Notre Dame Journal of Formal Logic
Volume 30, Number 2, Spring 1989

Combinators and Categorial Grammar

PETER SIMONS*

Abstract From Ajdukiewicz onwards categorial grammar has failed to characterize adequately the twin roles of variable-binding operators and the variables they bind. Categorial grammars are, however, adequate for languages using combinators and dispensing with bound variables, but the combinators used hitherto have been insufficiently flexible to eliminate bound variables from languages which allow both higher-order quantification and multi-place functors, such as those of Leśniewski. This paper introduces seven multigrade combinators and shows how they can be used to eliminate bound variables from Leśniewskian languages, yielding logical languages as powerful as these but in perfect harmony with Ajdukiewicz's type of grammar. An alternative to multigrade combinators will also be considered and the significance of the result for the understanding of languages with more flexible category assignments is studied.

1 Introduction: Categorial grammar and bound variables The history of categorial grammar is usually said to have started with the 1935 paper by Ajdukiewicz [1] on syntactic connection. The practices Ajdukiewicz attempts to codify in the paper were applied explicitly by Leśniewski, although the basic principles of categorial grammar had already been enunciated without being applied by Husserl and had been applied without being enunciated by Frege. Ajdukiewicz's achievement consisted principally in the introduction of a notation for syntactic categories and an algorithm for testing the syntactic connection of strings of expressions. Since Ajdukiewicz's inception of categorial grammars they have been considerably refined in numerous ways, in particular in the direction of making them more adequate for natural languages (cf. [2] and the literature cited therein). Ajdukiewicz's account is, however, unsatisfactory in one crucial respect, which shows up in the treatment not of natural languages but of the languages of mathematics and logic which employ bound variables and operators binding

*My thanks go to Karel Lambert and to the referee for their advice.

Received February 24, 1987; revised June 23, 1987

them. Ajdukiewicz does indeed offer an analysis, but it fails to capture what he recognized as the essence of such operators, namely the way in which they bind variables. Consider for example the universal quantifier: on Ajdukiewicz's analysis there is no difference between a sentence $\forall x(Fx)$, where the quantifier binds a variable in its scope, and a sentence $\forall x(Fa)$ where it does not. This difference, for instance, is crucial for a description of Leśniewski's logical systems, where vacuous quantification is not even syntactically well-formed. Ajdukiewicz is forced to regard the string consisting of the universal quantifier followed by one or more variables as a simple, unanalyzed symbol, because he sees no way (consistent with an extensional account of logic) of taking $\forall$ as a functor taking the variables as arguments. Hence his account fails to preserve the compositionality which is such an attractive feature of categorial grammar. It is thus ironic that Ajdukiewicz's grammar, which is inspired by Leśniewski's work in logic, is adequate only for the tiny quantifier-free fragments of languages, such as those of Leśniewski, employing bound variables. This inadequacy does not show up directly in the application of categorial grammar to natural languages, which at least superficially lack the device of bound variables.

Despite the developments categorial grammar has undergone since Ajdukiewicz's paper, I consider this problem still unsolved. No categorial grammar I have come across satisfactorily captures the roles of bound variables and the operators binding them. In a way this is not surprising, since variable binding is an essentially nonfunctorial operation, whereas categorial grammars codify only functorial combination. Bound variables mark places, of particular syntactic categories, for operators to reach into, and the use of equiform variables allows an operator to reach into more than one place at once. Thus, in a real sense bound variables are only, as the older terminology had it, "apparent". As physical constituents of physical expressions, bound variables are an expedient, and do not so to speak correspond to any constituent of the proposition the sentence expresses, but represent rather a *pattern* of binding. A pattern is, however, precisely something which is not a constituent expression, since a pattern cannot, while a constituent can, be exhibited in isolation from its context. It is nevertheless attractive to view a variable binder together with its variable pattern as a functor, and its matrix as an argument: in the semi-English sentence '$\forall x(x$ flows)' the quantifier-variable pattern '$\forall x(x-)$' takes the one-place predicate '. . . flows' to yield a sentence, in just the way that its English counterpart 'everything' does. So if we can find a way to eliminate binder-variable patterns in favor of functors like this, we can apply categorial grammar directly to the result.

Now it is in principle possible to avoid bound variables in many formal systems if one employs combinators, as was shown, independently, by Schönfinkel and Curry ([3],[4],[11]). Combinators were indeed introduced in order to provide a clearer understanding of variable binding. Suppose then that for every expression containing bound variables there is an equivalent one containing combinators but no bound variables, to which the principles of categorial grammar apply directly. This can be viewed in one of two ways. If we consider the combinatorial language to reveal the logical form of the variable-binding language, it shows how categorial grammar may be applied to variable-binding languages, albeit not to their surface form. If on the other hand we consider the combinatorial language as merely providing a language of equivalent expressive power to the variable-binding language, while categorial grammar fails to provide a face-value syntactic analysis of this, we have still shown how to find a categorial language equivalent to it in expressive capacity. Of these two viewpoints, I incline towards the first, and would regard the combinatorial analysis as revealing the way in which categorial grammar applies to variable-binding languages.

However, the Schönfinkel–Curry combinators are not adequate to the task of eliminating bound variables from all languages, since they apply only to functors which take one argument, whereas in most languages there are multi-place functors. This is not an insurmountable drawback to the use of these combinators in formulating logical languages, since a multi-*place* functor can be replaced by what Leśniewski calls a multi-*link* functor (see [7]). Consider for example the two-place addition functor +, which for any two numerical terms, for example 2 and 3, forms a complex numerical term, here 2 + 3, by being simultaneously "saturated" by the two terms, and which accordingly has syntactical category n/nn in Ajdukiewicz's familiar fraction notation. One may instead take the functor \$ which is of category $(n/n)/n$, and obtain a numerical term not by two simultaneous "saturations" but by two successive ones: first form the complex functor \$(3), of category n/n, and then saturate it with the term 2 to get \$(3)(2), which is finally of category n. By use of this device the syntax of combinators may be kept simple. However there are languages, such as those of Leśniewski, which explicitly distinguish syntactically between multi-link functors and their associated single-link, multi-place ones: the functors + and \$ belong to different categories and cannot be substituted *salva congruitate* for one another. So the Schönfinkel–Curry combinators cannot be used as they stand to eliminate variables from such a language.

On the other hand, Quine has used combinators to eliminate variables from languages with multi-place functors (see [9],[10]). Quine's combinators are however specially adapted to languages of first-order, whereas other languages—again, such as those of Leśniewski—may contain bound variables of any syntactic category. Quine's combinators are too weak to eliminate bound variables of categories other than that of names.

So, if bound variables are to be eliminated from languages as rich in their syntactic categories as those of Leśniewski, or any other language in which both higher-order variables are quantified and multi-place functors occur, in such a way that the principles of categorial grammar apply directly to the result, features of both approaches must be combined. This paper indicates how this can be done, and so in my opinion solves the problem of the adequacy of categorial grammars for a wide range of variable-binding languages. Rather than present and describe a particular language, I give a general recipe for applying the result to languages constructed on the pattern of those of Leśniewski. The result is of interest not just to Leśniewski scholars. Leśniewski's logical systems are comparable with simple type theories, so the results are relevant to these and to higher-order logics in general. Leśniewski's languages are taken as exemplary because (modulo bound variables) they are explicitly constructed along categorial lines, so their recursively constructed grammar is relatively simple, and yet also very rich in its categories, so that in describing how to provide a grammar for Leśniewskian languages it becomes clear how to do so for languages with more restricted grammars.

It is not initially necessary to consider categorial grammars in complete generality, but only to the extent required for the description of the grammar of Leśniewskian languages. For this one need only consider *simple* categorial grammars, which are such that each token expression in a category belongs just to this category and to no other. Many categorial grammars allow a single token expression to belong simultaneously to many categories, which are derived according to what are called type-change rules.[1] Such *extended* categorial grammars are more adequate for describing the grammar of natural languages than are simple categorial grammars. However, Leśniewskian languages do not require this refinement for their description; herein consists both their simplicity and their rigidity in comparison with natural languages. I will come back later to the relevance of the main result of this paper to languages with type-change rules.

2 Leśniewskian languages

2.1 Categories The syntactic or grammatical categories are generated recursively: they divide exhaustively into *basic* categories and *functor* categories. Functor categories are characterized uniquely by the categories of their syntactic *inputs* (in order) and the category of their *output*. Simple categorial languages recognize a single structural principle: the simultaneous saturation of a functor expression by one or more argument expressions, taken in a certain order, forms a unified complex expression. How saturation and the order of arguments is marked *in concreto* varies, as comparisons among natural languages show. However, the constraints of linearization suggest one natural way of realizing such structure, which is to split the principle into two structural principles: a linear, "horizontal" principle in which expressions are formed into *sequences*, and a hierarchical, "vertical" principle in which functors are applied to such sequences. The sequential principle can be realized in a written language by forming strings of expressions, letting the usual conventions of writing the successive words of a text represent the order of sequence. The hierarchical principle may be realized by enclosing such strings and attaching a functor to the resulting package. For linear terms this is easily attained by delimiting the string with parentheses and pre- or (less frequently) post-posing the functor expression. It is this way which Leśniewski adopts, and his procedure is followed here both in the grammar of the languages and (here deviating somewhat from tradition) in the way expressions signifying the categories are constructed. We shall not therefore consider directional categorial grammars, in which a distinction is made between functorial application to the left and to the right. Directionality may be required for an adequate description of many formal and most or all natural languages, but it introduces complications that would obscure our main point. Leśniewski's languages were explicitly constructed in a way which made it unnecessary to consider directionality.

We are now able to characterize the categories available in a Leśniewskian language by a simultaneous recursion defining both *category* and *category sequence.* We presuppose as a primitive the concept *basic category.* Basic categories are given by a finite list:

C1 Each of the finite number of basic categories is a category.
C2 For all α, if α is a category then α is a category sequence.
C3 For all α and ρ, if α is a category and ρ is a category sequence then the sequence consisting of ρ followed by α is a category sequence distinct from the other two.
C4 For all α and ρ, if α is a category and ρ is a category sequence, then the functor category taking as inputs the members of the sequence ρ in order and yielding as output the category α is a category distinct from all the input categories and from the output category. Functor categories formed in this way are distinct iff their input category sequences are distinct, or their output categories are distinct, or both.
C5 Nothing is a category or category sequence except in virtue of C1–C4.

I adopt the following notation for categories:

N1 Basic categories are symbolized by simple symbols. I shall mention only 's' for the category of sentences and 'n' for the category of names.
N2 Each category symbol is also used to symbolize the same category considered as a category sequence.
N3 If 'ρ' symbolizes a category sequence and 'α' a category, then the concatenation '$\rho\alpha$' symbolizes the category sequence consisting of the first category sequence ρ followed by the category α.
N4 If α is a category and ρ a category sequence, then '$\alpha(\rho)$' symbolizes the functor category whose output category is α and whose input category sequence is ρ.
N5 All category symbols are formed in accordance with the directives N1–N4.

The notation may be illustrated with examples. The categories of + and \$ from the previous example are now represented by $n(nn)$ and $n(n)(n)$, respectively. The category of the functor of sentential negation is $s(s)$, that of sentential conjunction is $s(ss)$, while that of a functor forming an intransitive verb from a transitive verb and a name (in that order) is $s(n)(s(nn)n)$.

In the languages formulated by Leśniewski there are only two different hierarchies of categories. The language of protothetic, in effect a system of propositional types, has as its sole basic category that of sentence, while the language of ontology, Leśniewski's general logic, which is akin to a simple type theory in expressive power, adds the second basic category of name. Here, however, the only restriction we place on the number of basic categories is that it be finite.

2.2 Parentheses and other brackets In his directives for constructing logical systems Leśniewski is very liberal in what he allows as expressions: there are no set prescriptions concerning the shape or typeface of symbols (whether constant or variable) of a given category. We shall follow this liberality in not specifying exactly which symbols are to be admitted into the languages under consideration, provided only that certain restrictions are observed. In this way our account applies to any language constructed along these lines.

Leśniewski even allows distinct but equiform expression-tokens to belong to different categories and leaves it to the context to disambiguate. This is made possible by requiring that each functor category have associated with it a characteristic shape of parentheses for enclosing its arguments. While the device of using different shapes of parentheses has its attractions, because it allows equiform symbols to be used for analogous constants of a different category, and because it sometimes aids clarity, it is theoretically dispensable and can be typographically extravagant.[2] Things are kept simple here by tightening up the requirements on symbols: once a symbol is assigned to a category all symbols equiform with it are stipulated to belong to that category and no other. In this way only one shape of parentheses is needed, the usual curved ones: (,). This minimal deviation from Leśniewski's liberalism is worth the gain in typographical and grammatical simplicity. We follow Leśniewski, however, in distinguishing the parentheses which enclose the arguments of a functor from the symbols which enclose the variables bound by a universal quantifier, which are lower left and right corners: $\llcorner$, $\lrcorner$, and from the symbols which mark the scope of the quantifier, which are upper left and right corners: $\ulcorner$, $\urcorner$. It is clear that these two punctuating tasks are so different from that performed by parentheses that it is appropriate to mark the difference by different symbols.[3]

2.3 Metalinguistic conventions Although in principle I agree with Leśniewski that the primary and ontologically most acceptable way of considering expressions is as concrete tokens between which there are relations of equiformity, the convention adopted above allows us to ignore the difference between equiform tokens and to employ the more usual metalinguistic devices, treating metalinguistic expressions as denoting expression types. The metalinguistic devices employed in this paper are as follows, where 'wfe' abbreviates 'well-formed expression' and 'wfeseq' abbreviates 'finite sequence of well-formed expressions':

Letter Terms The letters a, b, c, d, e, f, g, with or without subscripts are used as schematic letters for wfe types.

Autonymous Terms Tokens of left and right parentheses, left and right upper and lower brackets, and any other stipulated expressions of the object language, may be employed in a metalinguistic manner as names for their own types in the object language.

Sequence Letter Terms The letters r, s, t with or without subscripts stand for finite sequences of expressions. Because wfes are sequences of length 1 they also count as sequence terms.

Descriptive Sequence Terms Two basic expressions are employed as schematic terms signifying kinds of sequence of wfes: '$r\ldots$' signifies wfeseqs whose first members are the sequence r (there need be no others, and r may be a single wfe); '$\ldots r$' signifies wfeseqs whose last members are the sequence r (with similar qualifications).

Concatenation If x and y name or signify expressions, the concatenation xy signifies their sequence. So we can name sequences like ab, rs, ras, and wfes like $f(r)$. In combination with descriptive sequence terms, concatenation is a powerful device enabling us to signify various kinds of wfe and sequence, e.g., $\ldots a \ldots$, $a \ldots b$, $f(a \ldots)$, $\llcorner b \lrcorner \ulcorner f(a \ldots b \ldots c) \urcorner$.

2.4 Formation rules for Leśniewskian languages Here we present recursive principles for constructing well-formed expressions (wfes) of a Leśniewskian language. Formation rules normally concern only well-formed formulas (wffs) and sometimes also well-formed singular terms, but more extensive stipulations are required here because of the syntactic complexity of Leśniewskian languages.

2.4.1 Preliminary requirements Every expression is either *simple* or *complex* and no expression is both. Simple expressions may be called (following Leśniewski) *words*. Words are either *terms* or *punctuators* and no word is both. Punctuators are left and right parentheses and upper and lower corners and nothing else, and do not belong in any category. Each term belongs to a single category. Every term is either a *constant* or a *variable* of its category and no term is both. The distinction between constants and variables is here a purely syntactic one. Complex expressions consist of certain finite sequences of words. Every expression, whether simple or complex, must have exactly one decomposition into words. So, for instance, the expression ')(' may not be a word, since it is equiform to a complex expression consisting of two punctuators.[4]

2.4.2 Proper variable sequences A *proper variable sequence* (pvs) is a nonempty sequence of variables, none of which are alike:

PVS1 Every variable is a pvs.
PVS2 If r is a pvs and a is a variable and a does not occur in r then ra is a pvs.
PVS3 Nothing else is a pvs.

One pvs is a *shuffle* of another iff they contain the same variables but perhaps differ in the order in which they occur.

2.4.3 Definitions for the specification of well-formedness A *well-formed formula* (wff) is a wfe of category s.

If r is a pvs and a is a wff, then $\llcorner r \lrcorner \ulcorner a \urcorner$ is a *quantifier expression*, the initial $\llcorner r \lrcorner$ of which is called its *quantifier* and the a of which is called its *scope*.

If the variable a occurs in the wfe b in the scope of a quantifier in b whose pvs contains a, then this occurrence of a is *bound* in b. If a occurs in b but not in the pvs of a quantifier in b and is not bound in b this occurrence of a is *free* in b, and a is said to *occur free* in b.

A wfe in which some variable occurs free is called *open*, one with no free occurrences of any variable is *closed*. A closed wff is called a *sentence*.

2.4.4 Formation rules The following specification recursively defines 'well-formed expression' (wfe) and 'finite sequence of well-formed expressions' (wfeseq).

WF1 Every term of category α is a wfe of category α.
WF2 Every wfe of category α is a wfeseq of category sequence α.
WF3 If r is a wfeseq of category sequence ρ and a is a wfe of category α then ra is a wfeseq of category sequence $\rho\alpha$.
WF4 If f is a wfe of category $\phi(\rho)$ and r is a wfeseq of category sequence ρ then $f(r)$ is a wfe of category ϕ.

WF5 If r is a pvs and a is a wff such that each variable in r occurs free in a, then $\llcorner r \lrcorner \ulcorner a \urcorner$ is a wff.

WF6 Nothing is a wfe or wfeseq except by virtue of WF1–WF5.

Notice that this recursive specification runs parallel to the recursive specification of the categories except for the quantifier clause WF5. It is precisely here that the syntax of languages with bound variables breaks the bounds of simple categorial grammar.

3 Multigrade combinators Schönfinkel–Curry combinators work only for one-place functors, but they are categorially flexible, whereas Quine's combinators can cope with multi-place functors, but eliminate only individual variables. The combinators introduced here combine both aspects. Because they are not assigned to a fixed syntactic category I call them *multigrade* combinators. In Section 5 this conception will be examined in more detail.

3.1 Syntax To a Leśniewskian language we add the following seven symbols:

$$I \quad W \quad R \quad K \quad J \quad B \quad U$$

and require that no word or complex expression of the unextended language be equiform to any of them. The following clauses must now be added to the formation rules:

WFI If a is a wfe of category α, so is $I(a)$.

WFW If f is a wfe of category $\phi(\alpha_1 \ldots \alpha_n \alpha_n)$, where $n \geq 1$, then $W(f)$ is a wfe of category $\phi(\alpha_1 \ldots \alpha_n)$.

WFR If f is a wfe of category $\phi(\alpha_1 \alpha_2 \ldots \alpha_n)$, where $n \geq 2$, then $R(f)$ is a wfe of category $\phi(\alpha_2 \ldots \alpha_n \alpha_1)$.

WFK If f is a wfe of category $\phi(\beta_1 \ldots \beta_n)(\alpha_1 \ldots \alpha_m)$, where $m \geq 1$ and $n \geq 1$, then $K(f)$ is a wfe of category $\phi(\beta_2 \ldots \beta_n)(\alpha_1 \ldots \alpha_m \beta_1)$.

WFJ If f is a wfe of category $\phi(\beta_1 \ldots \beta_n)(\alpha_1 \ldots \alpha_m)$, where $m \geq 1$ and $n \geq 1$, then $J(f)$ is a wfe of category $\phi(\alpha_m \beta_1 \ldots \beta_n)(\alpha_1 \ldots \alpha_{m-1})$.

WFB If g is a wfe of category $\gamma(\phi_1 \ldots \phi_m)$, where $m \geq 1$, then $B(g)$ is a wfe of category

$$\gamma(\alpha_{11} \ldots \alpha_{1n_1} \ldots \alpha_{m1} \ldots \alpha_{mn_m})(\phi_1(\alpha_{11} \ldots \alpha_{1n_1}) \ldots \phi_m(\alpha_{m1} \ldots \alpha_{mn_m}))$$

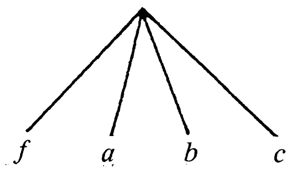
where for each i, $1 \leq i \leq m$, $n_i \geq 1$.

WFU If f is a wfe of category $s(\alpha_1 \ldots \alpha_n)$, where $n \geq 1$, then $U(f)$ is a wff.

The exclusion clause WF6 needs to be modified to accommodate these new clauses.

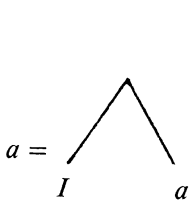
The extended language contains both the combinators and the bound variables and quantifiers. So there are wfes which are "mixed", containing some combinators and some bound variables. The "pure" fragments (combinator-free and bound-variable-free) are what is principally of interest; the extended language is there to allow us to proceed step by step from one pure fragment to the other.

3.2 Transformation rules Each combinator is now introduced by name and its effect shown by an equation. The equation means that two expressions of the kind signified by the two sides are synonymous and can therefore be substituted for one another in all contexts. The equations may be treated as schematic bidirectional transformation rules, schematic because each illustrative example covers an infinite range of structurally similar cases. Because of the need for parentheses, linear notation is difficult to survey and in practice I always work with structural diagrams. The structure of a complex functor/argument expression is shown by a tree diagram in which the leftmost node at any level represents the functor and those to the right of it represent the arguments (in order) of this functor. So the diagram



represents the syntactic structure of the wfe $f(abc)$. Because of the recursivity of the grammar, both the functor f and the arguments a, b, c may themselves be complex to any finite degree. Schematic diagrams, including dots to mark omitted arguments, correspond to the use of descriptive sequence terms, and schematic diagrams corresponding to linear notation are given for each combinator. The dots in descriptive sequence terms and in schematic diagrams are theoretically dispensable, but they are so widely used that it is helpful to employ them to present the rules in a compact and easily understood way. The variables in the illustrations are chosen to harmonize with the schematic indications of category given in the formation rules in Section 3.1. So f is of category $\phi(\dots)$, a_n of category α_n, and so on. In this way the effective category of a combinator on an occasion of its use may be computed using the inputs and outputs as given in the syntactic rules.

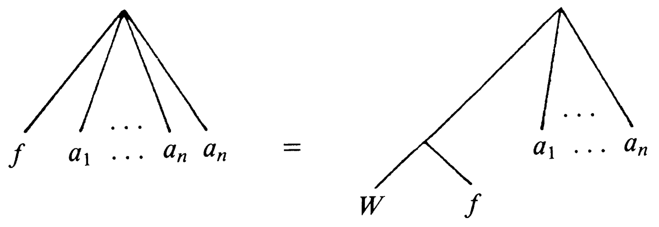
Identity: I
Effect: $a = I(a)$
Diagram:



Comment: Applied to any wfe, I yields a wfe of the same category, but with an extra layer of functorial structure.

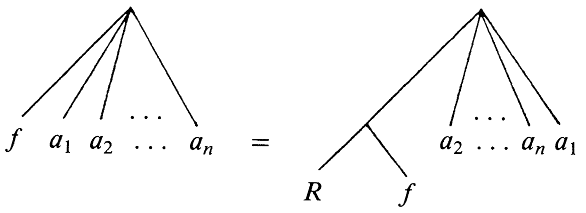
Reflection: W
Effect: $f(a_1 \dots a_n a_n) = W(f)(a_1 \dots a_n)$

Diagram:



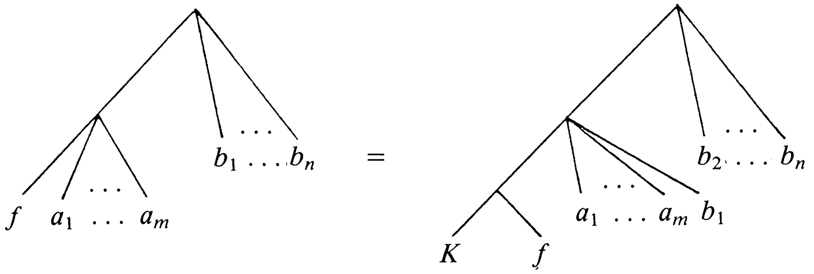
Comment: Applied to a functor whose last two arguments are alike, W eliminates this repetition.

Rotation: R
Effect: $f(a_1 a_2 \dots a_n) = R(f)(a_2 \dots a_n a_1)$
Diagram:



Comment: The first shall be last. Applied to functors of at least two arguments, R takes the front one to the end and moves the rest forward. R is nilpotent of degree equal to the number of arguments the functor takes.

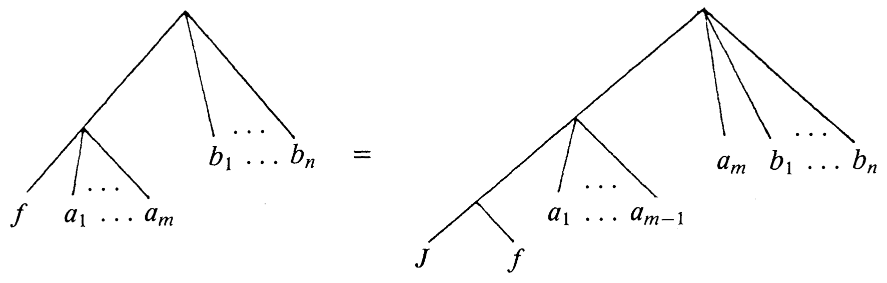
Lowering: K
Effect: $f(a_1 \dots a_m)(b_1 \dots b_n) = K(f)(a_1 \dots a_m b_1)(b_2 \dots b_n)$
Diagram:



Comment: Applies to multi-link functors and shifts the first of the uppermost group of arguments ("upper" in view of the usual tree representation of the syntax) down to be the last of the arguments in the next lower batch. If $n = 1$ the upper arguments all disappear and the syntactic depth of the sequence of batches of arguments is reduced by one.

Raising: J
Effect: $f(a_1 \dots a_m)(b_1 \dots b_n) = J(f)(a_1 \dots a_{m-1})(a_m b_1 \dots b_n)$

Diagram:

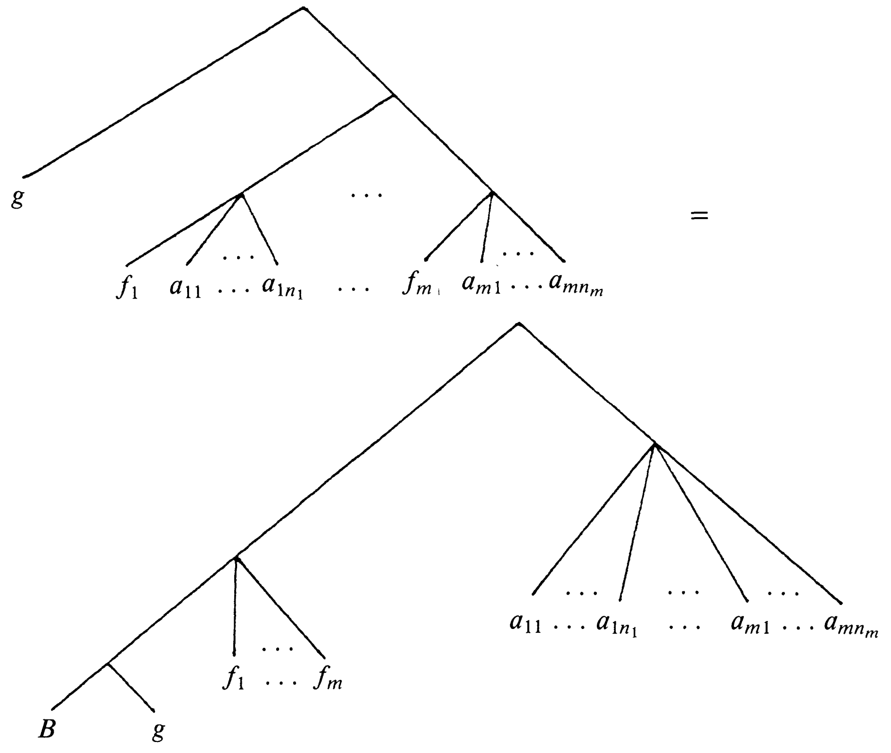


Comment: The converse of lowering. If $m = 1$ the lower level of arguments disappears and the depth of the argument batches is again reduced by one. Both lowering and raising can thus be used repeatedly to "level up" deeply nested argument structures.

Composition: B

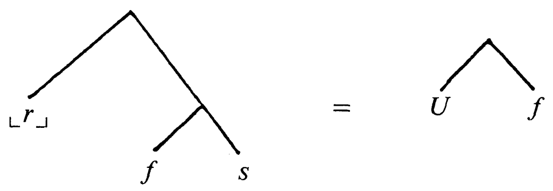
Effect: $g(f_1(a_{11} \dots a_{1n_1}) \dots f_m(a_{m1} \dots a_{mn_m}))$
$= B(g)(f_1 \dots f_m)(a_{11} \dots a_{1n_1} \dots a_{m1} \dots a_{mn_m})$

Diagram:



Comment: Takes a functor whose arguments are themselves functor-argument combinations and separates these inner functors from their arguments, forming a multi-link functor whose lower argument batch is the original functors in turn and whose upper argument batch is their arguments taken in turn. It is thus comparable to the operation of composing functions, whence its name. Structurally, composition is the most important combinator, and may be likened to the conductor of an orchestra: the other combinators are as it were cued in by *B*.

Universalization: *U*
Effect: $\llcorner r \lrcorner \ulcorner f(s) \urcorner = U(f)$
where f has category $s(\alpha_1 \ldots \alpha_n)$ for $n \geq 1$, where s is a pvs whose category sequence is $\alpha_1 \ldots \alpha_n$, and r is a pvs which is a shuffle of s, and finally, none of the variables in r (or s) occur free in f.
Diagram:



Comment: If *B* is the conductor, *U* is the star soloist. It is the combinator for whose sake all the others exist, because it alone eliminates bound variables (*W* only eliminates repetitions). *U* is the universal counterpart to Quine's (existential) *derelativization* combinator, but it is categorially far more flexible, as it needs to be.[5]

4 Eliminating bound variables

Theorem *Every combinator-free wfe of an extended Leśniewskian language is equivalent to a wfe which contains neither quantifiers nor variables which are bound in that wfe.*

Remark Because the quantifier is the only variable binder and because vacuous quantification is not permitted, the conditions of a wfe's containing no quantifiers and its containing no variables bound in it are equivalent.

Lemma *Let b be a quantifier-free wfe in which the variable a occurs. Then b is equivalent to a wfe of the form $f(a)$, where a does not occur in f.*

Sublemma *Let b be as in the Lemma. Then b is equivalent to a wfe of the form $g(a)$, where a occurs in g one less time than in b.*

Proof: The proof of the Sublemma is by induction on the depth of the shallowest occurrence of a in b.

Definition The *depth* of occurrence of one wfe in another is defined as follows:

D1 Each wfe occurs within itself with depth 0.
D2 If $f(r)$ occurs in a wfe with depth d then the occurrences of f and the arguments in r occur in this wfe with depth $d + 1$.
D3 If $\llcorner r \lrcorner \ulcorner a \urcorner$ occurs in a wfe with depth d then both a and the variables in the pvs occur in this wfe with depth $d + 1$.

This covers all cases. Intuitively, depth can be pictured as the number of steps one must take down a constituency tree from the base to the occurrence of the expression in question.

The first base case is where this depth is 0. Then b is a, so replace b by $I(a)$. We need to consider another base case: depth 1. Then a is an immediate constituent of b. Since b is quantifier-free, a must be either a functor of b or an argument of b. If a is an argument of b then b has the form $f(r_1 a r_2)$, where r_1 and r_2 are sequences, either or both of which may be null. We may notate the steps required as a series of equations:

$$\begin{aligned} f(r_1 a r_2) &= R(\ldots R(f)\ldots)(r_2 r_1 a) \\ &= I(R(\ldots R(f)\ldots))(r_2 r_1 a) \\ &= B(I)(R(\ldots R(f)\ldots))(r_2 r_1 a) \\ &= K(\ldots K(B(I))\ldots)(R(\ldots R(f)\ldots) r_2 r_1)(a). \end{aligned}$$

This is of the form required. Note that either or both of R, K might not be needed at all. In the following such null cases will not be specially mentioned. The second subcase is the case where a is a functor of b. Then b has the form $a(r)$. We proceed as follows:

$$\begin{aligned} a(r) &= I(a(r)) \\ &= B(I)(a)(r) \\ &= J(B(I))(ar). \end{aligned}$$

This now has the form of the previous subcase, with a as argument, and the procedure from that may be applied to bring a out to the top right on its own.

Having dealt with the base cases, assume now that we can raise variables to the top right from depths up to and including n, where $n \geq 1$, and suppose our shallowest a is at depth $n + 1$. So a is at depth 2 or more. Suppose a is an immediate constituent of a wfe c, and this in turn is an immediate constituent of a wfe d. We have as part of the structure tree then

$$\begin{tikzcd} d \arrow[d, dash] & \text{depth } n-1 \\ c \arrow[d, dash] & \text{depth } n \\ a & \text{depth } n+1 \end{tikzcd}$$

and there are precisely four cases to consider:

Case 1: c is a functor of d, a is a functor of c
Case 2: c is a functor of d, a is an argument of c
Case 3: c is an argument of d, a is a functor of c
Case 4: c is an argument of d, a is an argument of c.

Since b is quantifier-free, these are all the cases that can occur: we take them in turn.

Case 1. $d = a(r)(s)$ (for certain sequences r, s)

$$\begin{aligned} d &= a(r)(s) \\ &= I(a(r))(s) \\ &= B(I)(a)(r)(s) \\ &= J(B(I))(ar)(s) \\ &= R(J(B(I)))(ra)(s) \\ &= J(R(J(B(I))))(r)(as). \end{aligned}$$

Now a is at level n and the inductive hypothesis can be applied.

Case 2. For a certain functor f and sequences r_1, r_2, and s, where r_1 and r_2 are possibly null, we have

$$\begin{aligned} d &= f(r_1 a r_2)(s) \\ &= R(\ldots R(f)\ldots)(r_2 r_1 a)(s) \\ &= J(R(\ldots R(f)\ldots))(r_2 r_1)(as) \end{aligned}$$

and a is at level n again.

Case 3. For a certain functor f, and sequences r_1, r_2, and s, where as before either or both of r_1 and r_2 might be null

$$d = f(r_1 a(s) r_2).$$

If any of the members of r_1, r_2 are terms like e, replace them by the equivalent complex expressions $I(e)$. Then d is equivalent to an expression of the form

$$f(g_1(s_1)\ldots a(s)\ldots g_n(s_n)) = B(f)(g_1\ldots a\ldots g_n)(s_1\ldots s\ldots s_n).$$

a is still at level $n + 1$, but it is now an argument, and the procedure of Case 2 applies.

Case 4. $d = g(r_1 f(s_1 a s_2) r_2)$.
Again, if there are any plain terms in r_1 or r_2 replace them by their I counterparts, so we obtain a formula equivalent to d of the form

$$g(f_1(t_1)\ldots f(s_1 a s_2)\ldots f_n(t_n)) = B(g)(f_1\ldots f\ldots f_n)(t_1\ldots s_1 a s_2\ldots t_n)$$

and a is at depth n, where the inductive hypothesis can apply. This completes the inductive step.

So we can always bring a shallowest occurrence of a out to the top right. This completes the proof of the Sublemma.

We now use this to prove the Lemma. Since for any quantifier-free wfe b containing a we have an equivalent wfe $c(a)$, where c contains one less occurrence of a than b, we simply apply the sublemma as often as required to successively bring out occurrences of a until there are none left. We then have a wfe equivalent to b of the form

$$f(a)(a)\ldots(a)$$

where f does not contain a. We now apply J or K as often as required to level up all the a's into one batch, and then apply W as often as required to eliminate repetitions, and we end up with a wfe of the form $g(a)$ as required, where a does not occur in g, since g has the form $W(\ldots W(J\ldots(J(f))\ldots)\ldots)$, or $W(\ldots W(K\ldots K(f))\ldots)\ldots)$, and a does not occur in f. This proves the Lemma.

Now we turn to the Theorem. The proof proceeds by induction on the depth of nesting of quantifiers. If there are no quantifiers in b we need to do nothing. If there are quantifiers in b, then at least one quantifier expression c in b is of the form $\llcorner r \lrcorner \ulcorner a \urcorner$, where a is quantifier-free. Consider all the variables in the pvs r. Each one occurs free in a. Applying the Lemma, we bring these variables out, one by one, to the right, so that only one occurrence of each remains. We then apply J or K as often as required to level all these variables into one batch, so we have $a = g(s)$ for some g and s, where s is a pvs which is a shuffle of r and g contains none of the variables in r. We then apply U to get $U(g) = \llcorner r \lrcorner \ulcorner g(s) \urcorner = c$. This quantifier and these bound variables may thus be made to disappear. Note that because of the liberality of Leśniewski's quantifier rules we do not need to arrange the variables in s in the same order as in r: any shuffle suffices. It is obvious that because The wfe b is only finitely complex, finitely many applications of this technique will remove all quantifiers and bound variables, as the Theorem states.

The elimination of bound variables may be illustrated by an example. The so-called short axiom of Leśniewski's ontology may be written as follows:

$$[ab]: a \in b \;.\!\equiv\!.\; [\exists c].\; a \in c \;\&\; c \in b.$$

We now rewrite this according to the syntactic rules of the grammar given in this paper, with functors always preceding their arguments in parentheses, and eliminating the particular quantifier. We then get

$$\llcorner ab \lrcorner \ulcorner \equiv (\in(ab) \sim (\llcorner c \lrcorner \ulcorner \sim (\&(\in(ac)\in(cb)))\urcorner))\urcorner.$$

Applying the combinator rules this may be converted step by step into a formula without quantifiers or bound variables. One such possible conversion (there are several possible) yields the following result:

$$U(W(R(R(W(R(B(\equiv)(\in B(\sim)(K(B(U))(K(K(B(\sim)))$$
$$W(R(R(R(B(\&)(\in\in)))))))))))))).$$

As can be seen from this example the combinator formulation of a wfe is—at least for those of us (the majority) accustomed to quantifier-variable notation—much harder to scan. It will be noted that there are as many U's in the result as there were quantifiers in the original, and each logical constant occurs just as many times as in the original. Use of W could have allowed us to reduce the epsilons to one, but there is no particular point in doing so.

5 Taming wild combinators A simple categorial grammar is one in which no token of an expression belongs to more than one category. In natural languages, for instance, this appears not to be the case: in the sentence 'Someone admires Ronald' the quantifier 'someone' may be taken as of category $s(s(n))$ to account for the syntactic connection of the phrase 'admires Ronald', while it is also of category $s(n)(s(n)(n))$ to account for the syntactic connection of the phrase 'someone admires'. In the system given here this does not occur: each token of each wfe has only one category. But with the advent of the combinators, *different* tokens of one and the same combinator type belong to different categories. This is easiest to see in the case of the combinator I, which can belong to category $\alpha(\alpha)$ for any α. Hence, although each individual token of a combinator symbol is assigned to exactly one category, the symbol as such is not. The combinators are thus, in a sense, in contrast to the other symbols in our form of Leśniewskian language, of a "wild" category, although the formation rules set limits in each case to the range of categories a combinator may assume. In Leśniewski's own languages, terms themselves may be "wild", in that different occurrences may belong to different categories. We disallowed this option for the sake of simplicity, so the circumstance that combinators are not of fixed category is slightly discomforting.

One consequence of the wildness of combinators is that, when working backwards from a combinator wfe to an equivalent one without combinators by a top-to-bottom application of the equivalence rules (moving in the opposite direction to the one used up to now), we need to know the effective categories of the combinators if we are to know what the wfe that we end up with means. Consider for example the following reduction:

$$\begin{aligned} \llcorner af \lrcorner \ulcorner f(a) \urcorner &= \llcorner af \lrcorner \ulcorner I(f(a)) \urcorner \\ &= \llcorner af \lrcorner \ulcorner B(I)(f)(a) \urcorner \\ &= \llcorner af \lrcorner \ulcorner J(B(I))(fa) \urcorner \\ &= U(J(B(I))). \end{aligned}$$

Notice that this reduction takes no account of the specific category of a (and hence of f): the same reduction goes through for a of any category α and f of the corresponding category $s(\alpha)$. Hence, if we did not know the category of one of the initial variables the wff without variables could not tell us. Even in cases where knowledge of the categories of constants remaining in a reduced formula (as in the case of the reduced form of the short axiom of ontology) enables us to determine uniquely the effective categories of the combinators if the whole is to be well-formed, this only emerges at the end of the top-to-bottom (outside-in) analysis, and until that point a number of tiresome dummy variables have to be dragged through the analysis until we find out that some of them are superfluous.

To overcome these problems, one could adopt the alternative approach of replacing the seven variable-category combinators by seven sheafs or families of fixed-category combinators, where the category is denoted by an index attached to the relevant letter. This is unambiguous and is clearer in cases of possible ambiguity or doubt, but it requires a cumbersome and often heavily redundant notation, and we are still obliged to recognize the structural similarity among the members of these sheafs by using the letters I, R, W, etc. in a quasifunctional way. For these reasons the procedure adopted in this paper appears to me to have the edge, though this is a pragmatic stance which could be overturned by other considerations.

6 Languages with extended categorial grammars

Our considerations have applied to Leśniewskian languages, which, aside from the usual nominalistic treatment of them by Leśniewskians, can be considered as simple type theories: propositional ones in the case of protothetic, nominal and propositional in the case of ontology. So the recipe for eliminating variables of all categories can be applied to languages which are poorer in their categories than those of Leśniewski, with minor alterations to account for the peculiarities of the language in question. Most languages, for instance, allow quantifiers to bind variables only one at a time, which is a special case of the one admitted by Leśniewski. The actual combinators which will need to be employed to obtain a variable-free language of equivalent strength to a given language will vary according to the syntactic categories admitted by the language and the variable-binding primitives of the language. Leśniewski happened to use only one variable binder, the universal quantifier.

A theoretically more important issue is how the approach adopted here applies to extended categorial grammars, in which there are type-change rules. The use of combinators allows us to shed light on what is actually going on in such languages.

The variability of category admitted in this case goes beyond that countenanced by the admission of multigrade combinators. While different combinators' tokens may have different categories, each individual token has only one category. In the case of extended categorial grammars the same expression token may *in situ* have more than one category.

One feature remarked above in connection with the use of Schönfinkel's and Curry's combinators was that they apply only to one-place functors. Schönfinkel and Curry compensate for the loss of multi-place functors by using multi-link ones instead, with binary nesting to the left. In natural languages it often appears to matter little whether the one or the other structural analysis is employed, e.g. whether 'Nancy supports Ronald' is seen as involving the double saturation of a two-place verb 'supports'—category $s(nn)$—by two names, or the saturation of a multi-link 'supports'—category $s(n)(n)$—by one name and the saturation of the resulting one-place predicate by the other name. Considerations of grammaticality and intuitions about what counts as a grammatical unit appear to require more than one analysis. (We are here once more ignoring questions about directionality, which also plays a role in natural languages.) So for many purposes it appears that the same functor token may be assigned more than one category, e.g. $\phi(\alpha\beta)$ or $\phi(\beta)(\alpha)$. Now in fact in a fixed-category language, combinators give us the means to move from one of these categories to another. For instance, if f has category $\phi(\beta)(\alpha)$, then $K(f)$ and $J(f)$ have category $\phi(\alpha\beta)$. So it is always possible to replace a multi-link functor by a closely related multi-place one. Getting from a multi-place to a multi-link functor requires a little more work (our combinators were chosen to make the other direction easy), but it can be done. For instance, if g is of category $\gamma(\alpha\beta)$, then $K(B(I))(K(B(I))K(B(I)))(g)$ has category $\gamma(\beta)(\alpha)$. For suppose g has category $\gamma(\alpha\beta)$, a has category α, and b has category β. Then we have

$$\begin{aligned} g(ab) &= I(g(ab)) \\ &= B(I)(g)(ab) \\ &= K(B(I))(ga)(b). \end{aligned}$$

Putting $X = K(B(I))$ and repeating these three steps on $X(ga)$

$$= X(Xg)(a)(b)$$

and finally repeating them again on $X(Xg)$

$$= X(XX)(g)(a)(b)$$

which when written out in full is

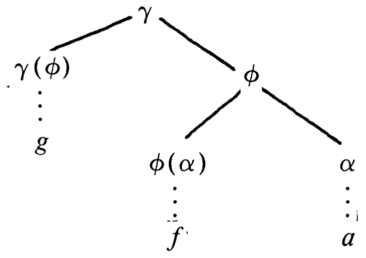
$$= K(B(I))(K(B(I))K(B(I)))(g)(a)(b).$$

So a language in which a functor is allowed to be either multi-place or multi-link (or any of the intermediate categories which occur in more complex cases) may be viewed as one in which all of the fixed-category functors, related to one another by the combinatorial transformations of the kind illustrated, are comprised together as possible categorial guises of a particular expression.

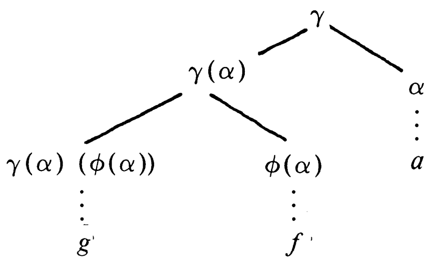
This perspective can also be adopted for the type-change rules suggested in the literature. The most widely accepted rule is known, after its author, as Geach's Rule.[6] It may be stated in this form:

GR If an expression has category $\gamma(\phi)$, it may also be assigned the category $\gamma(\alpha)(\phi(\alpha))$ for any α.

The rationale is this: if f has category $\phi(\alpha)$, g has category $\gamma(\phi)$, and a has category α, then we may analyze the string gfa either thus



or alternatively thus



replacing the right-branching by a left-branching structure and giving g a different category. A moment's thought shows that this is what happens in a fixed-category language when the combinator B is applied. What Geach's Rule effectively does is to allow an expression g to have, as need be, any of the categories allowable to $B(g)$. The obvious generalization of Geach's Rule to multi-place functors simply mimics the categorial effects of B. Once again, the flexible assignment of categories is elucidated in terms of the relationships B imposes among fixed-category expressions.

Another rule to have found widespread currency is Montague's Rule[7]:

MR If an expression has category α, it may also be assigned category $\phi(\phi(\alpha))$ for any ϕ.

Here the rationale is that in a combination such as $f(a)$, where f has category $\phi(\alpha)$ and a has category α, it may be expedient to reverse the roles of functor and argument, i.e., see the structure as $a(f)$. Once again, if f and a have the fixed categories as given above, we can obtain a complex combinator which works on a to yield the same result. Here is the derivation:

$$\begin{aligned} f(a) &= I(f(a)) \\ &= B(I)(f)(a) \\ &= J(B(I))(fa) \\ &= R(J(B(I)))(af) \end{aligned}$$

and using the same combinator $X(XX) = K(B(I))(K(B(I))K(B(I)))$ as before we can bring f out on its own, thus getting

$$f(a) = M(a)(f)$$

where

$$M = K(B(I))(K(B(I))K(B(I)))(R(J(B(I)))).$$

So Montague's Rule effectively allows us to exchange a for $M(a)$, which has the category $\phi(\phi(\alpha))$, without needing to change the expression itself.

What is the significance of this? The categorial flexibility of languages with type-change rules is made up for among fixed-category languages by the presence of the combinators, or rather, just the structure-modifying ones *B*, *K*, *J*, *I*, and *R*. (The variable-eliminating combinators *W* and *U* do not enter the picture.) I find it illuminating to view type-changes or categorial flexibility in this way, since type-change rules are there to facilitate different structural analyses of complex expressions, and this is precisely what the structure-modifying combinators make explicit. One way in which this approach may be helpful is that the use of combinators together with fixed-category expressions allows us a way to plot the limits of categorial freedom in a flexible-category language. Conversely, someone who has seen the attraction of extended categorial grammars will see another rationale for introducing the combinators into fixed-category languages, over and above the wish to provide variable-less alternatives to a language with bound variables. To eliminate the variables from a language with flexible categories is then simply an easier task than for Leśniewskian languages, since we may choose structural guises for an expression which allow the reduction of repetitions and elimination of quantifiers and variables that much more quickly. But we still need combinators to get rid of bound variables and the operators binding them.

As in the case of the issue of fixed-category vs. wild combinators, I leave it as a theoretically undecided issue whether fixed-category or flexible-category languages in general are in any sense more fundamental than the other. It may be that there is no true or false answer to such a vague question, but that it is simply an issue of purpose and expediency.

7 ***Final remarks*** Most logical languages allow for wffs to contain free variables—what Russell called "real" variables. In some cases the use of free variables is merely an abbreviation, and results from omitting universal quantifiers whose scope is the whole formula. Such variables are not really free, and can be eliminated along the lines of this paper. Genuinely free variables on the other hand obviously cannot be eliminated using combinators, since variables can only be induced to disappear in tandem with their binding operators. In this respect free variables behave syntactically exactly like constants. The essential difference between constants and free variables is not a syntactic but a semantic one. There are several ways in which this difference may be treated semantically, and discussion of the various approaches and their respective advantages and disadvantages goes beyond the scope of this paper.

Quine has claimed that his elimination of bound variables from first-order logic offers a "purer analysis of the variable than Schönfinkel's."[8] This is true only if by 'the variable' we understand 'the nominal variable'. But for anyone who—unlike Quine—is prepared to accept bound variables in other categories than that of name, Quine's analysis is insufficiently powerful, and the elimination given here can be offered as a more general analysis of the variable than either Schönfinkel's or Quine's, one which at the same time finally realizes the aim of Ajdukiewicz's categorial grammar in its intended generality.

NOTES

1. See [5]; [2], pp. 125 ff.
2. In [6], for instance, there are more than thirty different shapes of parentheses.
3. The corners have the added advantage of aiding legibility by not crossing the center of a line, and the similarity of upper and lower corners marks their functional association.
4. The perils of ignoring this condition are illustrated in Leśniewski's criticisms of von Neumann; cf. [8], pp. 156 ff.
5. Geach in [5] uses a combinator corresponding to *U*.
6. Cf. [5], p. 485; [2], p. 125.
7. Cf. [2], p. 126.
8. [10], p. 304 of the reprint. See also Quine's commentary on Schönfinkel [11], p. 357 of the English translation.

REFERENCES

[1] Ajdukiewicz, K., "Die syntaktische Konnexität," *Studia Philosophica*, vol. 1 (1935), pp. 1-27. Partially translated as "On syntactical coherence," by P. Geach, *Review of Metaphysics*, vol. 20 (1966), pp. 635-647. Translated as "Syntactic connection," by H. Weber, pp. 207-231 in *Polish Logic, 1920-1939*, edited by S. McCall, Clarendon Press, Oxford, 1967.

[2] van Benthem, J., *Essays in Logical Semantics*, Reidel, Dordrecht, 1986.

[3] Curry, H. B., "Grundlagen der kombinatorischen Logik," *American Journal of Mathematics*, vol. 52 (1930), pp. 509-536 and 789-834.

[4] Curry, H. B. and R. Feys, *Combinatory Logic*, vol. 1, North Holland, Amsterdam, 1958.

[5] Geach, P. T., "A program for syntax," pp. 483-497 in *Semantics for Natural Languages*, edited by D. Davidson and G. Harman, Reidel, Dordrecht, 1972.

[6] Kowalski, J. G., "Leśniewski's ontology extended with the axiom of choice," *Notre Dame Journal of Formal Logic*, vol. 18 (1977), pp. 1-78.

[7] Lejewski, C., "A theory of non-reflexive identity and its ontological implications," pp. 65-102 in *Grundfragen der Wissenschaften und ihre Wurzeln in der Metaphysik*, edited by P. Weingartner, Pustet, Salzburg, 1967.

[8] Leśniewski, S., "Introductory remarks to the continuation of my article: 'Grundzüge eines neuen Systems der Grundlagen der Mathematik'," translated by S. Teichman and S. McCall, pp. 116-169 in *Polish Logic, 1920-1939*, edited by S. McCall, Clarendon Press, Oxford, 1967.

[9] Quine, W. V. O., "Variables explained away," *Proceedings of the American Philosophical Society*, vol. 104 (1960), pp. 343-347. Reprinted as pp. 227-235 in *Selected Logical Papers*, Random House, New York, 1966.

[10] Quine, W. V. O., "Algebraic logic and predicate functors," pp. 214-238 in *Logic and Art: Essays in Honor of Nelson Goodman*, edited by R. Rudner and I. Scheffler, Bobbs-Merrill, Indianapolis, 1972. Revised version printed as pp. 283-307 in *The Ways of Paradox and Other Essays*, Harvard University Press, Cambridge, Massachusetts, 1976.

[11] Schönfinkel, M., "Über die Bausteine der mathematischen Logik," *Mathematische Annalen*, vol. 92 (1924), pp. 305-316. Translated as "On the building blocks of mathematical logic," by S. Bauer-Mengelberg, pp. 355-367 in *From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931*, edited by J. van Heijenoort, Harvard University Press, Cambridge, Massachusetts, 1967.

Institut für Philosophie
Universität Salzburg
Franziskanergasse 1
A-5020 Salzburg, Österreich