rightarrow (o)(c)
\end{align*}
\]

and its metarules

\[
\begin{align*}
K & \rightarrow \lambda \\
K & \rightarrow c \\
U & \rightarrow u \\
U & \rightarrow d.
\end{align*}
\]

The properties to be possessed by the identifiers are

0—neutral (meaning the identifier is invisible in the construct);
1—positive (meaning the identifier is merely present);
2—defined.

As in Stearns and Lewis (1969), the rule schemata have vector sets associated with them. For example,

\[
\begin{align*}
(s) & \rightarrow (s)(KU) \\
2 & \ 0 \ 2 \\
2 & \ 2 \ 0 \\
2 & \ 2 \ 1.
\end{align*}
\]

This system may be understood: A sequence \( s_1 \) consists of a sequence \( s_2 \) followed by a construct \( K \), which is an (identifier-) definition or an (identifier-)
use or a constant-use. (Notice there are no productions for \( <\text{cd} > \), so that we may omit constant-definition as a possibility.) It is permissible for an identifier to be defined in \( s_2 \) or \( K \), and invisible in the other, or to be defined in \( s_2 \) and merely present in \( K \). In any of these cases, such an identifier is said to be defined in \( s_1 \). In assembly language terms, the absence of a right-hand side \((2, 2)\) vector prohibits multiple definitions, and the absence of \((1, 2)\) prohibits forward references. An example of a rule (what will be called an extended strict rule) based on (10) is

\[
S (x \ x x 2) \rightarrow S (x \ x x 2) u (x \ x x 0) \quad (11)
\]

which may be understood: A sequence in which \( x \) and \( xx \) are defined may consist of a sequence in which \( x \) and \( xx \) are defined, followed by a use of \( x \).

As suggested in the subsection "Adjuncts to Syntax," above, we wish to treat identifiers (indices, more generally) as terminals, leaving their "spelling" as a matter of representation. At the same time, we wish to have the index set, which will possess the properties, specified grammatically. We accomplish both these ends in two steps. First, we incorporate sets of indexing rule schemata and indexing metarules into the grammar. In our example, these are

\[
<i> \rightarrow <T > \quad <c> \rightarrow <N > , \quad (12)
\]

and

\[
T \rightarrow x \quad T \rightarrow xT \quad N \rightarrow z \quad N \rightarrow zN , \quad (13)
\]

respectively. The right-hand sides of indexing rule schemata are required to have just one component, and each of those components is required to be a single metavariable. The set of indices is then defined to be the union of the languages determined by the indexing metarules with the metavariables just named as starting symbols. In the example, these indices are the identifiers and constants of our crude assembly language.

Second, we require the terminal elements of the present structured vocabulary to be of a form which relates the indices as property-bearers to the indices as grammatical elements. We require the terminals to be either of the form \((w, t_w)\), where \( w \) is in the terminal vocabulary of the underlying vWg and \( t_w \) is a table which specifies that each index has the neutral property; or of the form \((w, t_w)\), where \( w \) is an index and \( t_w \) is a table which specifies that \( w \) has
the positive property and all other indices have the neutral property. In addition to (10), the vector sets for the rules of our example are

\[
\begin{align*}
\langle p \rangle & \rightarrow \langle s \rangle & \langle d \rangle & \rightarrow \langle i \rangle & \langle u \rangle & \rightarrow \langle o \rangle \langle i \rangle \\
0 & & 2 & & 2 & 1 & 1 & 0 & 1 \\
\langle i \rangle & \rightarrow \langle T \rangle & \langle c \rangle & \rightarrow \langle N \rangle \\
1 & & 1 & & 0 & & 1.
\end{align*}
\]

The vector sets, taken all together, are considered to be the values of a function, the \textit{table transition function}, whose domain is the set of rule schemata.

Figure 3 is a tree portraying a complete derivation, in our example grammar, of a string which, in a suitable representation (given in a following subsection), corresponds to

\[
\begin{align*}
x & \mathbf{op} & z \\
xx & \mathbf{op} & x \\
\mathbf{op} & xx.
\end{align*}
\]

Only the portion of the tables giving the properties for “x”, “xx”, and “z” is shown in Fig. 3. All other indices have the neutral property throughout. Notice that the terminals are of the required form (“\langle o \rangle” is terminal in the underlying grammar), and that the table for the starting vocabulary element is \(t_e\). The latter is a general requirement of the grammars we are defining. Notice also that it is always permissible for an index to have the neutral property with respect to all components of a rule.

There are two important differences between the sort of grammar defined here and that of Stearns and Lewis (1969): The underlying grammar here is a vWg, and the index set here is effectively generated and is included in the terminal vocabulary. Nevertheless, the main idea of the two definitions is the same, and the reader is urged to consult Stearns and Lewis (1969) for further examples and motivation.

The system of definitions given in the present section and the specification of the \textit{ALGOL-68 context conditions} each provide a mechanism for contextual restrictions on the occurrences of indices (identifiers and indicants, in \textit{ALGOL 68}) in derivations. It is common to the two mechanisms that each is defined in terms of an underlying production-rule syntax and that each treats its indices independently of one another. The author believes that these points of similarity are definitive, so that the present work may fairly be said to include a faithful model for the \textit{ALGOL-68 context conditions}. Even if the
present model be admitted to be faithful in principle, however, it must still be seen as incomplete: In terms of the model, ALGOL 68 certainly requires at least one distinct property for each of its infinitely many modes, whereas we define the property set to be finite. For example, we must admit that our style of grammar cannot, in an obvious way, force consistent use of procedure identifiers and agreement of actual and formal parameters for the procedures identified, whereas the ALGOL-68 syntax, including context conditions, is able to do just that.

There may be a useful analogy between the incompleteness just described
(the need for an infinite property set) and the incompleteness of the system of definitions of Section I discussed just above Definition 1 (the need for an infinite terminal vocabulary). The reader will notice that the present system of definitions has satisfied that need, by including the indices in the terminal vocabulary. In compiling terms, the infinite terminal vocabulary has been enabled by relegating to the recognizer the task of locating and distinguishing the identifiers in a program. [Again see Cheatham and Sattley (1964).] Perhaps an infinite property set could be enabled by also relegating to the recognizer the task of assigning suitable properties within a program. The author believes that it is thoroughly worthwhile to search for a system of definitions extending the present one and incorporating a grammatical generation of a possibly infinite property set.

We proceed to the formal definitions.

**Definition 10.** A *van Wijngaarden property grammar* (vWpg) is an ordered quadruple \((G_U, A_I, A_P, J)\) where

\[
G_U = (M, P, Q, R, S, T) \quad \text{is a vWg, the underlying vWg.}
\]

\[
A_I = (M_I, P_I, Q_I, R_I) \quad \text{is an ordered quadruple, the indexing additions (to } G_U), \text{ where}
\]

- \(M_I\) is a finite set, the indexing metavariables, \(M_I \cap M = \emptyset\);
- \(P_I\) is a finite set, the indexing protovariables, \(P_I \cap P = \emptyset\);
- \(Q_I\) is a finite set of indexing metarules, \(X \rightarrow Y\), where \(\{X, Y\} \subseteq (M_I \cup P_I)^*\) and \(\forall w \in M_I, \lambda \notin L((M_I, P_I, Q_I, W))\);
- \(R_I\) is a finite set of indexing rule schemata \(\langle x \rangle \rightarrow \langle W \rangle\), where \(x \in P^*\) and \(W \in M_I\); and

\[
(M \cup M_I, P \cup P_I, Q \cup Q_I, R \cup R_I, S, T) \text{ is a vWg.}
\]

\[
A_P = (F, e_0, e_1) \quad \text{is an ordered triple, the property additions (to } G_U), \text{ where}
\]

- \(F\) is a finite set, the properties;
- \(e_0 \in F\), \(e_0\) is the neutral property;
- \(e_1 \in F\), \(e_1\) is the positive property, \(e_1 \neq e_0\).
is the table transition function, \( \text{dom}(J) = R \cup R_1 \),
if \( r = (\langle X_0 \rangle \rightarrow \langle X_1 \rangle \cdots \langle X_k \rangle) \in R \cup R_1 \), then \( J(r) \subset F^{k+1} \).

Our example may be formally defined as \( G_1 = (G_{u1}, A_{I1}, A_{P1}, J_1) \), where

\[
G_{u1} = (\{K, U\}, \{p, s, d, u, o, i, c\}, Q_1, R_1, \langle p \rangle, \{\langle o \rangle\}), \text{where } Q_1 \text{ is given at (9)}, \text{and } R_1 \text{ is given at (8)},
\]

\[
A_{I1} = (\{T, N\}, \{x, z\}, Q_{I1}, R_{I1}), \text{where } Q_{I1} \text{ is given at (13)}, \text{and } R_{I1} \text{ is given at (12)},
\]

\[
A_{P1} = (\{0, 1, 2\}, 0, 1),
\]

\( J_1 \) is given at (10) and (14). For example, (10) is, in formal terms,
a statement that

\[
J_1(\langle s \rangle \rightarrow \langle s \rangle \langle KU \rangle) = \{(2, 0, 2), (2, 2, 0), (2, 2, 1)\},
\]

while the absence of any vector list for \( \langle cu \rangle \rightarrow \langle o \rangle \langle c \rangle \) means that \( J_1(\langle cu \rangle \rightarrow \langle o \rangle \langle c \rangle) = \emptyset \).

Notice that the languages determined by the two vWg mentioned in
Definition 10 are, in the case of \( G_1 \), empty. This is typical, and a result of the
indices' not being included as terminals in this definition. Requiring that the
two sextuples in question be vWg is merely an economical way to establish
the structural characteristics of their constituents.

**Notation.** If \( G \) is a vWpg as in Definition 10 and \( W \in M \cup M_1 \), then
\( L_W = L((M \cup M_1, P \cup P_1, Q \cup Q_1, W)) \).

**Definition 11.** If \( G \) is a vWpg as in Definition 10, then the set of indices
for \( G \) is

\[
I(G) = \bigcup \{L_W | \exists_{r \in R_1} r = (\langle x \rangle \rightarrow \langle W \rangle)\}.
\]

For example, \( I(G_1) = L_T \cup L_N = \{x^i | i > 0\} \cup \{z^i | i > 0\} \).

**Definition 12.** If \( G \) is a vWpg as in Definition 10, the set of tables of \( G \) is

\[
H(G) = \{t : I(G) \rightarrow F \mid \text{t}^{-1}(F \setminus \{e_0\}) \text{ is finite}\},
\]

the set of all functions assigning properties to indices which assign the neutral
property to all but finitely many indices.
**Notation.** If $G$ is a vWpg as in Definition 10, and $w \in I(G)$, then $t_w$ denotes the table of $G$ which assigns $w$ the positive property $e_1$ and all other indices the neutral property $e_0$. The table assigning all indices the neutral property is denoted $t$.

**Definition 13.** If $G$ is a vWpg as in Definition 10, then the set of extended variables of $G$ is $P_E^*(G) = (P^* \cup P^*_t) \times H(G)$. (The """"*"""" is part of the symbol """"$P_E^*$"""".) The set of extended terminals of $G$ is

$$T_E(G) = (T \times \{ t \}) \cup \{ (w, t_w) \mid w \in I(G) \}. \quad (T_E(G) \subseteq P_E^*(G)).$$

**Definition 14.** If $G$ is a vWpg as in Definition 10, then the set of extended strict rules of $G$ is

$$\hat{R}_E(G) = \{ (x_0, t_0) \rightarrow (x_1, t_1) \cdots (x_k, t_k) \mid \exists r \in R, \exists c \in C(M \cup M_t, P \cup P_t, \cup \cup G) \} \subseteq \{ (x_0) \rightarrow \langle x_1 \rangle \cdots \langle x_k \rangle \} = \hat{r}(r, c)$$

and $\forall_{i=0}^k t_i \in H(G)$

and $\{ (t_0(w), t_1(w), \ldots, t_k(w)) \mid w \in I(G) \} \subseteq J(r) \cup \{ e_0 \}^{k+1}$.

(Definitions for $\hat{r}$ and $\hat{C}$ are Definitions 3 and 2, respectively.) Let us consider in detail, for example, (11) in the notation of Definition 14:

$k = 2, x_0 = x_1 = s, x_2 = u$.

Choosing

$r = (\langle s \rangle \rightarrow \langle s \rangle \langle KU \rangle); c : K \rightarrow \lambda, U \rightarrow u$;

we have

$$\langle x_0 \rangle \rightarrow \langle x_1 \rangle \langle x_2 \rangle = \langle x_0 \rangle \rightarrow \langle s \rangle \langle u \rangle = \hat{r}(r, c).$$

$$t_0(x) = t_0(xx) = t_1(x) = t_1(xx) = 2,$n

$$t_2(x) = 1, \quad t_2(xx) = 0, \quad t_0(w) = t_1(w) = t_2(w) = 0, \quad \text{if } w \in I(G) \setminus \{ x, xx \}.$$

Thus $\forall_{i=0}^2 t_i \in H(G_1)$.

$$\{(t_0(w), t_1(w), t_2(w)) \mid w \in I(G_1)\} = \{(t_0(x), t_1(x), t_2(x)), (t_0(xx), t_1(xx), t_2(xx))\}$$

$$\cup \{(t_0(w), t_1(w), t_2(w)) \mid w \in I(G_1) \setminus \{ x, xx \}\} = \{(2, 2, 1), (2, 2, 0), (0, 0, 0)\}$$

$$\subseteq \{(2, 0, 2), (2, 2, 0), (2, 2, 1)\} \cup \{0, 0, 0\} = J_1(r) \cup \{0\}^2.$$

Thus (11) is an element of $\hat{R}_E(G_1)$. 

DEFINITION 15. If $G$ is a vWpg as in Definition 10, then the extended language specified by $G$ is

$$L_{E}(G) = \{X \in (T_{E}(G))^{*} | (S, t_{s}) \xrightarrow{G} X\},$$

where $\xrightarrow{G}$ is the reflexive-transitive closure of the binary relation $\Rightarrow_{G}$ in $(P_{E}^{*}(G))^{*}$, which is defined by

$$X \xrightarrow{G} Y \quad \text{if and only if} \quad \exists \{v. vIc<e\ast_{E}(a)^{*} X \Rightarrow UX'V \quad \text{and} \quad Y = UY'V \quad \text{and} \quad (X' \rightarrow Y') \in \bar{R}_{E}(G).$$

The language specified by $G$ is

$$L(G) = \{\langle x_{0}\rangle \langle x_{1}\rangle \cdots \langle x_{m} \rangle | \exists \{t_{0}, t_{1}, \ldots, t_{m}\} \in H(g) (x_{0}, t_{0})(x_{1}, t_{1}), \ldots, (x_{m}, t_{m}) \in L_{E}(G)\}.$$ 

For example, the derivation of Fig. 3 shows that

$$\langle x \rangle \langle o \rangle \langle z \rangle \langle xx \rangle \langle o \rangle \langle x \rangle \langle o \rangle \langle xx \rangle$$

is an element of $L(G_{1})$.

**Definition of Representation Language**

The (nontechnical) motivation for the notion of representation as applied to vWpg is the same as for vWg. Both this motivation and the technical utility of the definitions of the present subsection are discussed in the analogous subsection of Section 1. The definitions given here are, in fact, merely adaptations to vWpg of Definitions 5, 6, and 7.

DEFINITION 16. If $G$ is a vWpg as in Definition 10, then the canonical representation of $G$ is a function

$$c_{G} : (T \cup I(G))^{*} \rightarrow (P \cup P_{I} \cup \{\#\})^{*}$$

: $\lambda \mapsto \lambda,$

$$\langle x \rangle \mapsto x \# c_{G}(X),$$

where $x \in T \cup I(G)$ and $X \in (T \cup I(G))^{*}$.

DEFINITION 17. If $G$ is a vWpg as in Definition 10, and $E$ is a set, then
a representation of $G$ (in $E$) is a gsm $D$ with input alphabet $E$ and output alphabet a subset of $P \cup P_1 \cup \{\#\}$.

For example,

$$D_1 = ([0, 1], \{\text{nl, op, x, z}\}, \{0, x, z, \#\}, g, 0, \{0\}),$$

where $g$ is given by

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>nl</td>
<td>\emptyset</td>
<td>{(0, #)}</td>
</tr>
<tr>
<td>op</td>
<td>{(1, o#)}</td>
<td>{(1, #o#)}</td>
</tr>
<tr>
<td>x</td>
<td>{(1, x)}</td>
<td>{(1, x)}</td>
</tr>
<tr>
<td>z</td>
<td>\emptyset</td>
<td>{(1, z)},</td>
</tr>
</tbody>
</table>

is a representation of $G_1$ in $\{\text{nl, op, x, z}\}$. If we assume that each line of (15) is followed by "\text{nl}" ("new line"), call the resulting string $X$, and denote by $Y$ the string given at (16), then it is true that

$$D_1(X) = \{x\#o\#z\#xx\#o\#x\#o\#xx\#\} = \{c_G(Y)\}.$$

**Definition 18.** If $G$ is a vWpg as in Definition 10, and $D$ is a representation of $G$, then the representation language determined by $(G, D)$ is

$$L(G, D) = D^{-1}(\{c_G(X) \mid X \in L(G)\}).$$

For example, the remarks following Definitions 15 and 17 show that (15) (including the three implicit "\text{nl}"s) is an element of $L(G_1, D_1)$.

**Definition of Context-Sensitivity, Results**

The results of this section place the class of representation languages determined by vWpg in the Chomsky hierarchy. As defined above, this class is exactly the class of type-0 languages. However, a simple structural restriction on the vWpg defines a class of representation languages coinciding with the class of $\lambda$-free type-1 languages. The restriction requires that all strict rules of the underlying vWg be length-increasing and that, if an index has a nonneutral property in an extended variable, then that index has a nonneutral property somewhere in each string descended from that extended variable in a derivation. (Indices do not become invisible, reading top-down.)
Definition 19. A vWpg \((G_U, A_I, A_p, J)\) is context-sensitive (cs) if and only if

(i) \(G_U\) is a cs vWg (Definition 9) and

(ii) if \(r = (X_0 \rightarrow X_1 \cdots X_k)\) is a rule schema of \(G_U\) or \(A_I\), then

\[ \forall (p_0, p_1, \ldots, p_k) \in J(r) \quad p_0 \neq e_0 \Rightarrow \exists i=1 \ldots k \quad p_i \neq e_0. \]

That is, each vector associated with a rule schema either has a neutral first component or has some nonneutral component other than the first.

Theorem 3. If \(G\) is a \([cs]\) vWg and \(D\) is a representation of \(G\), then there is a \([cs]\) vWpg \(G'\) with \(D\) a representation of \(G'\) and \(L(G') = L(G)\).

Proof. The idea is to construct \(G'\) just like \(G\), with an inactive property apparatus.

Let \(G = (M, P, Q, R, S, T)\) be a [cs] vWg, \(D\) a representation of \(G\). Define \(G' = (G, (\emptyset, \emptyset, \emptyset, \emptyset), \{(0, 1), (0, 1), J\}\), where \(\forall r \in R J(r) = \emptyset\). \(G'\) is a [cs] vWpg and \(D\) is a representation of \(G\). \(I(G') = \emptyset\), so the only element of \(H(G')\) is a function with empty domain, and this is, for \(G', t_e\).

\[ \hat{R}_E(G') = \{(x_0, t_e) \rightarrow (x_1, t_e) \cdots (x_k, t_e) \mid (x_0, t_e) \cdots (x_k, t_e) \in \hat{R}(G)\}. \]

Therefore \((S, t_e) \in G' (x_0, t_e) \cdots (x_n, t_e)\) if and only if \(S \rightarrow_G x_0 \cdots x_n\). Furthermore, \(T_E(G') = T \times \{t_e\}\). Thus \(L(G') = L(G)\).

An analogue of Lemma 3 is useful in the proof of Theorem 4.

Lemma 4. If \(G = ((M, P, Q, R, S, T), A_I, A_p, J)\) is a \([cs]\) vWpg, then there is a \([cs]\) vWpg \(G' = ((M', P', Q', R', S', T'), A_I', A_p', J')\) with

\[ \forall \lambda \in \lambda \not\in L((M', P', Q', W)) \quad \text{and} \quad L_E(G') = L_E(G). \]

Proof (analogous to that of Lemma 3). Define

\[ M' = M; \quad P' = P; \quad S' = S; \quad A_I' = A_I; \quad A_p' = A_p; \]

\[ Q' = Q \{(W \rightarrow \lambda) \mid W \in M\}; \]

\[ R' = \{\bar{r}(r, c) \mid r \in R \text{ and } c \in \hat{C}(M, P, Q)\}, \]

where, if \(r = (X_0, X_1, \ldots X_k) \in R \text{ and } c \in \hat{C}(M, P, Q), \text{ then } \bar{r}(r, c) = (Y_0, Y_1, \ldots Y_k), \text{ where each } Y_i \text{ is obtained from the corresponding } X_i \text{ by omitting all occurrences of each } W \in M \text{ for which } c(W) = \lambda; \text{ and } \]

\(J'\) determined as an extension of \(J\) by \(J'(\bar{r}(r, c)) = J(r)\).
It is easy to see that $\hat{R}_E(G') = \hat{R}_E(G)$, whence $L_{\hat{E}}(G') = L_{\hat{E}}(G)$. Evidently $G'$ is cs if $G$ is.

For example, the above construction applied to $G_1$ gives rise to $G_2 = (G_{U_2}, \mathcal{A}_{I_2}, \mathcal{A}_{P_2}, \mathcal{J}_2)$, where

$$G_{U_2} = ([K, U], \{p, s, d, u, o, i, c\}, \mathcal{Q}_2, \mathcal{R}_2, \langle p \rangle, \langle \langle o \rangle \rangle),$$

where

$$\mathcal{Q}_2 = \{K \rightarrow c, U \rightarrow u, U \rightarrow d\},$$

and

$$\mathcal{R}_2 = \{\langle p \rangle \rightarrow \langle s \rangle, \langle s \rangle \rightarrow \lambda, \langle s \rangle \rightarrow \langle s \rangle \langle U \rangle, \langle s \rangle \rightarrow \langle s \rangle \langle KU \rangle, \langle d \rangle \rightarrow \langle i \rangle, \langle u \rangle \rightarrow \langle o \rangle \langle i \rangle, \langle cu \rangle \rightarrow \langle o \rangle \langle c \rangle\},$$

$$\mathcal{A}_{I_2} = ([T, N], \{x, z\}, \mathcal{Q}_{I_2}, \mathcal{R}_{I_2}),$$

where

$$\mathcal{Q}_{I_2} = \{T \rightarrow x, T \rightarrow xT, N \rightarrow z, N \rightarrow zN\},$$

$$\mathcal{R}_{I_2} = \{\langle i \rangle \rightarrow \langle T \rangle, \langle c \rangle \rightarrow \langle N \rangle\},$$

$$\mathcal{A}_{P_2} = \{\{0, 1, 2\}, 0, 1\},$$

and

$\mathcal{J}_2$ is given in the following display:

$$\begin{array}{cccccccc}
\langle p \rangle & \rightarrow & \langle s \rangle & \langle s \rangle & \rightarrow & \langle s \rangle \langle U \rangle & \langle s \rangle & \rightarrow & \langle s \rangle \langle KU \rangle \\
0 & 2 & 2 & 0 & 2 & 2 & 0 & 2 \\
2 & 2 & 0 & 2 & 2 & 0 & \\
2 & 2 & 1 & 2 & 2 & 1 & \\
\langle d \rangle & \rightarrow & \langle i \rangle & \langle u \rangle & \rightarrow & \langle o \rangle & \langle i \rangle & \rightarrow & \langle T \rangle & \langle c \rangle & \rightarrow & \langle N \rangle \\
2 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1.
\end{array}$$

**Theorem 4.** If $G$ is a [cs] vWpg and $D$ is a representation of $G$, then there is a [cs] vWg $G'$ and a representation $D'$ of $G'$ with $L(G', D') = L(G, D)$.

**Proof.** By construction, given $(G, D)$, $G$ as in Definition 10, of $(G' = (M', \mathcal{P}', \mathcal{Q}', \mathcal{R}', S', T'), \mathcal{D}')$ as required. We assume that $Q$ includes no $\lambda$-rules, without loss of generality, by Lemma 4. The construction, the ideas motivating it, and an application to $(G_2, D_1)$ are given here, but no proof of its correctness. A formal proof, based on this construction, is given in Baker (1971).

The $G'$ constructed will, of course, do exactly the work of $G$. There are
two apparent difficulties in the construction of $G'$, and a third problem that arises in connection with the solution of the first two: First, $G'$ must mimic $G$ with respect to the handling of the indices and their properties. Second, $G'$ must suitably dispose of the infinite terminal vocabulary of $G$. The solution given for these two difficulties involves providing the "variables" (strings of protovariables which actually occur as components of steps in a derivation) of $G'$ with a structure incorporating the index-and-property information of $G$, together with enough markers to make the structure tractable. The third problem is to dispose of these markers in a manner generally consistent with the context-sensitive restrictions which may apply to $G'$. The solution of this third problem is obtained in the definition of $D'$, which inserts (recall that the relation of $D'$ to a derivation in $G'$ is inverse) the markers suitably.

Figure 4 is a tree portraying a derivation in $G_2'$, obtained from $G_2$ by the present construction. The derivation of Fig. 4 corresponds exactly to that of Fig. 3.

We first consider the problem of constructing $G'$ to handle indices and their properties as $G$ does.

The idea, in solving this problem, is, given an extended variable $v$ of $G$, to make the corresponding "variable" of $G'$ include, for each nonneutral property $p$, a list of the indices which are assigned $p$ by the table of $v$. Consider, for example, the strict rule (11) of $G_2$. Our construction will include, as a strict rule of $G_2'$,

$$\langle as12xx*x*\rangle \rightarrow \langle bs12xx*x*\rangle \langle bu1x*2\rangle$$

(17)

corresponding to (11). (The role of "a" and "b" will appear below). Rule (11) is based on the rule schema and table transition

$$\langle s \rangle \rightarrow \langle s \rangle \langle U \rangle$$

$$2 \quad 0 \quad 2$$

$$2 \quad 2 \quad 0$$

$$2 \quad 2 \quad 1$$

(18)

of $G_2$. We obtain (17) by including in $G_2'$ a rule schema

$$\langle as12U_1U_2U_3 \rangle \rightarrow \langle bs12U_2U_3 \rangle \langle bU1U_2U_1 \rangle$$

(19)

corresponding to (18), and metarules (in addition to those of $G_2$)

$$U_1 \rightarrow U_p \quad U_1 \rightarrow \lambda \quad U_p \rightarrow V_1$$

$$U_2 \rightarrow U_p \quad U_2 \rightarrow \lambda \quad U_p \rightarrow V_1U_p$$

$$U_3 \rightarrow U_p \quad U_3 \rightarrow \lambda \quad V_1 \rightarrow T*$$

$$V_1 \rightarrow N* .$$

(20)
This construction will force $G'_2$ to handle properties as $G_2$ does, because the pattern of occurrences of each $U_i$ in (19) matches the property requirements of the $i$-th row of the table-transition display of (18), and because the metarules generate strings of indices of $G_2$, marked with "*"s, as substitutions for the $U_i$. Notice also that (19), constructed as just suggested, is a context-sensitive rule schema because the rule schema of (18) is, and because the transition-table display of (18) satisfies condition (ii) of Definition 19. One can see, then, that this construction will lead from $cs \ Vwp$ to $cs \ vwg$.

In formal terms, the portion of the construction just described requires the inclusion of $P, F, P_f$, and $\{a, b, *\}$ in $P'$; the inclusion of $M, M_f, \{U_p, V_1\}$,
and \( \{U_1, U_2, \ldots, U_{\hat{m}}\} \), where \( \hat{m} \) is sufficiently large, in \( M' \); the metarules

\[
\{U_i \rightarrow U_p \mid 1 \leq i \leq \hat{m}\} \\
\cup \{U_i \rightarrow \lambda \mid 1 \leq i \leq \hat{m}\} \\
\cup \{U_p \rightarrow V_1, U_p \rightarrow V_1 U_p\} \\
\cup \{V_1 \rightarrow W_\ast \mid (\langle x \rangle \rightarrow \langle W \rangle) \in R_1\},
\]

as well as \( Q \) and \( Q_1 \), in \( Q' \); and, corresponding to each rule schema \( r \in R \), one rule schema \( r' \) in \( R' \), defined as follows: Suppose

\[
r = (\langle X_0 \rangle \rightarrow \langle X_1 \rangle \cdots \langle X_k \rangle)
\]

and

\[
J(r) = \{(e_{10}, e_{11}, \ldots, e_{1k}), (e_{20}, e_{21}, \ldots, e_{2k}), \ldots, (e_{n0}, e_{n1}, \ldots, e_{nk})\}.
\]

Suppose also \( F = \{e_0, e_1, e_2, \ldots, e_n\} \). (For the construction, fix an order for \( F \) and for each \( J(r) \).) Then

\[
r' = \langle aX_0 e_1 U_{011} U_{012} \ldots U_{01m_0} e_2 U_{021} \ldots U_{02m_1} \ldots e_n U_{0n1} U_{0n2} \ldots U_{0nm} \rangle \\
\rightarrow \langle bX_1 e_1 U_{111} U_{112} \ldots U_{11m} e_2 U_{121} U_{122} \ldots U_{12m} \ldots e_n U_{1n1} U_{1n2} \ldots U_{1nm} \rangle \\
\rightarrow \langle bX_1 e_1 U_{111} U_{112} \ldots U_{11m} e_2 U_{121} U_{122} \ldots U_{12m} \ldots e_n U_{1n1} U_{1n2} \ldots U_{1nm} \rangle \\
\rightarrow \langle bX_1 e_1 U_{111} U_{112} \ldots U_{11m} e_2 U_{121} U_{122} \ldots U_{12m} \ldots e_n U_{1n1} U_{1n2} \ldots U_{1nm} \rangle,
\]

where

\[
\forall_{i=0}^k \forall_{j=1}^n \forall_{l=1}^m U_{ijl} = \begin{cases} U_i & \text{if } e_{il} = e_j \\ \lambda & \text{otherwise} \end{cases}.
\]

The remaining rule schemata of \( G_2' \) constructed according to (22) are

\[
\langle ap12 \rangle \rightarrow \langle bs12U_1 \rangle \\
\langle as12 \rangle \rightarrow \lambda \\
\langle as12U_1 U_2 U_3 \rangle \rightarrow \langle bs12U_2 U_3 \rangle bKU_1 U_3 2U_1 \\
\langle ad12U_1 \rangle \rightarrow \langle bi1U_1 \rangle \\
\langle au1U_1 \rangle \rightarrow \langle bo12 \rangle bi1U_1 \\
\langle acu12 \rangle \rightarrow \langle bo12 \rangle bc12.
\]

Two gaps remain in this part of the construction.
First, it may be that a series of applications of strict rules obtained from (22) is blocked by having the list of indices which possess a certain property out of order. This difficulty is illustrated in Fig. 4, where the step

\[ s \begin{pmatrix} x \\ xx \end{pmatrix} 2 \rightarrow s \begin{pmatrix} x \\ xx \end{pmatrix} 2 u \begin{pmatrix} x \\ xx \end{pmatrix} 1 \]

of \( G^*_2 \) must be mimicked by

\[ \langle asl_{12x*xx*} \rangle \rightarrow \langle bs1_{12x*xx*} \rangle \langle bu_{1xx*2} \rangle. \]

But the following step

\[ s \begin{pmatrix} x \\ xx \end{pmatrix} 2 \rightarrow s \begin{pmatrix} x \\ xx \end{pmatrix} 2 u \begin{pmatrix} x \\ xx \end{pmatrix} 0 \]

must be mimicked by

\[ \langle asl_{12xx*xx*} \rangle \rightarrow \langle bs1_{12xx*xx*} \rangle \langle bu_{1xx*2} \rangle, \]

that is, using lines 2 and 3 of the display (18) in reverse order with respect to "x" and "xx". The necessary switch is effected by application of the strict rule

\[ \langle asl_{12x*xx*} \rangle \rightarrow \langle asl_{12xx*xx*} \rangle. \]

We obtain all necessary switching rules by including in \( G^*_2 \) the (cs) rule schemata

\[ \langle aL_{11}U_{11}V_{11}V_{12}U_{12} \rangle \rightarrow \langle aL_{11}U_{12}V_{11}V_{12}U_{12} \rangle \]
\[ \langle aL_{11}U_{12}V_{11}V_{12}U_{12} \rangle \rightarrow \langle aL_{11}U_{12}V_{11}V_{11}U_{12} \rangle \]

and metarules (in addition to those given at (20))

\[ \begin{align*}
L \rightarrow p & \quad L \rightarrow i & \quad V_2 \rightarrow V_1 \\
L \rightarrow s & \quad L \rightarrow u \\
L \rightarrow U & \quad L \rightarrow o \\
L \rightarrow KU & \quad L \rightarrow cu \\
L \rightarrow d & \quad L \rightarrow c.
\end{align*} \]
In formal terms, we require (in addition to what has already been stated) the inclusion of \( \{L, V_2\} \) in \( M' \); the inclusion of metarules

\[
\{L \rightarrow X_i \mid \langle X_0 \rangle \rightarrow \langle X_1 \rangle \cdots \langle X_k \rangle \in R \text{ and } 0 \leq i \leq k\}
\]

\( \cup \{V_2 \rightarrow V_1\} \)  \( (26) \)

in \( Q' \); and, supposing \( F = \{e_0, e_1, e_2, \ldots, e_n\} \), the inclusion of the \( n \) cs rules

\[
\begin{align*}
\langle aL e_1 U_1 V_1 U_2 e_2 U_2 \cdots e_n U_n \rangle &\rightarrow \langle aL e_1 U_1 V_1 U_2 e_2 U_2 \cdots e_n U_n \rangle \\
\langle aL e_1 U_1 e_2 U_2 V_1 U_2 e_2 U_2 \cdots e_n U_n \rangle &\rightarrow \langle aL e_1 U_1 e_2 U_2 V_1 U_2 e_2 U_2 \cdots e_n U_n \rangle \\
&\cdots \\
\langle aL e_1 U_1 e_2 U_2 \cdots e_n U_n V_1 V_2 U_{n+1} \rangle &\rightarrow \langle aL e_1 U_1 e_2 U_2 \cdots e_n U_n V_1 V_2 U_{n+1} \rangle 
\end{align*}
\]

\( (27) \)

in \( R' \).

The second gap in the part of the construction forcing \( G' \) to handle indices and their properties as \( G \) does is that, although the inclusions of (21) and (22) permit \( G' \) to mimic \( G \), they also permit the same index to be listed several times in a "variable" of \( G' \), even under different properties. To exclude this equivocation in \( G' \) with respect to the properties possessed by indices, we include in \( Q' \) the set \( \hat{Q} \) of production rules of a context-sensitive grammar for the language consisting of strings in which all the nonneutral properties of \( G \) appear in a definite order, say \( (e_1, e_2, \ldots, e_n) \), each followed by a (possibly empty) string of indices of \( G \), each index followed by "\( \cdot \)", each element of the language having the property that no index appears twice in it. It is a straightforward matter to construct a linear-bounded automaton accepting the language just described. By familiar results, then, the context-sensitive grammar

\[
(\hat{M}, P_1 \cup \{*\} \cup F(e_0), \hat{Q}, V_H) \]

\( (28) \)

as required, exists.

We use (28) to complete this part of the construction by including \( \hat{M} \) in \( M' \), \( \hat{Q} \) in \( Q' \), and the one cs rule

\[
\langle bL V_H \rangle \rightarrow \langle aL V_H \rangle 
\]

\( (29) \)

in \( R' \).

To recapitulate: starting from a sentential form of \( G' \) in which all components begin with "\( a \)", one can mimic a step in a \( G \)-derivation by applying the corresponding rule obtained from (22). This application results in a sentential form of \( G' \) including some components which begin with "\( b \)".
These are eliminated, and the application checked for validity, by the application of rules obtained from (29). Finally, the resulting sentential form is put into shape for the next mimicking step by the application of rules obtained from (27).

We next consider the problem of disposing of the infinite terminal vocabulary of $G$.

The main idea here is to spell out the indices (which constitute the infinite part of the terminal vocabulary of $G$), including $P_l$ in $T'$ for the purpose. $T'$ will also include $T$ and one extra element, a string of "*'s long enough to make the rules used to spell out the indices context-sensitive. The terminal vocabulary for $G_2'$, for example, is \{\langle o\rangle, \langle x\rangle, \langle z\rangle, \langle *****\rangle\}. The spelling-out process is illustrated in Fig. 4.

To see what rule schemata are convenient for this part of the construction, consider that the application of the idea of (22) to the indexing rule schemata of $G_2$ would result in

$$\langle a_1 U_12 \rangle \rightarrow \langle b T_1 U_12 \rangle$$
$$\langle a c12 \rangle \rightarrow \langle b N_1 U_12 \rangle.$$  

However, considering the definition of extended terminals in a vWpg, it is clear that these rule schemata are too inclusive. The indexing steps of derivations in $G_2$ would be more accurately mimicked by

$$\langle a_1 T_1*2 \rangle \rightarrow \langle b T_1 T_1*2 \rangle$$
$$\langle a c12 \rangle \rightarrow \langle b N_1 N_1*2 \rangle.$$  

Finally, considering the redundancy in the above and the fact that the properties assigned to the index involved should not change at later stages of a derivation in $G_2'$ (there being no corresponding later stages in $G_2$), we determine to include

$$\langle a_1 T_1*2 \rangle \rightarrow \langle T_1***** \rangle$$
$$\langle a c12 \rangle \rightarrow \langle N_1***** \rangle.$$  

(30)

among the rule schemata of $G_2'$.

For the spelling-out itself, we include the rule schemata

$$\langle x U_1***** \rangle \rightarrow \langle x \rangle \langle U_1***** \rangle$$
$$\langle z U_1***** \rangle \rightarrow \langle z \rangle \langle U_1***** \rangle$$
$$\langle x***** \rangle \rightarrow \langle x \rangle \langle ***** \rangle$$
$$\langle z***** \rangle \rightarrow \langle z \rangle \langle ***** \rangle.$$  

(31)
along with the metarules

\begin{align*}
U_I & \rightarrow x & U_I & \rightarrow xU_I \\
U_I & \rightarrow z & U_I & \rightarrow zU_I.
\end{align*}

(32)

For uniformity in the terminal strings of \( G_2' \), we also include the rule schema

\[
\langle a_012 \rangle \rightarrow \langle a \rangle \langle **\rangle.
\]

(33)

In formal terms, we include \( \{U_I\} \) in \( M' \); the metarules

\[
\{U_I \rightarrow a \mid a \in P_I\}
\]

\cup \{U_I \rightarrow aU_I \mid a \in P_I\}

(34)

in \( Q' \); and the rule schemata

\[
\{\langle aU_I(\ast)\rangle \rightarrow \langle a \rangle \langle U_I(\ast) \rangle \mid a \in P_I\}
\]

\cup \{\langle a(\ast) \rangle \rightarrow \langle a \rangle \langle (\ast) \rangle \mid a \in P_I\}

\cup \{\langle axe_1e_2 \ldots e_n \rangle \rightarrow \langle x \rangle \langle (\ast) \rangle \mid x \in T'\}

(35)

in \( R' \), where, as before, \( F = \{e_0, e_1, e_2, \ldots, e_n\} \), and

\[
\hat{n} = n + 2 + \max\{|x| \mid \langle x \rangle \rightarrow \langle W \rangle \in R_j\}.
\]

In addition, we include, corresponding to each indexing rule schema, \( r = \langle \langle x \rangle \rightarrow \langle W \rangle \rangle \in R_f \), a set \( r' \) of rule schemata in \( R' \), defined as follows:

Suppose \( F, \hat{n} \) are as above, and \( J(r) = \{(e_{10}, e_{11}), (e_{20}, e_{21}), \ldots, (e_{m0}, e_{m1})\} \).

(For the construction, fix an order for \( F \) and for each \( J(r) \).) Then

\[
r' = \langle axe_1 U_{11} U_{12} \ldots U_{1m} W_{1j} e_2 U_{21} U_{22} \ldots U_{2m} W_{2j} \ldots e_n U_{nj} U_{n2} \ldots U_{nm} W_{nn} \rangle
\]

\[
\rightarrow \langle W(\ast) \rangle \mid 1 \leq i \leq m \text{ and } e_{i1} = e_1\),

(36)

where

\[
\forall_{i=1}^m \forall_{j=1}^n W_{ij} = \begin{cases} W \ast & \text{if } e_{i0} = e_j \\ \lambda & \text{otherwise,} \end{cases}
\]

\[
\forall_{j=1}^m \forall_{k=1}^m U_{jk} = \begin{cases} U_k & \text{if } e_{k0} = e_j \text{ and } e_{k1} = e_0 \\ \lambda & \text{otherwise.} \end{cases}
\]

As indicated in the discussion preceding (30), \( G' \) responds to the definition of extended terminals of \( G \), as applied to indices, by including a rule schema corresponding to \( r \in R_f \) and an element \((e_{i0}, e_{i1})\) of \( J(r) \) exactly when \( e_{i1} = e_1 \), and requiring that each other index (derived from a \( U_{jk} \)) which appears in
the left-hand side of such a rule schema be assigned a property $e_j$ for which $(e_j, e_0) \in J(r)$. That is, such a rule schema is included in $r'$ if and only if each "variable" of $G'$ derived from its left-hand side corresponds to an extended variable of $G$ which leads directly to an extended terminal of $G$ based on the right-hand side of $r$.

If $G$ is cs, then all the $U_{jk}$ are $\lambda$. Furthermore, it is always true for each $i$ that exactly one of the $W_{ij}$ is not $\lambda$. In the context-sensitive case, then, each rule schema of (36) is of the form

$$\langle a e_1 e_2 \cdots e_j W^* \cdots e_n \rangle \rightarrow \langle W(*)^\delta \rangle,$$

which, by definition of $n$, is cs. That is, this construction leads from cs $vWg$ to cs $vWg$.

The construction of $G'$ may now be specified formally, by the following definitions:

$$\hat{m} = \max(\{\text{card}(F)\} \cup \{\text{card}(J(r)) \mid r \in R \cup R_i\}).$$

$$\hat{n} = \text{card}(F) + 1 + \max\{\text{x} \mid \langle \langle \text{x} \rangle \rightarrow \langle W \rangle \rangle \in R_i\}).$$

We suppose the context-sensitive grammar (28) is as described above. We suppose $M, M_1, \hat{M}, P, P_1, F, \{a, b, *, L, U_I, U_P, U_1, U_2, ..., U_{\hat{m}}, V_1, V_2\}$ pairwise disjoint.

$$M' = M \cup M_1 \cup \hat{M} \cup \{L, U_I, U_P, U_1, U_2, ..., U_{\hat{m}}, V_1, V_2\}.$$

$$P' = P \cup P_1 \cup F \cup \{a, b, *\}.$$

$Q'$ consists of $Q, Q_I, Q$, and the metarules defined at (21), (26), and (34).

$R'$ consists of the rule schemata defined at (22), (27), (29), (35), and (36).

$$S' = \langle a e_1 e_2 \cdots e_n \rangle,$$

supposing $F = \{e_0, e_1, e_2, ..., e_n\}$, as before.

$$T' = T \cup \{\langle a \rangle \mid a \in P_1\} \cup \{\langle(*)^\delta \rangle\}.$$

Notice that $V_H$, used in $R'$, is specified as an element of $\hat{M}$ at (28). Recalling that $Q$ includes no $\lambda$-rules, notice that $\lambda \in L_W$ only for $W \in M \cup \{U_1, U_2, ..., U_{\hat{m}}\}$, so that $Q'$ satisfies Definition 1.

For example ($\hat{m} = 3, \hat{n} = 5$),

$$G_2' = \langle \hat{M}_2 \cup \{K, U, N, T, L, U_I, U_P, U_1, U_2, U_3, V_1, V_2\},$$

$$\{c, d, i, o, p, s, u, x, z, 0, 1, 2, a, b, *\},$$

$$Q_2 \cup Q_{I2} \cup \hat{Q}_2 \cup Q^\delta,$$

$$R'^n,$$

$$\langle a p l 2 \rangle,$$

$$\{\langle o \rangle, \langle x \rangle, \langle z \rangle, \langle * * * * \rangle\}.$$
where $Q''$ is given at (20), (25), and (32); $M_2, Q_2$ are suitably chosen as at (28); and $R''$ is given at (19), (23), (24), (29), (30), (31), and (33).

We turn finally to the construction of $D'$, which will do generally the work of $D$, but also will insert the markers "#(a)" required by $G'$ and will spell out all indices, in the sense of following each of their constituents by "#".

As a first approximation to the required construction, let $D'$ differ from $D$ only in that

(i) the output alphabet of $D'$ includes "#";
(ii) whenever $D$ outputs "#", $D'$ outputs "#(#)";
(iii) whenever $D$ outputs $a \in P_i$, $D'$ outputs "a#".

This is almost right. Unfortunately, however, it results in $D'$ outputting "a##(#)" rather than "a##(#)", when $D$ outputs "a#", if $a \in P_i$. We remedy this defect by supplying $D'$ with a pair of states $(t, 0), (t, 1)$ corresponding to each state $t$ of $D$, $(t, 1)$ to be entered by $D'$ whenever $D$ would have entered $t$ having just output $a \in P_i$. We also correct the single-step output of $D'$ in this connection.

Formally, suppose $D = (K, E, P \cup P_i \cup \{#, \}, g, s, A)$, and let $D$ be determined by $G$, as above. Define functions $\alpha_0, \alpha_1$ with

$$
\alpha_i : (P \cup P_i \cup \{#, \})^* \to (P \cup P_i \cup \{#, \})^* \quad (i = 0 \text{ or } 1)
$$

$$
: \lambda \mapsto \lambda,
$$

$$
ax \mapsto \begin{cases} a \alpha_0(x) & \text{if } a \in P \\ a \# \alpha_1(x) & \text{if } a \in P_i \end{cases}
$$

and $\alpha_0(#x) = #(a) \# \alpha_0(x)$, $\alpha_1(#x) = (a) \# \alpha_0(x)$. Define

$$
D' = (K \times \{0, 1\}, E, P \cup P_i \cup \{#, \}, g', (s, 0), A \times \{0\}),
$$

where

$$
g'(t, i, e) = \{((u, j), \alpha_i(x)) \mid (u, x) \in g(t, e) : j = \begin{cases} 1 & \text{if } (x = ya \text{ and } a \in P_i) \text{ or } (x = \lambda \text{ and } i = 1) \\ 0 & \text{otherwise} \end{cases} \}
$$

For example,

$$
D'_{1'} = (\{0, 1\} \times \{0, 1\}, \{\text{nl}, \text{op}, x, z\}, \{0, x, z, #, \}, g', (0, 0), \{(0, 0)\}).
$$
where \( g' \) is given by

\[
\begin{array}{c|c|c}
\text{nl} & (0, 0) & (0, 1) \\
\hline
\text{op} & \emptyset & \emptyset \\
x & \{(1, 0), o\#\\\\^{}
\emptyset & \{(1, 0), o\#\\\\^{}
\hline
z & \{(1, 1), x\#} & \{(1, 1), x\#}
\end{array}
\]

If we assume, as before, that each line of (15) is followed by "nl", call the resulting string \( X \), and denote by \( Y \) the string whose derivation is given in Fig. 4, then it is true that

\[
D_{l'}(X) = \{(1, 0), \#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\##
GRAMMARS WITH STRUCTURED VOCABULARY


