A Complete Mechanization of Second-Order Type Theory

TOMASZ PIETRZYKOWSKI

University of Waterloo, Waterloo, Ontario, Canada

ABSTRACT. A generalization of the resolution method for higher order logic is presented. The languages acceptable for the method are phrased in a theory of types of order ω (all finite types)—including the λ-operator, propositional functors, and quantifiers. The resolution method is, of course, a machine-oriented theorem search procedure based on refutation. In order to make this method suitable for higher order logic, it was necessary to overcome two sorts of difficulties. The first is that the unifying substitution procedure—an essential feature of the classic first-order resolution—must be generalized (it is noted that for the higher order unification the proper notion of substitution will include λ-normalization). A general unification algorithm is produced and proved to be complete for second-order languages. The second difficulty arises because in higher order languages, semantic intent is essentially more "interwoven" in formulas than in first-order languages. Whereas quantifiers could be eliminated immediately in first-order resolution, their elimination must be deferred in the higher order case. The generalized resolution procedure which the author produces thus incorporates quantifier elimination along with the familiar features of unification and tautological reduction. It is established that the author's generalized resolution procedure is complete with respect to a natural notion of validity based on Henkin's general validity for type theory. Finally, there are presented examples of the application of the method to number theory and set theory.

KEY WORDS AND PHRASES: theorem-proving, resolution, second-order logic, type theory, unification, matching

CR CATEGORIES: 3.61, 5.21

Introduction

In the last few years interest has increased in mechanization of proving theorems in higher order logical systems. A motivation for this is the following: basic mathematical theories such as set theory, number theory, and in a sense even the theory for the equality relation cannot be finitely axiomatized in the first-order predicate calculus. This fact destroys the possibility (in a first-order approach) of using finite methods which are the core of mechanical theorem-proving. These and other reasons have been discussed in [5, 12] and also [15].

The approaches followed for the higher order theorem-proving differ mainly in the degree of generality. The most general approach is represented by [3, 12]. However, the main objective there is to mechanize the description of mathematical theories rather than proofs of theorems.

The second approach deals with systems based on some equivalents of full

This research was supported by NRC Grant A 5267.

Author's Address: Department of Applied Analysis and Computer Science, University of Waterloo, Waterloo, Ontario, Canada.

ω-calculus with types (see [1, 7, 16]). However, these results can be considered as a basis for only partial mechanization of proofs.

The third approach involves a more complete mechanization of proofs, but restricts the language to a portion of the full ω-order type calculus. We will concentrate on this approach. The first result of this kind was obtained by J. L. Darlington in [5] and then developed in [6]. Both are a generalization of the resolution principle for skolemized theorems of restricted second-order logic. Other attempts to mechanize fragments of set theory are presented in [17].

This paper is a direct continuation of the author's earlier report [13]. The major progress is due to the fact that the preskolemization is not needed here. It has a vital significance for the scope of application of the method since even very simple theorems of set theory cannot be directly skolemized (see Section 6, Examples 3 and 4). It should be noted that both papers are a continuation of Darlington's approach.

Section 1 of this paper provides a specification of the class of languages which can be used to formulate theorems. Roughly speaking these languages are unambiguous, they are type-based with built-in "λ" operator, existential and universal quantifiers, and they have the possibility of using other (defined) operators which bind variables. The theorems are collections of clauses which are finite sets of literals. Literals are well-formed formulas which may be composed, among other things, of propositional connectives and quantifiers. In Section 2 we give the basic rules of object manipulation: substitutions and λ-normalization.

In Section 3 we present a procedure for forming a general unifier of a set of object classes. We prove that this procedure is complete for languages of the second order.

Section 4 introduces the notion of a system (countable collection of clauses) and such properties of systems as inconsistency and unsatisfiability. The latter concept corresponds to the intuitive one as well as to an adaptation of Henkin's notion of validity in general models of type theory [1, 8].

Section 5 is devoted to a method of detecting the unsatisfiability of a system. To this end, we produce a generalized resolution rule which combines Robinson's resolution principle with some rules specific for the higher order like λ-contraction and $\forall$, $\exists$-elimination. This resolution procedure is proved to be complete for second-order languages (complete in the sense of validity given in Section 4). More specifically, the unsatisfiability of a system can be finitely detected by iterating the production of finite resolution subsets.

In Section 6 we give examples of applications of the above method to prove some theorems of number and set theory (among them Cantor's theorem).

In the concluding remarks there are stated some open problems of theoretical and practical nature. An Appendix includes a proof of convergence for the iterative λ-contraction in the discussed languages, and also flowcharts for the various algorithms presented in the paper.

1. *Language*

In this section we specify the class of languages acceptable by the methods of the remaining sections of the paper. Subsections 1.1 and 1.2 give a very general and abstract specification of the structure of the languages. This inclusion seems desirable, as higher order languages have features affecting mechanization unfamiliar

in the typical approach of first-order logic. However, if the reader is familiar with a conventional language for higher order logic and wishes to pass over the abstract specification of the languages (or Subsections 1.1 and 1.2), the example of Subsection 1.3 gives concrete and familiar illustrations of the definitions essential to understanding the results of the remaining sections of the paper.

1.1. Let **A** be a countable set of distinct elements called *characters*. Such notions as *string of characters*, *equality of strings*, *concatenation of strings*, and *occurrence of a string in another string* are assumed to be familiar to the reader and will not be defined. In the following, if not stated otherwise, the small letters a, b, d, f, g, u, v, w, x, y, z will denote strings. The nth ($n \geq 1$) character from the left in a string x will be denoted by $x[n]$, and the set of all strings of **A** by $\mathbf{A}^*$.

1.2. Let $\boldsymbol{L}$ be a subset of $\mathbf{A}^*$ whose elements are called *objects* (e.g. in a typical language objects will be terms, well-formed formulas, etc.). x is *subobject* of y means that x and y are objects and x occurs in y. We shall say that x is a *proper subobject* of y iff x is a subobject of y and $x \neq y$. A proper subobject x of y is a *direct subobject* of y iff there is no proper subobject z of y such that x is a proper subobject of z.

$\boldsymbol{L}$ is called an *acceptable set of objects* iff it satisfies the following conditions:

1.2.1. The characters "$\cdot$", "$\neg$", "$\exists$", "$\forall$", and "λ" are elements of **A**.

1.2.2. There exists a decidable procedure to recognize objects from other strings.

1.2.3. If x, y are distinct direct subobjects of z then there is no string occurring simultaneously in x and y (i.e. x and y do not "overlap"; on the other hand, z might have characters which do not occur in any direct subobject).

1.2.4. There is a set **T** whose elements are called *types*. **T** is defined inductively as follows from a fixed countable set $\mathbf{T}_0 : t \in \mathbf{T}$ iff

(a) $t \in \mathbf{T}_0$, or

(b) $t = (t_1, \cdots, t_n)$ $(n \geq 2)$, where $t_i \in \mathbf{T}$ $(1 \leq i < n)$ and $t_n \in \mathbf{T}_0$.

We shall define a mapping *ord* of **T** into the set of integers by the following formula:

$$\operatorname{ord}(t) =^{\text{def}} \begin{cases} 1 & \text{if } t \in \mathbf{T}_0, \\ \max\limits_{1 \leq i \leq n} \{\operatorname{ord}(t_i)\} + 1 & \text{where } t = (t_1, \cdots, t_n),\ n \geq 2 \text{ if otherwise.} \end{cases}$$

1.2.5. There exists a recursively decidable mapping τ of $\boldsymbol{L}$ into **T** such that if x is a string and y, w, z are objects and x is the result of replacement in y of some nonbound (see 1.2.8) occurrences of z by w, then $\tau(z) = \tau(w)$ implies that x is an object and $\tau(x) = \tau(y)$. $\tau(x)$ is called the *type of* x.

1.2.6. There exist two mutually disjoint subsets **V** and **E** of **A**. Their elements are called respectively *variables* and *existential parameters*. They satisy the following conditions:

1.2.6.1. If $u \in \mathbf{V}$ there exist infinitely many distinct variables $u_1, u_2, \cdots$, such that $\tau(u) = \tau(u_i)$ $(i \geq 1)$.

1.2.6.2. For each $t \in \mathbf{T}$, there exist infinitely many distinct existential parameters $c_1, c_2, \cdots$, such that $\tau(c_i) = t$ $(i \geq 1)$.

1.2.7. If $\tau(f) = (t_1, \cdots, t_n)$ $(n \geq 2)$ and $x_1, \cdots, x_m$ $(0 \leq m < n)$ are objects such that $\tau(x_i) = t_i$ $(1 \leq i \leq m)$, then the concatenation $fx_1 \cdots x_m$ is called a *functional semi-object*. A functional semi-object $fx_1 \cdots x_m$ becomes an object, called a *functional object*, if it is not a part of a functional semi-object $fx_1 \cdots x_m x_{m+1} \cdots x_k$ $(k > m)$, *in the given context*.

If $fx_1 \cdots x_m$ is a functional object with $\tau(f)$ as above, then f, $x_1, \cdots, x_m$ are its direct subobjects and

$$\tau(fx_1 \cdots x_m) =^{\text{def}} \begin{cases} t_n & \text{in case } m = n - 1, \\ (t_{m+1}, \cdots, t_n) & \text{in case } m < n - 1. \end{cases}$$

f is called its *head* and x_i $(1 \leq i \leq n)$ its *i-th argument.*

This definition makes our grammar context sensitive. It could be avoided by introducing full bracketing (see [16]); however it would cause some undesirable technical difficulties in the Unification Algorithm and disturb the simplicity of the notation.

1.2.8. There will be certain classes of objects called *binding scope objects* which will be distinguished in having some (at least one) fixed direct subobjects called *binding designators.* These satisfy the following conditions:

(a) A binding designator must be an occurrence of a variable.

(b) Whenever an object y is obtained from a binding scope object x by replacing an occurrence of a variable u by a variable v (where $\tau(u) = \tau(v)$), then y is a binding scope object, and u is a binding designator for x iff v is a binding designator for y.

In a binding scope object x, if u is a binding designator for x, then every occurrence of u in x is called *bound in* x. Furthermore, such occurrences will be called *bound in* z if x is a subject of an object z. If an occurrence of a variable in an object if not bound it is called *free.*

1.2.9. If $u_1, \cdots, u_m$ $(m \geq 1)$ are distinct variables and x is an object then $\lambda u_1 \cdots u_m \bullet x \bullet$ is a binding scope object, $u_1, \cdots, u_m$, x are its only subobjects, and $u_1, \cdots, u_m$ are its binding designators.

$$\tau(x) =^{\text{def}} \begin{cases} (\tau(u_1), \cdots, \tau(u_m), t_1) & \text{if } \tau(x) = t_1, \text{ where } t_1 \text{ is a basic type,} \\ (\tau(u_1), \cdots, \tau(u_m), t_1, \cdots, t_n) & \text{if } \tau(x) = (t_1, \cdots, t_n). \end{cases}$$

Such an object will be called a λ-*function.*

1.2.10. The set $\mathbf{T}$ contains a type LITERAL. Each object of type LITERAL will be called a *literal* (these are the "well-formed formulas"). If x is a literal then the concatenation $\neg x$ is an object and also a literal. The set of literals will be called the *language of* $\boldsymbol{L}$ or simply the *language.*

1.2.11. If u is a variable, Q is $\forall$ or $\exists$, and x is a literal, then Qux is a binding scope object of type LITERAL, where u and x are its only direct subobjects and u is its binding designator.

Comments. The conditions 1.2.2–1.2.5 and 1.2.10 essentially say that our languages are unambiguous. The condition 1.2.5 additionally guarantees that the application of a substitution is always possible.

Moreover, conditions 1.2.7, 1.2.9, 1.2.10, and 1.2.11 introduce into the syntax of some special objects which respectively correspond to functional expressions, λ-expressions, well-formed formulas, and their existential and universal quantification. The feature of types (which may be interpreted as metalinguistic variables) makes it possible to avoid bracketing without a danger of ambiguity (compare with [4, 7, 16]).

1.3. Example of an acceptable language. What kind of languages does our abstract definition allow? An immediate observation is that a language using "bracket-free" or "Polish prefix notation" will be acceptable. But this is not required except for functional objects with variable heads. Subject to this feature,

and the requirement that $\forall$, $\exists$, λ, and $\neg$ are "built into" the language, virtually *any normal notation* used by mathematicians can be easily adapted to *satisfy the requirements* of our *acceptable languages*.

We shall now give the description of a particular language which we shall use in examples *in the remainder of the paper*. Furthermore, we shall give illustrations of the abstract definitions of Section 1.2.

The *specific characters* of the language (which are *also objects* of the language) are given in Table I.

We now give three rules for recursively generating the set of objects $\boldsymbol{L}$:

(1) Each of the characters in Table I is an object.

(2) New objects of type CLASS or LITERAL are formed according to Table II. The words *class* and *literal* in Table II denote any object of that type.

(In each of the objects formed in Table II, the *direct subobjects* are *exactly those* denoted by the *class* and/or *literal*. Thus, the characters $\{$, $\cap$, $=$, $\vee$, etc., are *not* direct subobjects of, and are *not* included in any proper subobject of the newly formed object, but they can be thought of as characters which identify or characterize the newly formed object. The first three objects on the left side of the table are binding scope objects. The "b.d." and arrow indicates the binding designator, but this notation has no status in the language.)

(3) Finally, it is apparent from the definitions above that new objects may be formed (from those obtained by Table I and Table II) by the *built-in* features of

TABLE I

TYPE	*CLASS*	*(CLASS, CLASS)*	*LITERAL*	*(CLASS, LITERAL)*	*(CLASS, CLASS, LITERAL)*
Intuitive interpretation of the type	individual element	unary function	0-ary predicate	unary predicate	binary predicate
Constants: (existential parameters)	0	A, C, S			
	a, b, c, d	F, G			
Variables	x, y, z	$f, f_1, \cdots$	p, q	P, Q, R	P^2
	$u, v, \cdots$	$g, g_1, \cdots$			

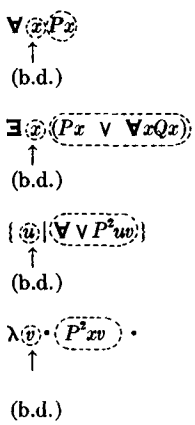
TABLE II

New objects formed of type CLASS	*New objects formed of type LITERAL*
$\{class \mid literal\}$	$class = class$
$\uparrow$ (b.d.)	$class \in class$
$\cup\ class\ class$	$class \subseteq class$
$\uparrow$ (b.d.)	$(literal \wedge literal)$
$\cap\ class\ class$	$(literal \vee literal)$
$\uparrow$ (b.d.)	$(literal \supset literal)$
$class + class$	

the language: forming functional objects, λ-expressions, forming quantified literals with $\forall$ or $\exists$, and placing $\neg$ in front of a literal.

All of the objects of $\boldsymbol{L}$ for our example are given by the above three rules.

Quantified literals and λ-expressions give examples of binding scope objects as well as those of Table II. In each case the direct subobjects will be identified by dotted lines drawn around them (no official status, of course!), and the binding designator will be indicated by "b.d." and an arrow (see Figure 1).



(Here, x in Qx is bound *in* the main object, but not *by* the binding designator of the main object.)

(Notice that the type of this λ-object is (CLASS, LITERAL).)

FIG. 1

2. λ-*Normalization and Substitution*

We start with defining the notion of object classes.

2.1. A pair of occurrences of a variable is called *bound together in an object* x iff these occurrences are bound in x and for each subobject y of x both occurrences are bound in y or neither of them is.

Let x, y be a pair of objects. We shall say that $x \sim y$ iff x results from y by changing only some bound occurrences of variables providing that if a pair of occurrences is bound together in y then the corresponding pair is bound together in x and vice versa, and that the type of corresponding variables remains the same (such change is usually called "alphabetic change of bound variables").

It is easy to prove that the relation "$\sim$" is reflexive, symmetric, and transitive on $\boldsymbol{L}$, and hence it is an equivalence relation. Therefore it partitions the set $\boldsymbol{L}$ into the set of equivalence classes denoted $\boldsymbol{L}'$. Elements of $\boldsymbol{L}'$ we shall call *object classes*. We shall adopt the following notation: If x denotes an object then $\bar{x}$ denotes the object class to which x belongs. The notion of object classes simplifies considerably the definition of substitution and consequently λ-contraction. It will become clear in further discussion (see Subsection 2.2).

The mapping τ can be naturally induced on the set of object classes in the following manner:

$$\text{if} \quad \bar{x} \in \boldsymbol{L}' \quad \text{then} \quad \tau(\bar{x}) = \tau(x).$$

Notice that τ on $\boldsymbol{L}'$ is well defined because $x \sim y$ implies $\tau(x) = \tau(y)$.

2.2. A finite set of ordered pairs $\{\langle u_1, \bar{x}_1\rangle, \cdots, \langle u_n, \bar{x}_n\rangle\}$ is called a *substitution* iff for all i $(1 \leq i \leq n)$ $\bar{x}_i$ is an object class and $\tau(\bar{x}_i) = \tau(u_i)$, and $u_1, \cdots, u_n$ are distinct variables. Substitutions will be denoted by Greek letters: σ, ξ, η, ζ,

and ϵ where ϵ denotes the empty substitution. In order to simplify the notation (to avoid barring and superfluous parentheses) in the further text we shall represent a substitution $\{\langle u_1, \bar{x}_1\rangle, \cdots, \langle u_n, \bar{x}\rangle\}$ as $\{u_1 \leftarrow x_1, \cdots, u_n \leftarrow x_n\}$.

Let x be an object and σ a substitution where $\sigma = \{u_1 \leftarrow w_1, \cdots, u_n \leftarrow w_n\}$ $(n \geq 1)$. $\bar{x}\sigma$ is called the *direct application of* σ *to* x and is defined as an object class $\bar{z}$ such that z is the result of the replacement of all free occurrences of $u_1, \cdots, u_n$ in an object y respectively by $w_1, \cdots, w_n$, provided that $x \sim y$ and each variable which occurs bound in y does not occur free in any w_i $(1 \leq i \leq n)$. It is easy to verify that such a y always exists (it follows from 1.2.5 and 1.2.6) and that $\bar{z}$ is determined uniquely.

However, the above definition of the application of substitution does not provide for the possibility of replacing functional subobjects, which is essential for higher order systems. In order to generalize it suitably we must first introduce the notion of λ-normalization.

2.3. A functional object such that its head is a λ-function we call a λ-*object*. Let b be a λ-object or else a λ-function such that $b = \lambda u_1 \cdots u_n \cdot x \cdot y_1 \cdots y_m$ $(0 \leq m \leq n)$ where x is *not* a λ-function and $y_1 \cdots y_m$ are the arguments of b. Now we define

$$Ap(\bar{b}) = \text{def} \begin{cases} \bar{x}\{u_1 \leftarrow y_1, \cdots, u_m \leftarrow y_m\} & \text{if } m = n, \\ \overline{\lambda u_{m+1} \cdots u_n \cdot x \cdot}\{u_1 \leftarrow y_1, \cdots, u_m \leftarrow y_m\} & \text{if } m < n. \end{cases}$$

Now suppose that b is a λ-function such that $b = \lambda u_1 \cdots u_k \cdot f u_j \cdots u_k \cdot$ $(0 \leq j \leq k$ and we also allow the trivial case that $k = 0)$, where $f u_j \cdots u_k$ is a functional object and $u_j, \cdots, u_k$ are some of its arguments which have no free occurrences in f. Then we define

$$Rd(\bar{b}) =^{\text{def}} \begin{cases} \bar{f} & \text{if } j = 1, \\ \overline{\lambda u_1 \cdots u_{j-1} \cdot f \cdot} & \text{otherwise.} \end{cases}$$

Finally for any λ-object or λ-function b we define:

$$|\, \bar{b} \,|_\lambda =^{\text{def}} Rd(Ap(\bar{b}))$$

We call $|\, \bar{b} \,|_\lambda$ the λ-*contraction* of $\bar{b}$.

The λ-contraction always exists (due to the fact that it is not an object but the object class). Its uniqueness clearly follows from uniqueness of the direct substitution application.

The above definition of λ-contraction is adopted from the original Church definition of λ-conversion [4] with some modifications.

Let x, y be objects. We shall say that x is *immediately convertible* into y and denote it *conv*(x, y) iff y is the result of the replacement of a λ-object z occurring in x by an element of the object class $|\, \bar{z} \,|_\lambda$. Furthermore we shall define that x is *convertible* into y and denote it *conv*$^*(x, y)$ iff conv(x, y) or there exists an object z such that conv$^*(x, z)$ and conv(z, y).

An object x is called *normal* iff for each of its subobjects b which is a λ-object or a λ-function we have $\bar{b} = |\, \bar{b} \,|_\lambda$. The following theorem relates the above notions and describes an important property of our languages.

THEOREM 1. *To each object class* $\bar{x}$ *there corresponds a unique object class* $\bar{y}$ *such that* y *is a normal object and conv*$^*(x, y)$ *holds.*

The proof of this theorem is given in Appendix I. On the basis of this theorem

we shall define a mapping *norm* on L' as follows:

$$\text{norm}(\bar{x}) =^{\text{def}} \bar{y},$$

where $\bar{x} \in L'$ and y is a normal object such that $\text{conv}^*(x, y)$ holds.

Finally, we shall define the *application of substitution* σ *to an object class* $\bar{x}$ by the formula below:

$$\bar{x} \circ \sigma =^{\text{def}} \text{norm}(\bar{x}\sigma).$$

In the case of the first-order language, the application and the direct application of substitutions are identical.

It should be noted that the above definition would be quite awkward if applied directly to objects instead of object classes.

Example.

$$\overline{\forall u(u \in x \wedge fu \in a)} \circ \{x \leftarrow Au, f \leftarrow \lambda v \cdot (a + v) \cdot\} = \overline{\forall z(z \in Au \wedge (a + z) \in a)}.$$

It is easy to notice that if we were to deal with objects instead of object classes, the above application of substitution would not be feasible: inserting Au in place of x would bind the previously free u. The use of object classes makes the generalized application of substitution a total operation, where in the case of objects it would only be partial.

2.4. Clearly generalized substitutions can be identified with mappings of L' into the set of normal object class. In order to preserve this property of substitutions we shall follow Robinson [14] in defining the composition of substitutions.

Let ξ, η be a pair of substitutions such that $\xi = \{u_1 \leftarrow x_1, \cdots, u_n \leftarrow x_n\}$ and $\eta = \{v_1 \leftarrow y_1, \cdots, v_m \leftarrow y_m\}$ where $m, n \geq 0$. Then the *composition of substitutions* ξ, η is defined as follows:

$$\xi \circ \eta =^{\text{def}} \{u_1 \leftarrow x_1 \circ \eta, \cdots, u_n \leftarrow x_n \circ \eta, v_{i_1} \leftarrow y_{i_1}, \cdots, v_{i_k} \leftarrow y_{i_k}\},$$

where $\{v_{i_1}, \cdots, v_{i_k}\} = \{v_1, \cdots, v_m\} - \{u_1, \cdots, u_n\}$. It can be proved that for each object class $\bar{x}$ and substitutions ζ, η, ξ, we have

$$(\bar{x} \circ \xi) \circ \eta = \bar{x} \circ (\xi \circ \eta) \quad \text{and} \quad (\xi \circ \eta) \circ \zeta = \xi \circ (\eta \circ \zeta).$$

3. *Unification*

In this section we give a unification procedure which will prove to be adequate for second-order logic. The term "matching procedure" has also been used elsewhere. The reader should recall that a mechanization of logic, in the style initiated by Robinson [14], replaces the many varied axioms and rules of inference of conventional logic by a single powerful rule of inference called resolution. This rule of inference must consequently be quite complex and must in a sense embody the conventional logical procedures which depend on instantiation and generalization. This task is, of course, accomplished by unification.

3.1. Let X be a set of object classes. We shall call a substitution σ a *unifier* of X iff for all $\bar{y}$, $\bar{z} \in X$ we have $\bar{y} \circ \sigma = \bar{z} \circ \sigma$. A set Ω of unifiers of X is called a *general unifier* of X iff for each unifier ξ of X there exists a pair of substitutions σ, η such that $\sigma \in \Omega$ and $\xi = \sigma \circ \eta$.

Remark. Intuitively, finding a general unifier for a set X of object classes

amounts to generating all (in a certain canonical sense) possible ways in which the objects can be deformed into each other through substitution. A trivial, but inessential, assumption for the discussion below is that all of the object classes in X are of the same type (since substitution preserves type). Also, the reader may realize that the main task of producing a general unifier for a (finite) set will reduce to producing a general unifier for two object classes.

The reader may recall that producing a general unifier in first-order logic can be accomplished by a rather direct process of substituting terms for individual variables until the unification is accomplished. However, in higher-order logic we should expect that the procedure must be more complicated because there will usually be a greater variety of ways in which object classes can be unified.

3.2. Suppose we wish to unify two object classes $\bar{a}_0$ and $\bar{b}_0$. We will proceed by attempting (through substitution) to make $\bar{a}_0$ and $\bar{b}_0$ more similar—in stages. At each stage, one or several different attempts (or possibly none) might be applicable. (What we mean by "attempts" will be specified soon, and called *elementary unification substitutions*.) Thus our procedure to unify $\bar{a}_0$ and $\bar{b}_0$ will appear as a tree, each node of which will be a triple $\langle \bar{a}, \bar{b}, \sigma \rangle$ where $\bar{a}, \bar{b}$ are object classes (representing our progress thus far in attempting to unify $\bar{a}_0$ and $\bar{b}_0$) and σ will be a substitution (showing how we obtained $\bar{a}$ and $\bar{b}$ from $\bar{a}_0$ and $\bar{b}_0$). Such a triple will be called a *progress triple*. Of course, the initial node of the tree will be the triple $\langle \bar{a}_0, \bar{b}_0, \sigma_0, \rangle$ which we will call the *initial triple*. (We remark that σ_0 could be thought of as ϵ, the empty substitution, but later when we wish to unify more than two object classes, it is convenient to be able to allow a nonempty initial substitution.) Each node will be connected to each (if any) of the subsequent nodes of the tree by an elementary unification substitution.

3.3. There will be two cases when a progress triple $\langle \bar{a}, \bar{b}, \sigma \rangle$ will have no subsequent nodes. First, if $\bar{a} = \bar{b}$, then we will call $\langle \bar{a}, \bar{b}, \sigma \rangle$ a *unified triple*. Second, if $\bar{a} \neq \bar{b}$, but no elementary unification substitution applies, we will call $\langle \bar{a}, \bar{b}, \sigma \rangle$ a dead end. In either case, that branch of the tree ends with $\langle \bar{a}, \bar{b}, \sigma \rangle$.

3.4. Elementary unification substitutions. We now explain what *elementary unification substitutions* can be applicable to a progress triple $\langle \bar{a}, \bar{b}, \sigma \rangle$. We may assume that $\bar{a} \neq \bar{b}$. Our first task is always to find the first character (from the left) at which $\bar{a}$ and $\bar{b}$ "essentially" differ. Formally this means that we find the first n (≥ 1) such that for all objects a and b belonging to object classes $\bar{a}$ and $\bar{b}$, respectively, $a[i] = b[i]$ $(1 \leq i \leq n-1)$ and $a[n] \neq b[n]$. Of course, by attempting alphabetical changes of variables, it is easy to find such n and particular objects a and b. Each of the rules below will be symmetric for both a and b, but we will only state each for a. Granting this, each rule depends on how $a[n]$ occurs in a, but we stress that in a given situation more than one rule might be applicable, or even that a particular rule might be applicable in more than one way. In each case the *elementary unification substitution* will be denoted by ξ and will consist of replacing a single free variable g occurring in a by a term (usually a λ-function) of the appropriate type. (Informally we remark that g will be $a[n]$ in the second and third rules, but will not be $a[n]$ in the first rule.)

(1) *Elimination Rule.* (Intuitively, in this case, we will replace the free head of some functional subobject in which $a[n]$ occurs in such a way as to "eliminate" $a[n]$.) Formally, find a functional subobject $gx_1 \cdots x_p$ of a where g is a free variable and such that $a[n]$ occurs in x_k $(1 \leq k \leq p)$. (So automatically g occurs to the left of $a[n]$.) Now let f be a new variable which does not occur in a or b and which

is of the proper type for the substitution:

$$\xi = \{g \leftarrow \lambda u_1 \cdots u_p \cdot f u_1 \cdots u_{k-1} u_{k+1} \cdots u_p \cdot\},$$

where $\tau(u_i) = \tau(x_i)$. (Notice that ξ will have the intended effect of "eliminating" $a[n]$.)

(2) *Projection Rule.* (Intuitively, in this case, we have found that $a[n]$ is itself the free variable head of a functional subobject, and we replace this entire subobject by one of its arguments of the appropriate type.) Formally, $a[n]$ is a free variable g in the functional subobject $gx_1 \cdots x_p$ of a, and also, for some i $(1 \leq i \leq p)$ we find that $\tau(x_i) = \tau(gx_1 \cdots x_p)$. Then we put

$$\xi = \{g \leftarrow \lambda u_1 \cdots u_p \cdot u_i \cdot\},$$

where $\tau(u_i) = \tau(x_i)$. (Notice that the substitution is of the proper type and has the intended effect of "projecting" onto an argument.)

(3) *Imitation Rule.* (In this case, we have again found (as in rule (2)) that $a[n]$ is the free variable head g of a functional subobject $gx_1 \cdots x_p$ in a where $p \geq 0$, so we also allow the case that g is an "individual" variable. However, now the substitution we make will also depend on the particular subobject y of b which begins with $b[n]$. Intuitively, we will replace g by a function which will "imitate" the structure of y.) Having found g and y in this situation we specify what we mean by an "imitation" y^* of y: $y^* = y$ in case y has no direct subobjects.

Otherwise $y = \alpha_1 y_1 \alpha_2 \cdots \alpha_m y_m \alpha_{m+1}$ $(m \geq 1)$ where the y_i are its direct subobjects which are not occurrences of variables bound in b, and α_i are concatenations of some constant letters and bounded variables. (It reflects the fact that our grammar does not require that y is a concatenation of its subobjects.) In this case $y^* = \alpha_1 h_1 \alpha_2 \cdots \alpha_m h_m \alpha_{m+1}$, where $h_i = f_i v_1 \cdots v_k u_1 \cdots u_p$ $(1 \leq i \leq m)$, $v_1, \cdots, v_k$ $(k \geq 0)$ are all the binding designators of y, and $v_1, \cdots, v_k, u_1, \cdots, u_p$ are distinct. Furthermore, $\tau(u_j) = \tau(x_j)$ $(1 \leq j \leq p)$ and each f_i is a new variable of the appropriate type which does not occur in a or b.

Finally we put

$$\xi = \{g \leftarrow \lambda u_1 \cdots u_p \cdot y^* \cdot\}.$$

We remark that the object of imitation is to force the eventual task of unification to subobjects—which are less "complex" syntactically. As this rule is the most particularly characteristic of our method, and also the most difficult to understand, examples illustrating its necessity will be given soon.

The above three rules are most frequently needed in the unification process. However, in order to preserve the completeness of the unification procedure we shall add two more rules which capture more pathological situations.

(4) *Repetition Rule.* In this case, like in rule (2), $a[n]$ is the free variable head g of a functional subobject $gx_1 \cdots x_p$ in a where $p \geq 1$. (Intuitively this rule adds an argument to this functional subobject which may be needed later.) Formally

$$\xi = \{g \leftarrow \lambda u_1 \cdots u_p \cdot f u_1 \cdots u_p u_l \cdot\},$$

where $1 \leq l \leq p$ and f is a new variable of appropriate type.

(5) *Identification Rule.* In this case, both $a[n]$ and $b[n]$ are free variable heads of respective functional subobjects $gx_1 \cdots x_p$ and $hy_1 \cdots y_r$ of a and b. This rule is

supposed to guarantee detection of unifying substitutions which are built jointly of subobjects of $x_1 \cdots x_p$, $y_1 \cdots y_r$.

$$\xi = \{g \leftarrow \lambda u_1 \cdots u_p \cdot f(u_1 \cdots u_p g_1(u_1 \cdots u_p) \cdots g_r(u_1 \cdots u_p)) \cdot$$
$$h \leftarrow \lambda v_1 \cdots v_r \cdot f(h_1(v_1 \cdots v_r) \cdots h_p(v_1 \cdots v_r) v_1 \cdots v_r) \cdot\}.$$

(The parentheses in the formula above are not a part of the expressions and are introduced only for visual convenience.)

This completes our list of elementary unification substitutions, as is formally characterized in the general theorems that follow. Some heuristic modifications will be mentioned in Subsection 3.8.

3.5. Next, for each elementary unification substitution ξ which is applicable to a progress triple $\langle \bar{a}, \bar{b}, \sigma \rangle$ we obtain the subsequent progress triple $\langle \bar{a}_\xi, \bar{b}_\xi, \sigma_\xi \rangle$ determined by ξ as follows:

$$\bar{a}_\xi = \bar{a} \circ \xi, \qquad \bar{b}_\xi = \bar{b} \circ \xi, \quad \text{and} \quad \sigma_\xi = \sigma \circ \xi,$$

with the particular modification that if ξ includes any substitutions of the form $f \leftarrow x$ where f is a variable which does not occur in the initial triple $\langle a_0, b_0, \sigma_0 \rangle$, then such $f \leftarrow x$ may be deleted from σ_ξ as superfluous. (That is, such f was introduced somewhere midway in the branch leading from $\langle \bar{a}_0, \bar{b}_0, \sigma_0 \rangle$ to $\langle \bar{a}_\xi, \bar{b}_\xi, \sigma_\xi \rangle$ and now may be forgotten. Later we will refer to such variables f as "new" variables.)

3.6. Now, given an initial triple $\langle \bar{a}_0, \bar{b}_0, \sigma \rangle$, form the *entire unification tree* $T\langle \bar{a}_0, \bar{b}_0, \sigma \rangle$ inductively by applying each applicable elementary unification substitution to each progress triple and thus obtaining its subsequent triples. We remark that it is by no means obvious whether or not this tree is finite. Next define:

$$S\langle \bar{a}_0, \bar{b}_0, \sigma \rangle$$
$$= \{\eta \mid \eta \text{ is the substitution designated in some unified triple of } T\langle \bar{a}_0, \bar{b}_0, \sigma \rangle\}.$$

(Recalling the definition of Subsection 3.3; one may see intuitively that we form S by collecting the "successful" substitutions from the ends of the branches of T.)

3.7. Finally, a natural proposal for a unifier for two object classes $\bar{a}_1$ and $\bar{a}_2$ would be

$$\Omega\{\bar{a}_1, \bar{a}_2\} =^{\text{def}} S\langle \bar{a}_1, \bar{a}_2, \epsilon \rangle.$$

And we may extend this definition inductively to more than two object classes by:

$$\Omega\{\bar{a}_1, \cdots, \bar{a}_i, \bar{a}_{i+1}\} =^{\text{def}} \{\sigma \circ \eta \mid \sigma \in \Omega\{\bar{a}_1, \cdots, \bar{a}_i\} \text{ and } \eta \in S\langle \bar{a}_1 \circ \sigma, \bar{a}_{i+1} \circ \sigma, \sigma \rangle\}.$$

Of course, this says that we extend Ω to one more object class $\bar{a}_{i+1}$ by attempting to unify $\bar{a}_1$ and $\bar{a}_{i+1}$ *subject* to the various substitutions σ which have already been found successful up through $\bar{a}_i$. Notice also that in this definition we could just as well try to unify $\bar{a}_{i+1} \circ \sigma$ with $\bar{a}_2 \circ \sigma, \cdots$, or $\bar{a}_i \circ \sigma$, since these are all the same as $\bar{a}_1 \circ \sigma$. (We remark that this construction of Ω might fail even to be the limit of effective processes because some $T\langle \bar{a}, \bar{b}, \sigma \rangle$ upon which its inductive definition depends might fail to be finite. This worry will not concern us now. However, it should be apparent that Ω could be given as the limit of effective processes by a slight complication of the definition.)

3.8. Heuristic modifications. Above we have stated the proposed Unification Algorithm in a simple, general form. This has been desirable to reflect the intuition behind the formulation of the elementary unification substitution rules. However, the reader will realize that in particular applications many duplications may be produced. We now list a few heuristic modifications which we will incorporate in the examples. Each of these modifications has the effect of removing superfluous elementary unification substitutions which would be specified by the rules of Section 3.4.

(a) In the Imitation Rule: If $a[n] = g$ is an "individual" variable (i.e. the number of arguments $p = 0$) then we may set the "imitation" of y, $y^* = y$.

(b) In the Imitation Rule: If y is a functional object with head h, then we may use h instead of $f_1u_1 \cdots u_{p+k}$ in forming y^*.

The two modifications just given will have the effect of removing some of the unnecessary "descendants" of the particular substitution modified. The next modification, however, is of a more complicated sort: In this case, the applicability of a particular elementary unification substitution ξ to a progress triple $\langle \bar{a}, \bar{b}, \sigma \rangle$ will enable us to disregard other substitutions which might, by Section 3.4, have been applicable to the *same* $\langle \bar{a}, \bar{b}, \sigma \rangle$ (i.e. this modification works on a "horizontal" level rather than a "vertical" level as with (a) and (b) above).

(c) Suppose we have found the Imitation Rule applicable where y (as in the rule) is an "individual" variable (i.e. type, $\tau(y) = t_0$) and furthermore, there is no free occurrence of y in $gx_1 \cdots x_p$. Then we keep *this* imitation substitution *exactly as is* (i.e. $\xi = \{g \leftarrow \lambda u_1 \cdots u_p \cdot y \cdot\}$ and $\langle \bar{a}_\xi, \bar{b}_\xi, \sigma_\xi \rangle$ is formed normally). However, we may now forget *all other* instances of the *Projection Rule* and the *Imitation Rule* which might normally be applicable starting with *either* $a[n]$ or $b[n]$ (recall the rules are symmetric). This is because the one imitation substitution will produce the most general unifier possible.

3.9.

Example 1. In the following we give examples of each of the elementary unifications:

Eliminations:
$\langle f^2ab, f^2xy, \epsilon \rangle \rightarrow \langle f_1b, f_1y, \{f^2 \leftarrow \lambda uv \cdot f_1v \cdot\} \rangle$,
$\langle f^2ab, f^2ay, \epsilon \rangle \rightarrow \langle f_1a, f_1a, \{f^2 \leftarrow \lambda uv \cdot f_1u \cdot\} \rangle$,
$\langle fx, fa, \epsilon \rangle \rightarrow \langle y, y, \{f \leftarrow \lambda u \cdot y \cdot\} \rangle$.
Projections:
$\langle f^2xy, a, \epsilon \rangle \rightarrow \langle x, a, \{f^2 \leftarrow \lambda uv \cdot u \cdot\} \rangle$,
$\langle f^2xy, b, \epsilon \rangle \rightarrow \langle y, b, \{f^2 \leftarrow \lambda uv \cdot v \cdot\} \rangle$.
Imitations:
$\langle fx, Aa, \epsilon \rangle \rightarrow \langle a, a, \{f \leftarrow \lambda u \cdot a \cdot\} \rangle$,
$\langle fx, a, \epsilon \rangle \rightarrow \langle Aa, Aa, \{x \leftarrow Aa\} \rangle$,
$\langle x, Ab, \epsilon \rangle \rightarrow \langle Af_1x, Ab, \{f \leftarrow \lambda u \cdot Af_1u \cdot\} \rangle$,
$\langle f^2xy, (a + b), \epsilon \rangle \rightarrow \langle (f_1^2xy + f_2^2xy), (a + b), \{f^2 \leftarrow \lambda uv \cdot (f_1^2uv + f_2^2uv) \cdot\} \rangle$.

Example 2. In Figure 2 we show the elementary unifications applied to a single node of a unification tree. The example is chosen to illustrate how many elementary unifications may all apply to the same progress triple.

Example 3. In Figure 3 there is presented a complete unification tree. This tree has five terminals. Terminals 1, 2, 4, and 5 represent unified triples and 3 is a "dead end."

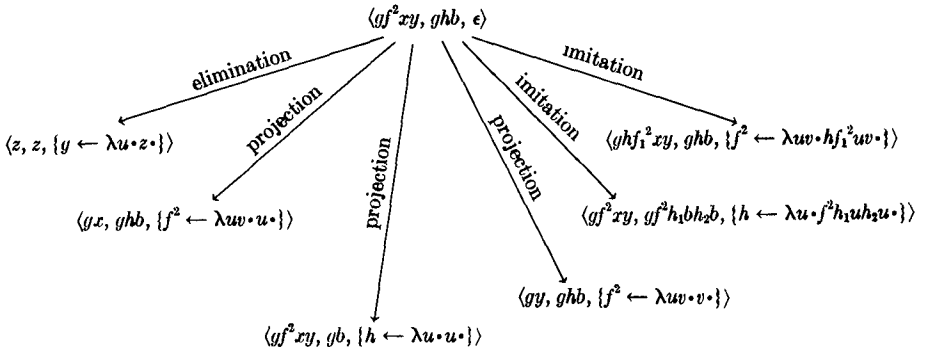


FIG. 2

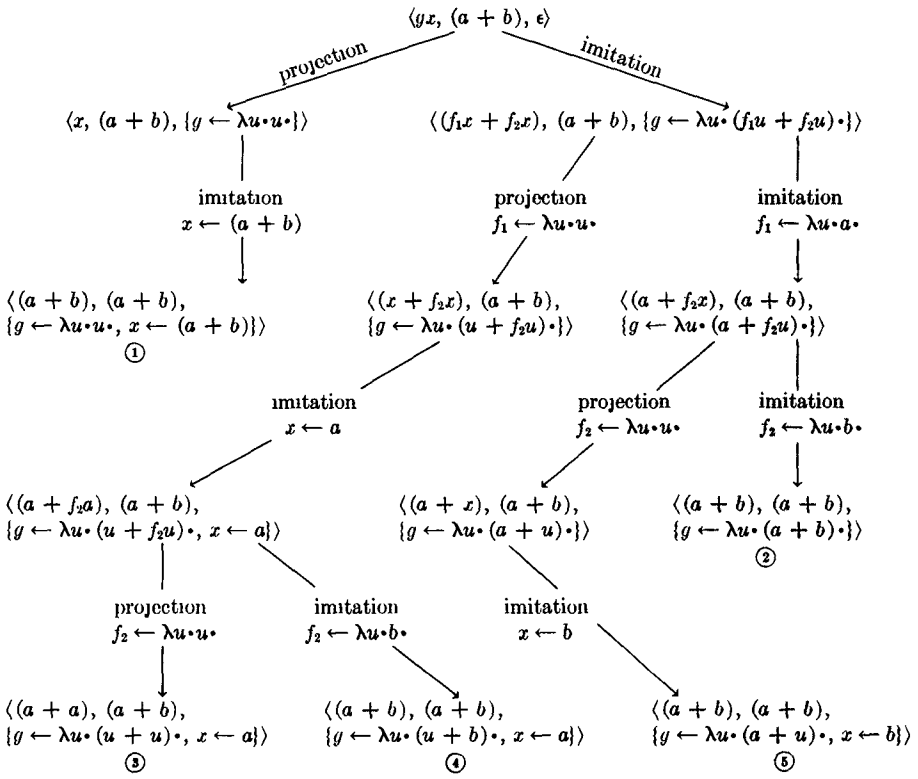


FIG. 3

3.10. Order and complexity of languages. In this subsection we introduce some notions which are pertinent to the main theorem of this section: to establish the applicability of the unification procedure for second-order languages.

Let $fx_1 \cdots x_m$ $(m \geq 1)$ be a functional object. We shall say that

$$\text{ford}(fx_1 \cdots x_m) =^{\text{def}} \max_{1 \leq i \leq m} \{\text{ord}\ \tau(x_i)\} + 1.$$

(Recall the definition of ord in Subsection 1.2.4.) We now define that a *set X of object classes has order k iff*

$$k = \max\{\text{ford}(fx_1 \cdots x_m)\,;\ \text{where } \overline{fx_1 \cdots x_m} \in X \text{ and } f \text{ is variable.}\}$$

Notice that this notion of order depends only on variables, and thus seems to suggest the "complexity" of possible substitutions into X. Indeed, it will be very useful in this subsection to have a suitable measure of the complexity of substitutions. As a first step we will introduce the "complexity" of an object which expresses the extent to which its subobjects are "nested."

Define a mapping *comp* of L into the set of integers as follows:

$$\text{comp}(x) =^{\text{def}} \begin{cases} 0 & \text{if } x \text{ has no direct subobjects} \\ \max_{x_i \in X}\{\text{comp}(x_i)\} + 1 & \text{where } X \text{ is the set of all direct subobjects of } x. \end{cases}$$

Now we shall define a mapping *Comp* of the set of substitutions into the set N^* of ordered tuples of integers (this set is described in the proof of Theorem 1; see Appendix I):

$$\text{Comp}(\sigma) =^{\text{def}} (n_1, \cdots, n_k),$$

where n_i $(1 \leq i \leq k)$ is the total number of subobjects of complexity i which occur in any of $x_1, \cdots, x_m$ where $\sigma = \{u_1 \leftarrow x_1, \cdots, u_m \leftarrow x_m\}$.

This measure of complexity for substitutions is somewhat difficult to grasp because it is not apparently linear. This feature is indispensible for the "naturality" of the notion in our main application. The set N^* is, of course, given the usual well-order of transfinite ordinal type ω^ω which is denoted as $\prec$ in Appendix I. It is essential here only to note that $\prec$ is a well-order and that this order preserves the "naturality" of the complexity of substitutions. In forming $\text{Comp}(\sigma)$, notice that once an object x is counted in an n_i, then any of its subobjects can only be counted in some n_j for $j < i$. Thus for example,

$$\text{Comp}(\{g \leftarrow \lambda u \cdot (u + Au) \cdot, x \leftarrow Afa\}) = (2, 2, 1)$$

because there are two subobjects (Au and fa) with complexity 1; and two subobjects ($(u + Au)$ and Afa) with complexity 2; and one subobject

$$(\lambda u \cdot (u + Au) \cdot)$$

with complexity 3.

We will present one more notion *informally* to help motivate the proof of the main theorem of Subsection 3.11. We want to compare how different substitutions affect object classes $\bar{a}_0$ and $\bar{b}_0$ which we are trying to unify. Let V be a fixed set of variables (in application, V will generally be the free variables appearing in $\bar{a}_0$, $\bar{b}_0$). We will say that a substitution τ is *no more specific than* σ *for* V (or "more general") and write $\tau \leq_V \sigma$ in case there exists a substitution η such that

$$\sigma - \text{new}_V = \tau \circ \eta - \text{new}_V,$$

where $u \leftarrow x \in \text{new}_V$ iff $u \notin V$. Intuitively, $\tau \leq_V \sigma$ means that τ can be deformed into σ by composition of substitutions, *provided* that we restrict our attention only to the parts of the substitutions which replace variables in V. Clearly, $\leq_V$ is a reflexive partial order with $\epsilon \leq_V \sigma$. With this notation, if σ is a unifier for $\bar{a}_0$ and $\bar{b}_0$, then any general unifier for them must have a τ with $\tau \leq_V \sigma$.

The relationship between the "specific" order $(\leq_V)$ and the complexity $(\prec)$ of substitutions of the form $\sigma - \text{new}_V$ points out a clear distinction between first-order languages and those of higher order. For first-order languages $\tau \leq_V \sigma$

implies $\text{Comp}(\tau - \text{new}_V) \leq \text{Comp}(\sigma - \text{new}_V)$. This is because composition of substitutions involving only first-order variables cannot decrease complexity. However, this is not the case for languages with higher order variables because we can have situations such as the following:

$$\tau = \{x \leftarrow ffx\}, \qquad \text{Comp}(\tau) = (1, 1),$$

$$\eta = \{f \leftarrow \lambda u \cdot u \cdot\}, \qquad \text{Comp}(\eta) = (1),$$

but

$$\tau \circ \eta = \{x \leftarrow x, f \leftarrow \lambda u \cdot u \cdot\}, \qquad \text{Comp}(\tau \circ \eta) = (1).$$

And if we have $V = \{x\}$, then we even have $\tau \leq_V \tau \circ \eta - \text{new}_V = \epsilon$. Nevertheless, the proof of the main theorem given in Subsection 3.11 will involve relating $\leq_V$ and $\prec$ for applications connected with the unification procedure.

3.11. Completeness of unification for second-order languages.

THEOREM 2. *If* $X = \{\bar{a}_1, \cdots, \bar{a}_m\}$ $(m \geq 2)$ *where* $\bar{a}_1, \cdots, \bar{a}_m$ *are object classes of order not larger than two then* $\Omega\{\bar{a}_1, \cdots, \bar{a}_m\}$ *is a general unifier of* X.

PROOF. The proof will be presented in three major parts. We will now summarize the idea of the proof. Notice from the definition of a general unifier that this notion is nontrivial only for sets of object class which *have some* unifier. So at first, we will concentrate our attention on object classes $\bar{a}_0$ and $\bar{b}_0$ which have some unifier σ. The $\bar{a}_0$, $\bar{b}_0$ and σ are arbitrary but fixed. Motivated by the example of σ we will put the unification procedure to work on $\bar{a}_0$ and $\bar{b}_0$ to produce inductively a *particular* branch of the unification tree $T\langle \bar{a}_0, \bar{b}_0, \epsilon \rangle$. The kth level of this branch will be determined by a substitution σ_k (yielding $\langle \bar{a}_0 \circ \sigma_k, \bar{b}_0 \circ \sigma_k, \sigma_k \rangle$). We fix the set V (as in Subsection 3.10) as the variables free in $\bar{a}_0$, $\bar{b}_0$ and those being replaced by σ. Now, each of the σ_k which we produce inductively has the property of being *no more specific than* σ *for* V $(\sigma_k \leq_V \sigma)$. In particular, we will exhibit σ_k and η_k such that $\sigma = \sigma_k \circ \eta_k - \text{new}_V$. We can think of η_k as being a *reminder* for σ_k because it carries the essential remaining information about the form of σ which has not already been reduced (thus, $\eta_0 = \sigma$ because we must have $\sigma_0 = \epsilon$). Now clearly, if some σ_k unifies $\{\bar{a}_0, \bar{b}_0\}$, then σ is taken care of in producing a general unifier for $\{\bar{a}_0, \bar{b}_0\}$ because $\sigma_k \leq_V \sigma$. The second part of the proof involves showing that indeed, some σ_k does unify $\bar{a}_0$ and $\bar{b}_0$. The idea is that if σ_{k-1} did not already unify $\bar{a}_0$ and $\bar{b}_0$ then the new reminder η_k produced must have $\text{Comp}(\eta_k)$ *strictly* $\prec \text{Comp}(\eta_{k-1})$. And since $\prec$ is a well-order, we conclude that the production of distinct σ_k must end. The third part of the proof extends the result to sets of several object classes.

Now we give the first part of the proof. We have $\bar{a}_0$, $\bar{b}_0$ and σ as above and we will produce the inductive sequence σ_k, η_k such that

$$\langle \bar{a}_0 \circ \sigma_k, \bar{b}_0 \circ \sigma_k, \sigma_k \rangle \in T\langle a_0, b_0, \epsilon \rangle \tag{1}$$

and

$$\sigma = \sigma_k \circ \eta_k - \text{new}(\eta_k), \tag{2}$$

where $u \leftarrow x \in \text{new}(\eta_k)$ iff u is a "new" variable as defined in the unification procedure in Subsection 3.5. Clearly $\text{new}(\eta_k) \subseteq \text{new}_V$ for V as specified in the last paragraph.

To start the induction, of course $\sigma_0 = \epsilon$ and $\eta_0 = \sigma$. Let us assume (1) and (2) are valid for $k = i$ $(i \geq 0)$. Let us also denote $\bar{a} = \bar{a}_0 \circ \sigma_i$, $\bar{b} = \bar{b}_0 \circ \sigma_i$. First we

shall prove that if $\bar{a}_0 \circ \sigma_i \neq \bar{b}_0 \circ \sigma_i$ then there exist substitutions ξ and η such that

$$\sigma_{i+1} = \sigma_i \circ \xi - \text{new}(\xi) \tag{3}$$

$$\eta_i = \xi \circ \eta - \text{new}(\eta). \tag{4}$$

Let n be the integer as defined in the Unification Algorithm and let c, d be subobjects of a, b respectively, such that their leftmost characters are $a[n]$ and $b[n]$. Since $a[n] \neq b[n]$ the only way to unify a and b is to:

(α) eliminate c, d from a, b, or

(β) unify c and d.

In case (α) there must exist integers m, l ($1 \leq m \leq n$) such that $a[m]$ is a free occurrence of variable and the head of some functional subobject x of z, c is a subobject of the lth ($1 \leq l \leq p$) argument of x, and $a[m] \leftarrow \lambda u_1 \cdots u_p \cdot v \cdot \in \eta_i$ ($p \geq 1$) such that u_l does not occur in v. This last condition is necessary to eliminate c from $\bar{a} \circ \eta_i$; however for objects of order higher than two it could be achieved otherwise (see [7, pp. 11–14]). These properties imply that there exists a "new" variable f_1 such that $\{a[m] \leftarrow \lambda u_1 \cdots u_p \cdot v \cdot\} = \{a[m] \leftarrow \lambda u_1 \cdots u_p \cdot f_1 u_1 \cdots u_{l-1} u_{l+1} \cdots u_p \cdot\} \circ \{f_1 \leftarrow \lambda u_1 \cdots u_{l-1} u_{l+1} \cdots u_p \cdot v \cdot\}$. In this case if we define

$\xi = \{a[m] \leftarrow \lambda u_1 \cdots u_p \cdot f_1 u_1 \cdots u_{l-1} u_{l+1} \cdots u_p \cdot\}$, and
$\eta = \{f_1 \leftarrow \lambda u_1 \cdots u_{l-1} u_{l+1} \cdots u_p \cdot v \cdot\} \cup (\eta_i - \{a[m] \leftarrow \lambda u_1 \cdots u_p \cdot v \cdot\})$,

then it is easy to verify that eq. (4) is satisfied.

In the case (β) let $c \circ \eta_i = d \circ \eta_i$. Obviously at least one of $a[n]$, $b[n]$ must be a variable. Let us assume that it is $a[n]$. (If $b[n]$ is also variable the following reasoning should be repeated with appropriate modifications.) In this case c is a functional object (possibly a variable). Assuming that a, b are normal, in order to unify c with d there must exist v such that $a[n] \leftarrow \lambda u_1 \cdots u_p \cdot v \cdot \in \eta_i$ ($p \geq 0$). Now one of the following subcases must occur:

($\beta 1$) d is unifiable with some lth argument of c, or

($\beta 2$) v and d must be identical at least on their highest syntactical level, so d can be obtained from v by replacing all the direct subobjects $v_1, \cdots, v_l$ of v by corresponding subobjects $d_1, \cdots, d_l$ of d ($l \geq 0$).

($\beta 3$) $b[n]$ is also variable, $b[n] \leftarrow \lambda v_1 \cdots v_r \cdot w \cdot \in \eta_i$ and v, w consists of a combination of subobjects of c and d.

In subcase ($\beta 1$) let $\xi = \{a[n] \leftarrow \lambda u_1 \cdots u_p \cdot u_l \cdot\}$ and $\eta = \eta_i - \xi$.

In subcase ($\beta 2$) the substitution $\{a[n] \leftarrow \lambda u_1 \cdots u_p \cdot v \cdot\}$ can be decomposed as follows: $\{a[n] \leftarrow \lambda u_1 \cdots u_p \cdot v \cdot\} = \{a[n] \leftarrow \lambda u_1 \cdots u_p \cdot w \cdot\} \circ \{f_1 \leftarrow \lambda u_1 \cdots u_p \cdot g_1 \cdot, \cdots, f_l \leftarrow \lambda u_1 \cdots u_p \cdot g_l \cdot\}$, where w is obtained from V by replacing $v_1, \cdots, v_l$ by $f_1 u_1 \cdots u_{p+k}, \cdots, f_l u_1 \cdots u_{p+k}$, where the f_i ($1 \leq i \leq l$) are "new" variables satisfying the requirements of the Unification Algorithm, and the g_i ($1 \leq i \leq l$) are appropriate objects. Let

$\xi = \{a[n] \leftarrow \lambda u_1 \cdots u_p \cdot w \cdot\}$ and
$\eta = (\eta_i - \{a[n] \leftarrow \lambda u_1 \cdots u_p \cdot v \cdot\}) \cup \{f_1 \leftarrow \lambda u_1 \cdots u_p \cdot g_1 \cdot\} \cup \{f_l \leftarrow \lambda u_1 \cdots u_p \cdot g_l \cdot\}$.

It is easy to notice that in both subcases eq. (4) is satisfied.

It is important to notice that ξ, as defined above, corresponds precisely to the substitution ξ specified in the Unification Algorithm (the cases (α), ($\beta 1$), ($\beta 2$), ($\beta 3$) correspond respectively to the elimination, projection, imitation, and combination of repetition with identification rules of the Unification Algorithm).

Hence from the assumption that (1) is valid for $k = i$ and from (3) it follows that (1) is also true for $k = i + 1$.

It remains to prove the validity of (2). In order to do that let us examine the following chain of equalities:

$$
\begin{aligned}
& \sigma_{i+1} \circ \eta - \text{new}(\eta) = \\
\text{I} \quad & (\sigma_i \circ \xi - \text{new}(\xi)) \circ \eta - \text{new}(\eta) = \\
\text{II} \quad & (\sigma_i \circ \xi \circ \eta - \text{new}(\xi \circ \eta)) - \text{new}(\eta) = \\
\text{III} \quad & (\sigma_i \circ (\eta_i \cup \text{new}(\eta)) - \text{new}(\xi \circ \eta)) - \text{new}(\eta) = \\
\text{IV} \quad & (\sigma_i \circ \eta_i - \text{new}(\xi \circ \eta)) - \text{new}(\eta) = \\
\text{V} \quad & (\sigma_i \circ \eta_i - \text{new}(\eta_i)) - \text{new}(\eta) = \\
\text{VI} \quad & \sigma_i \circ \eta_i - \text{new}(\eta_i) = \sigma.
\end{aligned}
$$

Each of the above equalities is a consequence of the one immediately above using the previous results as indicated: I follows from (3), II follows from the definition of composition of substitutions and properties of function *new*, III uses (4), IV follows for the same reasons as II, V depends on (4), and finally VI uses (3), (4), and the inductive assumption $k = i$ together with (2). These equalities imply that formula (2) holds for $k = i + 1$ and consequently $\eta = \eta_{i+1}$, which proves equality (2) for all k $(k \geq 0)$. This completes the first part of the proof.

The second part of this proof utilizes the results (1) and (2) to prove that there exists n $(n \geq 0)$ such that σ_n is a unifier of a_0, b_0.

Now we shall estimate the values of the complexity of the "reminder" substitution η_j $(j \geq 0)$, provided that $\bar{a}_0 \circ \sigma_j \neq \bar{b}_0 \circ \sigma_j$.

In case (α): if $\text{Comp}(\eta_j) = (n_1, \cdots, n_k, \cdots, n_m)$ where

$$k = \text{comp}(\lambda u_1 \cdots u_p \cdot w \cdot)$$

then $\text{Comp}(\eta_{j+1}) = (\eta_1, \cdots, n_k - 1, \cdots, n_m)$. This follows from the fact that $\eta_{i+1} = (\eta_i - \{a[m] \leftarrow \lambda u_1 \cdots u_p \cdot v \cdot\}) \cup \{f_1 \leftarrow \lambda u_1 \cdots u_{l-1} u_{l+1} \cdots u_p \cdot v \cdot\}$ and that $\text{comp}(\lambda u_1 \cdots u_p \cdot v \cdot) = \text{comp}(\lambda u_1 \cdots u_{l-1} u_{l+1} \cdots u_p \cdot v \cdot) + 1$.

In subcase $(\beta 1)$: if $\text{Comp}(\eta_j) = (n_1, \cdots, n_m)$ then since $\eta_{i+1} = \eta_i - \xi$ $\text{Comp}(\eta_{j+1}) = (n_1 - i_1, \cdots, n_m - i_m)$ where $(i_1, \cdots, i_m) = \text{Comp}(\xi) \neq 0$.

In subcase $(\beta 2)$: if $\text{Comp}(\eta_j) = (n_1, \cdots, n_k, \cdots, n_m)$ where

$$k = \text{comp}(\lambda u_1 \cdots u_p \cdot v \cdot)$$

then $\text{Comp}(\eta_{j+1}) = (m_1, \cdots, m_{k-1}, n_k - 1, n_{k+1}, \cdots, n_m)$. This follows from the fact that $\eta_{j+1} = (\eta_j - \{a[n] \leftarrow \lambda u_1 \cdots u_p \cdot v \cdot\}) \cup \bigcup_{i=1}^{l} \{f_i \leftarrow \lambda u_1 \cdots u_p \cdot g_i \cdot\}$ where g_i $(i = 1, \cdots, l)$ are objects whose complexities are clearly equal to or smaller than the corresponding complexities of the direct subobjects v_i $(i = 1, \cdots, l)$ of v. Consequently $\text{comp}(u_1 \cdots u_p \cdot v \cdot) \geq \text{comp}(u_1 \cdots u_p \cdot g_1 \cdot) + 1$ for all i $(1 \leq i \leq l)$.

Subcase $(\beta 3)$ can be treated in a similar way as $(\beta 2)$.

The results above imply that if $\bar{a}_0 \circ \sigma_j \neq \bar{b}_0 \circ \sigma_j$ then $\text{Comp}(\eta_j) > \text{Comp}(\eta_{j+1})$, where ">" is the well-order specified in Lemma 1 of Appendix I. Thus such a sequence $\eta_1, \cdots, \eta_j, \cdots$ must be finite. So there must exist k $(k \geq 0)$ such that $\bar{a}_0 \circ \sigma_k = \bar{b}_0 \circ \sigma_k$, which completes the second part of the proof.

The result above together with (1) implies that for an arbitrary pair of objects a_1, a_2 with a unifier σ there exists k $(k \geq 0)$ and substitutions σ_k, η_k such that $\sigma = \sigma_k \circ \eta_k$ and $\sigma_k \in S\langle \bar{a}_1, \bar{a}_2, \epsilon \rangle$. This completes the proof of our theorem for $m = 2$. The remainder of the proof proceeds by induction on m.

Let us assume that the theorem is valid for $m = i$ $(i \geq 2)$. Let σ be a unifier of

$\{\bar{a}_1, \cdots, \bar{a}_{i+1}\}$. Obviously σ is also a unifier of $\{\bar{a}_1, \cdots, \bar{a}_i\}$ and from our inductive assumption it follows that there exists a pair of substitutions ξ, η such that $\xi \in \Omega\{\bar{a}_1, \cdots, \bar{a}_i\}$ and $\sigma = \xi \circ \eta$. Clearly η is a unifier of $\bar{a}_1 \circ \xi, \bar{a}_{i+1} \circ \xi$; hence from the validity of the theorem for $m = 2$ follows the existence of substitutions ξ', η' such that $\xi' \in S\langle \bar{a}_1 \circ \xi, \bar{a}_{i+1} \circ \xi, \xi \rangle$ and $\eta = \xi' \circ \eta'$. Thus the definition of Ω gives $\xi \circ \xi' \in \Omega\{\bar{a}_1, \cdots, \bar{a}_{i+1}\}$. From $\sigma = \xi \circ \eta$ and $\eta = \xi' \circ \eta'$ follows $\sigma = (\xi\ \xi') \circ \eta'$. This completes the proof of Theorem 2.

It is important to notice that Theorem 2 is not valid for third-order objects. For example, let a, u, B, ϕ be characters, $\tau(a) = t$, $\tau(u) = \tau(B) = (t, t)$, and $\tau(\phi) = ((t, t), t)$ where u, ϕ are variables. Then a general unifier of the set $\{\phi B, Ba\}$ must contain the following substitutions: $\phi \leftarrow \lambda u \cdot Ba \cdot$ and $\phi \leftarrow \lambda u \cdot ua \cdot$. However, our unification procedure will provide only the first one.

3.12. In Figures 4–6 we present some examples of applications of the Unification Algorithm (for simplicity, the bars over object classes are omitted). The unified triples are circled.

Example 1. See Figure 4.

Example 2. See Figure 5.

Example 3. See Figure 6.

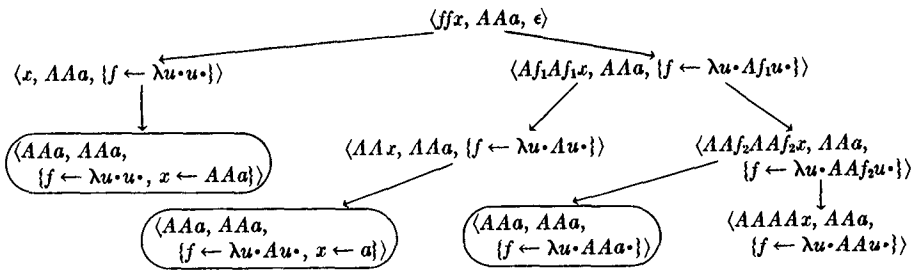


FIG. 4

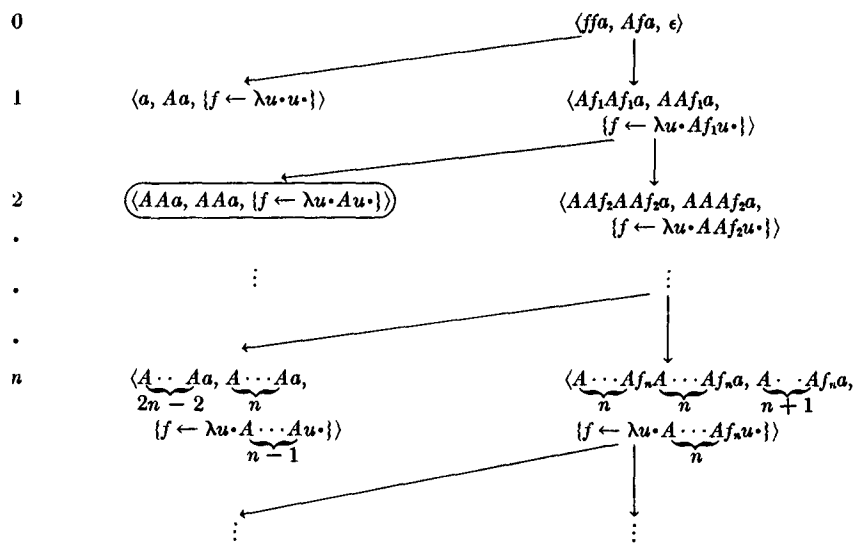


FIG. 5

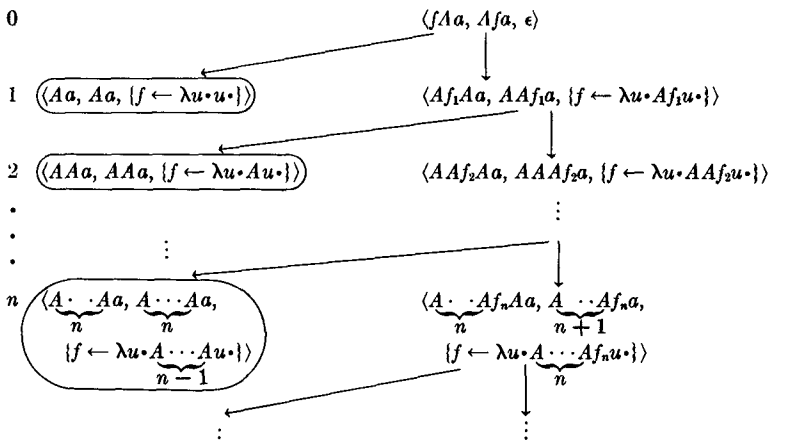


FIG. 6

3.13. In Example 1 the unification tree and, obviously, the most general unifier are finite. In Example 2 the unification tree is infinite but the most general unifier is finite (counts as a single substitution). In Example 3 both the unification tree and the most general unifier are infinite.

Generally speaking, the unification cases where the most general unifier is finite will be called *regular* (Examples 1 and 2). If in addition the unification tree is finite the case will be called *strongly regular* (Example 1). A unification in which there is no finite general unifier (Example 3) will be called *essentially singular*.

There is a very simple way to discover if a set Q is strongly regular, which is given in the following propositions.

PROPOSITION 1. *The unification tree for a pair of object classes of order not larger than 2 is strongly regular if there is never a case of the imitation rule being applicable to a pair of subobjects such that the variable g being replaced occurs free in the "imitated" object w.*

Obviously if the unification tree for $\langle \bar{a}, \bar{b}, \epsilon \rangle$ is regular then the general unifier of set $\{\bar{a}, \bar{b}\}$ is finite. This leads to the following generalization.

PROPOSITION 2. *If for all i,j* $(1 \leq i, j \leq m)$ *the unification trees for* $\{\langle a_i, a_j, \epsilon \rangle\}$ *are regular and* $\{\bar{a}_1, \cdots, \bar{a}_m\}$ *is a set of object classes of order not larger than 2, then* $\Omega\{\bar{a}_1, \cdots, \bar{a}_m\}$ *is finite.*

The proofs of these propositions are left to the reader.

4. *Systems*

In this section we discuss first the notion of a system, which serves as our basis for proving theorems by refutation. Next we develop a notion of unsatisfiability which corresponds to the usual intuitive notion, but mainly which will prove to be extremely useful when we consider generalized resolution in Section 5.

4.1. A finite set of literal classes is called a *clause*. Clauses will be denoted by capital Latin letters. A pair of clauses $X = \{l_1, \cdots, l_m\}$, $Y = \{p_1, \cdots, p_m\}$ $(m \geq 1)$ is called *complementary* iff $p_i = \neg\, l_i$ for $1 \leq i \leq m$. Such a pair of clauses will be denoted $X, \bar{X}$. An empty clause will be denoted $\square$.

A countable set of clauses is a *system*. Systems will be denoted by boldface capital Latin letters.

The above is obviously adopted from Robinson's first-order resolution terminology. According to it a clause is interpreted as the disjunction of literals, while a system is interpreted as the conjunction of clauses. It should be noted that the notion of "atom" is eliminated from this discussion, since we may use literal classes of great complexity. Although semantic intent might seem "submerged" by such a liberal definition avoiding atoms, it will prove very useful in higher order logic: preskolemization, much less "atomization," is not in general possible here.

We now give propositional definitions of model and inconsistency and note their property of "compactness."

A subset M of $\bigcup \boldsymbol{X}$ is called a *model* for a system $\boldsymbol{X}$ iff for each X, $X \in \boldsymbol{X}$ implies $X \cap M \neq \varnothing$ and M does not contain a pair of complementary literals.

A system $\boldsymbol{X}$ is *inconsistent* iff it has no model. Let $\boldsymbol{X}$ be a system. The set $\mathrm{GR}(\boldsymbol{X})$ called the *ground resolution* of $\boldsymbol{X}$ is defined below:

$$\mathrm{GR}(\boldsymbol{X}) =^{\mathrm{def}} \{Z \mid Z \in \boldsymbol{X} \text{ or exist } X, Y \in \boldsymbol{X} \text{ and } x \text{ such that } \bar{x} \in X, \neg\bar{x} \in Y, \text{ and } Z = (X - \{\bar{x}\}) \cup (Y - \{\neg\bar{x}\})\}.$$

Furthermore let $\mathrm{GR}^0(\boldsymbol{X}) =^{\mathrm{def}} \boldsymbol{X}$, and $\mathrm{GR}^{n+1}(\boldsymbol{X}) =^{\mathrm{def}} \mathrm{GR}(\mathrm{GR}^n(\boldsymbol{X}))$ for $n \geq 0$.

The following result is a generalization of Robinson's theorem from [14], for countable systems.

THEOREM 3. *A system $\boldsymbol{X}$ is inconsistent iff there exists n $(n \geq 0)$ such that* $\square \in \mathrm{GR}^n(\boldsymbol{X})$.

The theorem of course is familiar. It makes use of the (propositional) Compactness Theorem of logic [17] and is easily adapted to our needs by Robinson [14].

4.2. Clearly a system inconsistent by the propositional definition above is "intuitively inconsistent." But the reverse is not the case: By these propositional definitions the system composed of the clauses $\{\overline{\forall x P}\}$ and $\{\neg\bar{p}\}$ is formally not inconsistent. Yet we want to say that such a system is "unsatisfiable" in a natural sense. This will be somewhat complicated, but the obvious first step is to provide for the partial "elimination of quantifiers."

First we shall introduce some auxiliary notions. A substitution $\{v_1 \leftarrow u_1, \cdots, v_n \leftarrow u_n\}$ $(n \geq 1)$ is called a *variable change* iff $\{u_1 \leftarrow v_1, \cdots, u_n \leftarrow v_n\}$ is also a substitution. A pair of literal classes $\bar{p}$, $\bar{q}$ are called *variants* iff there exist variable changes σ, ξ such that $\bar{p} = \bar{q} \circ \sigma$, and $\bar{q} = \bar{p} \circ \xi$.

Now we shall introduce our version of combined rules of generalization and existentialization. For practical convenience the rule of double negation is also included.

Let p, q be literals. We shall say that $\bar{q}$ is a *simple Q-reduction* of p iff

$$(1) \quad \text{if } p = \begin{cases} \forall ur \\ \text{or} \\ \neg\exists ur \end{cases} \quad \text{then} \quad \bar{q} \text{ is a variant of } \begin{cases} \bar{r} \\ \text{or} \\ \neg\bar{r}, \end{cases}$$

which is obtained by replacement of all free occurrences of u in r by some arbitrary variable (possibly u) which is not free in r;

$$(2) \quad \text{if } p = \begin{cases} \exists ur \\ \text{or} \\ \neg\forall ur \end{cases} \quad \text{then} \quad \bar{q} = \begin{cases} \bar{r} \circ \{u \leftarrow fv_1 \cdots v_n\} \\ \text{or} \\ \neg\bar{r} \circ \{u \leftarrow gw_1 \cdots w_m\}, \end{cases}$$

where f or g is an existential parameter of the appropriate type arbitrary but uniquely assigned respectively to the set of variants of $\overline{\exists up}$ or $\overline{\neg\forall up}$; here $v_1, \cdots, v_n, w_1, \cdots, w_m$ $(n, m \geq 0)$ are all the distinct variables occurring free respectively in $\exists up$ and $\neg\forall up$;

(3) if $p = \neg\neg r$ then $q = r$;

(4) if otherwise then $q = p$.

The Q-rule could be given a shorter formulation where only upper subcases of (1), (2) were treated, and the other subcases together with (3) replaced by appropriate axioms. However, it would obviously make the proof longer, and does not make implementation of the Q-rule really easier.

We shall say that $\bar{q}$ is *Q-reduction* of $\bar{p}$ iff for some $\bar{r}$, $\bar{q}$ is a simple Q-reduction of $\bar{r}$ and $\bar{r}$ is a Q-reduction of $\bar{p}$.

We shall also say that a clause $X = \{\bar{x}_1, \cdots, \bar{x}_k\}$ is a Q-reduction of a clause $Y = \{\bar{y}_1, \cdots, \bar{y}_k\}$ iff $\bar{x}_i$ is a Q-reduction of $\bar{y}_i$.

4.3. Now we shall introduce a version of the instantiation rule which combines substitution with the elimination of some quantifiers.

A clause X is an *instance of a clause* Y *under the substitution* σ (or simply *instance*) iff for some subclause $Z \subseteq Y$, $X = ((Y - Z) \cup (Z \circ \sigma)_Q) \circ \sigma$, where W_Q denotes a Q-reduction of W.

The necessity of the double application of substitution in forming an instance is illustrated in the example at the end of this subsection.

Let $\boldsymbol{X}$ be a system. The *extension of* $\boldsymbol{X}$ is given by

$$E(\boldsymbol{X}) =^{\text{def}} \{Z \mid Z \text{ is an instance of some } X \in \boldsymbol{X}\}.$$

This is of course a system, and indeed a very rich system which consists exactly of all the instances of clauses of $\boldsymbol{X}$. Indeed, it provides the bridge from our propositional definition of inconsistency of Subsection 4.1 with the usual intuitive notion of unsatisfiability: A system is called *unsatisfiable* iff $E(\boldsymbol{X})$ is inconsistent. We give an example of how $E(\boldsymbol{X})$ provides an intuitive decomposition of the possibly complex literal classes appearing in $\boldsymbol{X}$. (In this example and elsewhere we will omit the bars which should indicate object classes.)

Example. Let $\boldsymbol{X}$ be the system:

(1) $\{(\forall x\, Qx \supset \forall x \exists y\, P^2xy)\}$,

(2) $\{Qu\}$,

(3) $\{\neg P^2av\}$,

(4) $\{\neg(p \supset q), \neg p, q\}$.

Now $\boldsymbol{X}$ itself is not inconsistent, but $E(\boldsymbol{X})$ has the following clauses as members (among many, many others):

(5) $\{Qa\}$,

(6) $\{\neg P^2aFa\}$,

(7) $\{\neg(\forall x\, Qx \supset \forall x \exists y\, P^2xy), Qa, P^2aFa\}$.

These arise from the substitutions $\{u \leftarrow a\}$, $\{v \leftarrow Fa\}$, $\{p \leftarrow \forall xQx, q \leftarrow \forall x\exists y\, P^2xy, x \leftarrow a\}$ and Q-reductions. Notice that clause (4) is essential to supply semantic intent to the symbol $\supset$.

4.4. We have given the notion of unsatisfiability in the above form as it will be more compatible with our discussion of generalized resolution in Section 5. However, our notion is exactly equivalent to the usual notion based on Henkin's theory of general models for type theory [8].

A system $\boldsymbol{X}$ is called *original* iff no existential parameter occurs in any literal of

any clause of X. X is *complete* iff whenever a propositional functor occurs in X then X contains its semantic description (all the three clauses from the list of axioms (1) to (9) in the beginning of Section 6 in which this functor occurs).

THEOREM 4. *If X is an original and complete system with extension $E(X)$ such that $E(X)$ has a model (see definition, Subsection 4.1), then X has a general Henkin model.*

The proof of this theorem was provided by D. C. Jensen [9]. The idea of the proof is as follows: A Henkin model H consists of a domain of elements for each type; the base universe is the domain of objects of type CLASS. The domain for objects of type LITERAL is {T, F}. The domain for each compound type (t_1, t_2) is a class of functions from the domain for t_1 to the domain for t_2. (This class is usually not all functions, but is closed under certain operations.) Then H is a model for a system X if each clause in X is valid under each assignment of variables to elements of the appropriate domains. Now a model M for $E(X)$ in the sense of Subsection 4.1 amounts only to an assignment of the truth value T to certain literals. However, we can construct a Henkin model H for X using this M. The proof is involved but proceeds along the following lines: First it is noted that $E(X)$ is so rich that M already includes a complete atomic description for a suitable base universe for H. This is the case because $E(X)$ forces a "decomposition" of complex formulas as illustrated in the example above. Next proper domains for compound type symbols are produced. Finally, it is shown that each clause in X is valid in the Henkin sense in the H which was produced. Our notion of unsatisfiability may be compared with others in the literature (see [1, 8, 15]).

The restriction that the system must be original is introduced to safeguard against the possibility of inserting existential parameters which already occur in the system. It expresses the conditions under which the rule-C (or existential instantiation) can be legally applied in predicate calculus.

5. *Resolution*

In this section we shall synthesize the earlier results and formulate the method o detecting the unsatisfiability of a system. This method can be interpreted as a generalized resolution principle. The resolution will be the only inference rule for our systems which combines the properties of cut rule, simple substitution, λ-normalization, and quantifier elimination. In a way it satisfies the conditions stated by Andrews in [1, p. 38].

Let X, Y be a pair of clauses. A clause Z is called a *resolvent* of X, Y iff $Z = ((X - M) \cup (Y' - \tilde{N})) \circ \sigma$, where Y' is an arbitrary but fixed variant Y such that no variable occurs free in X and Y' simultaneously, $M \subseteq X$, $\tilde{N} \subseteq Y'$, and $\sigma \in \Omega(P)$ where P is such Q-reduction of $M \cup N$ that no variable introduced in $M \cup N$ occurs free in X of Y'.

Let X be a system. The set $R(X)$, called the *resolution* of X, is defined as follows:

$$R(X) =^{\text{def}} X \cup \{Z \mid Z \text{ is a resolvent of some } X, Y \in X\}.$$

Furthermore we define $R^0(X) = X$ and for all n ($n \geq 0$), $R^{n+1}(X) = R(R^n(X))$. Now we may state the final result of this paper. We shall say that a system X is of *order k* iff the set of literals of which it consists is of order k.

THEOREM 5. *Let X be an original system of order not larger than 2. X is unsatisfiable iff there exists n such that $\square \in R^n(X)$.*

PROOF. First we shall prove the "if" part. To simplify this part of the proof we introduce the notion of *liberalized ground resolution* as follows:

$$^*\mathrm{GR}(X) = ^{\mathrm{def}} \{Z \mid Z \in X \text{ or there exist } X, Y \in X \text{ and a literal } x \text{ such that } Z = (X - \{\bar{x}\}) \cup (Y - \{\neg\bar{x}\})\}.$$

We note that $^*\mathrm{GR}(X)$ is larger than $\mathrm{GR}(X)$ but in fact they are essentially the same from the standpoint of refutation (i.e. $\square \in {}^*\mathrm{GR}(X)$ iff $\square \in \mathrm{GR}(X)$). Define $^*\mathrm{GR}^n(X)$ analogous to $\mathrm{GR}^n(X)$.

Now we shall establish the following implication: for all $n \geq 0$,

(1) $X \in E(R^n(X))$ implies that there exists Y, $Y \in {}^*\mathrm{GR}^n(E(X))$ and $Y \subseteq X$.

For $n = 0$ the above is obviously true.

Let us assume inductively that the assertion is valid for $n = i$. If $X \in E(R^{i+1}(X))$ then according to the definition of the operator E there is a clause $Z \in R^{i+1}(X)$ and a substitution σ such that X is an instance of Z under σ.

But this means that either

(a) $Z \in R^i(X)$, or

(b) $Z = ((U - M) \cup (V' - \tilde{N})) \circ \xi$ where $U, V \in R^i(X)$ and M, N, ξ satisfy the conditions stated in the definition of the resolvent.

In case (a) the proof of the assertion is obvious.

In case (b), from the definition of the resolvent it follows that there are Q-reductions M_Q, N_Q of M, N such that $M_Q \circ \xi = N_Q \circ \xi$.

It is an easy exercise to check that there will also exist Q-reductions M_1 and N_1 of $M \circ (\xi \circ \sigma)$ and $N \circ (\xi \circ \sigma)$ such that $M_1 \circ (\xi \circ \sigma) = N_1 \circ (\xi \circ \sigma) = \{\tilde{Z}\}$. (That is, the first application of $\xi \circ \sigma$ is harmless as far as unification is concerned.)

Now $(U - M) \circ \xi$ and $(V' - \tilde{N}) \circ \xi$ are subsets of Z, so let U_1 and V_1 be the corresponding subsets of X (which is an instance of Z). Using the singleton $\{\tilde{Z}\}$ obtained above define $U_2 = U_1 \cup \{\tilde{Z}\}$ and $V_2 = V_1 \cup \{\neg\tilde{Z}\}$. It is easy to check that U_2 and V_2 are instances of U and V' under $\xi \circ \sigma$. Thus $U_2, V_2 \in E(R^i(X))$.

From the inductive assumption it follows that there exist U_3, V_3 such that $U_3 \subseteq U_2$, $V_3 \subseteq V_2$ and $U_3, V_3 \in {}^*\mathrm{GR}^i(E(X))$. Setting

$$Y = (U_3 - \{\tilde{Z}\}) \cup (V_3 - \{\neg\tilde{Z}\}),$$

if follows that

$$Y \subseteq (U_2 - \{\tilde{Z}\}) \cup (V_2 - \{\neg\tilde{Z}\}) = U_1 \cup V_1 = X,$$

and, of course,

$$Y \in {}^*\mathrm{GR}^{i+1}(E(X)).$$

This proves implication (1).

Now let $\square \in R^n(X)$. Then obviously $\square \in E(R^n(X))$ and from (1) it follows that there exists Y, $Y \in {}^*\mathrm{GR}^n(E(X))$ and $Y \subseteq \square$. That clearly proves that $Y = \square$ and in view of Theorem 3 $E(X)$ is inconsistent. It proves that X is unsatisfiable and completes the "if" part of the proof.

In order to prove the "only if" part we shall show that

(2) $\mathrm{GR}(E(X)) \subseteq E(R(X))$.

Now let us assume that $Y \in \mathrm{GR}(E(X))$. This means that

(a) $Y \in E(X)$, or

(b) there exists a literal z and clauses U_1, V_1 such that U_1, $V_1 \in E(\boldsymbol{X})$, $\bar{z} \in U_1$, $\neg\bar{z} \in V_1$, and $Y = (U_1 - \{\bar{z}\}) \cup (V_1 - \{\neg\bar{z}\})$.

In case (a) the proof is obvious.

In case (b) from the definition of operator E it follows that there exist clauses U, $V \in \boldsymbol{X}$, and substitutions ξ, η such that $\mathrm{IN}(U_1, U, \xi)$ and $\mathrm{IN}(V_1, V, \eta)$. ($\mathrm{IN}(Y, Z, \sigma)$ should be read "Y is an instance of Z under σ.") Without losing generality we may also assume that U, V, ξ, η are chosen so that $V = V_U{}'$ and if $\xi = \{u_1 \leftarrow x_1, \cdots, u_m \leftarrow x_m\}$, $\eta = \{v_1 \leftarrow y_1, \cdots, v_n \leftarrow y_n\}$ then $u_1, \cdots, u_m$, $v_1, \cdots, v_n$ are all distinct, no v_i $(1 \leq i \leq n)$ occurs in U and no u_i $(1 \leq i \leq m)$ occurs in V. This assumption is justified by the fact that there are either infinitely many, or none, distinct variables of any type.

Let $\sigma = \{u_1 \leftarrow x_1, \cdots, u_m \leftarrow x_m, v_1 \leftarrow y_1, \cdots, v_n \leftarrow y_n\}$. Then obviously $\mathrm{IN}(U_1, U, \sigma)$ and $\mathrm{IN}(V_1, V, \sigma)$. It is easy to prove that there exist maximal subsets M, $\tilde{N}$ of U, V such that $\{\bar{Z}\}$ is an instance of both M and N under σ. Then from Theorem 2 it follows that there exist substitutions ζ and θ such that $\sigma = \zeta \circ \theta$ and $\zeta \in \Omega(P)$. In view of the results above we have $\mathrm{IN}(Y, (U - M) \cup (V - \tilde{N}), \zeta \circ \xi)$. So clearly there exists a Z such that $\mathrm{IN}(Y, Z, \xi)$ and $\mathrm{IN}(Z, (U - M) \cup (V - \tilde{N}), \zeta)$. Since $V = V'$ it becomes obvious that Z is a resolvent of U, $V \in \boldsymbol{X}$ so $Z \in R(\boldsymbol{X})$ and consequently $Y \in E(R(\boldsymbol{X}))$. This proves (2) and by induction on n we can easily establish that

(3) $\mathrm{GR}^n(E(\boldsymbol{X})) \subseteq E(R^n(\boldsymbol{X}))$ for $n \geq 0$.

From the definition of unsatisfiability it follows that $E(\boldsymbol{X})$ is inconsistent, and by Theorem 3 there exists n such that $\square \in \mathrm{GR}^n(E(\boldsymbol{X}))$. In view of (3), if $\square \in E(R^n(\boldsymbol{X}))$ that means that there exists $X \in R^n(\boldsymbol{X})$ such that $\mathrm{IN}(\square, X, \sigma)$. However, this implies that $X = \square$, and completes the entire proof.

5.2. Theorem 5 is a generalization of the original Robinson result for first-order logic. However there is an essential difference: Sets $R^n(\boldsymbol{X})$ may become infinite here. This fact forces us to define some strategy of selecting finite subsets of $R^n(\boldsymbol{X})$ without losing the completeness. It can be easily achieved in many ways; one of these methods is presented below.

Let $\bar{a}$, $\bar{b}$ be object classes and σ a substitution. We shall define a subset of $S\langle a, b, \sigma\rangle$ as follows:

$$S^{(k)}\langle a, b, \sigma\rangle = {}^{\mathrm{def}} \{\eta \mid \eta \text{ is a substitution designed in some unified triple of } T\langle \bar{a}, \bar{b}, \sigma\rangle \text{ before the level } k + 1\}$$

for $k \geq 0$.

Now let $\Omega^{(k)}\{\bar{a}_1, \bar{a}_2\} = {}^{\mathrm{def}} S^{(k)}\langle a, b, \sigma\rangle$ and

$$\Omega^{(k)}\{\bar{a}_1, \cdots, \bar{a}_i, \bar{a}_{i+1}\} = {}^{\mathrm{def}} \bigcup \{S^{(k)}\langle \bar{a}_1 \circ \sigma, \bar{a}_{i+1} \circ \sigma, \sigma\rangle \mid \sigma \in \Omega^k\{\bar{a}_1, \cdots, \bar{a}_i\}\}$$

for all $i \geq 2$.

Obviously $\lim_{k=\infty} S^{(k)}\langle a, b, \sigma\rangle = S\langle a, b, \sigma\rangle$ and $\lim_{k=\infty} \Omega^{(k)}\{\bar{a}_1, \cdots, \bar{a}_i\} = \Omega\{\bar{a}_1, \cdots, \bar{a}_i\}$ for $(i \geq 2)$.

In an analogous way if $\boldsymbol{X}$ is a system let $R_{(k)}(\boldsymbol{X})$ be a system defined as follows: $Z \in R_{(k)}(\boldsymbol{X})$ iff there exists a quadruple of clauses X, Y, M, N and substitution σ such that X, $Y \in \boldsymbol{X}$, $M \subseteq X$, $\tilde{N} \subseteq Y'$, and $Z = (X - M) \otimes \sigma \cup (Y' - \tilde{N}) \otimes \sigma$ where $\sigma \in \Omega_{(k)}(P)$ where P is as in the definition of resolution. We also define:

$$R_k^{(0)}(\boldsymbol{X}) = \boldsymbol{X} \text{ and } R_{(k)}^{n+1}(\boldsymbol{X}) = R_{(k)}(R_{(k)}^n(\boldsymbol{X})).$$

From this definition it immediately follows that for each X and all n, k $(n, k \geq 0)$, $R^n_{(k)}(X)$ is finite and $\lim_{k\to\infty} R^n_{(k)}(X) = R^n(X)$.

Finally, Theorem 5 can be generalized as follows:

THEOREM 6. *Let X be an original system of the order not larger than 2. X is unsatisfiable iff there exist k, n such that $\square \in R^n_{(k)}(X)$, and $R^n_{(k)}(X)$ is finite.*

The proof of this theorem is obvious.

However, it should be mentioned that the above method of finitizing the resolution sets is far from being practical, and the problem of dealing with infinite resolution sets requires separate consideration.

6. *Application of the Method*

This section is devoted to applications of our method to various proofs.

To economize on notation we shall assume that a clause $\{\bar{p}_1, \bar{p}_2, \cdots, \bar{p}_n\}$ $(n \geq 1)$ will be simply denoted as $p_1, p_2, \cdots, p_n$.

First we shall state the axioms of the propositional calculus, which will be used in later proofs. (A similar approach has been used by Bledsoe in [2].)

(1) $\neg(p \supset q), \neg p, q,$
(2) $(p \supset q), p,$
(3) $(p \supset q), \neg q,$
(4) $\neg(p \wedge q), p,$
(5) $\neg(p \wedge q), q,$
(6) $(p \wedge q), \neg p, \neg q,$
(7) $\neg(p \vee q), p, q,$
(8) $(p \vee q), \neg p,$
(9) $(p \vee q), \neg q.$

6.1. The following two examples from number theory are chosen to emphasize the advantages of not destroying the original form of the theorems. It is obviously possible in our system to realize the idea of "marco predicates" as postulated by Meltzer in [10].

Example 1. First we shall prove unsatisfiability of the system consisting of clauses (1) to (9) and the following:

(10) $x = x$ (property of equality),
(11) $\neg PO, \neg PSa, Px$ (part of the axiom of induction),
(12) $\neg \forall x(x = 0 \vee \exists y\, x = Sy)$ (negated theorem).

Proof.

(13) $\neg(b = 0 \vee \exists y\, b = Sy)$ from (12) and Q-rule,
(14) $\neg(0 = 0 \vee \exists y\, 0 = Sy), \neg(Sa = 0 \vee \exists y\, Sa = Sy)$ from (11), (13) $P \leftarrow \lambda u \cdot (u = 0 \vee \exists y\, u = Sy)\cdot$,
(15) $\neg 0 = 0, \neg(Sa = 0 \vee \exists y\, Sa = Sy)$ from (8), (14),
(16) $\neg(Sa = 0 \vee \exists y\, Sa = Sy)$ from (10), (15),
(17) $\neg Sa = Sy$ from (9), (16), and Q-rule,
(18) $\square$ from (10), (17).

Example 2. Now we shall prove another number theoretical theorem starting from its original form.

(10)	$\neg Sx = 0$	(axioms of number theory),
(11)	$(x + Sy) = S(x + y)$	
(12)	$(x + 0) = x$	
(13)	$\neg PO, \neg PSa, Px$	(part of the induction axiom),
(14)	$\neg x = y, \neg x = z, y = z$	(property of equality),
(15)	$\neg\forall x\,\forall y((y + x) = 0 \supset y = 0)$	(negated theorem).

Proof.

(16)	$\neg\forall y((y + b) = 0 \supset y = 0)$	from (15) and Q-rule,
(17)	$\neg((d + 0) = 0 \supset d = 0), \neg((c + Sa) = 0 \supset c = 0)$	from (13), (16), Q-rule, and $P \leftarrow \lambda u \cdot \forall y((y + u) = 0 \supset y = 0)\cdot$,
(18)	$\neg((d + 0) = 0 \supset d = 0), (c + Sa) = 0$	from (2), (17),
(19)	$(d + 0) = 0, (c + Sa) = 0$	from (2), (18),
(20)	$\neg d = 0, (c + Sa) = 0$	from (3), (18),
(21)	$\neg(d + 0) = y, y = 0, (c + Sa) = 0$	from (14), (19),
(22)	$\neg(d + 0) = d, (c + Sa) = 0$	from (20), (21),
(23)	$(c + Sa) = 0$	from (12), (22),
(24)	$\neg(c + Sa) = y, y = 0$	from (14), (23),
(25)	$S(c + a) = 0$	from (11), (24),
(26)	$\square$	from (10), (25).

It is clear that if proofs of the above theorems were made after completing skolemization and atomization they would be much longer. Moreover, there is one more advantage: If we keep the theorems in their "natural form" they seem to be more suitable to attack by heuristic strategies derived from some "natural" proof techniques. For example, the choice of proper substitution for the predicate P (line (14) in Example 1 and line (17) in Example 2) can be justified by some simple heuristic reasons.

6.2. In the following we shall present proofs of the three theorems from set theory.

Example 3.[1] The following presents a proof of a part of De Morgan's law. It should be noted that it is impossible to preskolemize this system.

(10)	$\neg\forall u x \in fu, x \in \bigcap ufu$	(definition of $\bigcap$),
(11)	$\neg\exists u x \in fu, x \in \bigcup ufu$	(definition of $\bigcup$),
(12)	$\neg x \in Cy, \neg x \in y$	(definition of C),
(13)	$x \in Cy, x \in y$	
(14)	$\neg\forall x\forall f(x \in C \bigcup ufu \supset x \in \bigcap uCfu)$	(negated theorem).

Proof.

(15)	$\neg(a \in C\bigcup uFu \supset a \in \bigcap uCFu)$	from (14) and Q-rule,
(16)	$a \in C\bigcup uFu$	from (2), (15),

[1] Intuitively, interpret $\bigcap ufu$ as $\bigcap_i f_i$ for intersection indexed by i, and similarly by $\bigcup ufu$.

(17) $\neg a \in \bigcup u F u$ from (12), (16),
(18) $\neg a \in Fu$ from (11), (17), and Q-rule,
(19) $\neg a \in \bigcap u C F u$ from (3), (15),
(20) $\neg a \in CFb$ from (10), (19), and Q-rule,
(21) $a \in Fb$ from (13), (20),
(22) $\square$ from (18), (21).

Example 4. This is another simple example of a theorem which cannot be preskolemized:

(10) $x \in \{u \mid Pu\}, \neg Px$
(11) $\neg x \in \{u \mid Pu\}, Px$ } (axiom of class formation),
(12) $\neg \forall z(z \in \{u \mid \exists v P^2 uv\} \supset \exists v\, z \in \{u \mid P^2 uv\})$ (negated theorem).

Proof.

(13) $\neg(a \in \{u \mid \exists v P^2 uv\} \supset \exists v\, a \in \{u \mid P^2 uv\})$ from (12) and Q-rule,
(14) $a \in \{u \mid \exists v P^2 uv$ from (2), (13),
(15) $P^2 ab$ from (11), (14), Q-rule and $P \leftarrow \lambda u \cdot \exists v P^2 uv \cdot$,
(16) $\neg a \in \{u \mid P^2 uv\}$ from (3), (13), and Q-rule,
(17) $\neg P^2 av$ from (10), (16), and $P \leftarrow \lambda u \cdot P^2 uv \cdot$,
(18) $\square$ from (15), (17).

Example 5. Here we shall prove half of Cantor's theorem (see [11, p. 56]).

(10) $x \in \{u \mid Pu\}, \neg Px$
(11) $\neg x \in \{u \mid Pu\}, Px$ } (axioms of class formation),
(12) $\neg x = y, \neg Px, Py$ (axiom of equality),
(13) $d \in x, x \subseteq y$
(14) $\neg d \in y, x \subseteq y$ } (definition of "$\subseteq$" relation[2]),
(15) $\exists f \forall x(x \subseteq a \supset \exists y(y \in a \wedge fy = x))$ (negated theorem).

Proof.

(16) $\forall x(x \subseteq a \supset \exists y(y \in a \wedge fy = y))$ from (15) and Q-rule,
(17) $(x \subseteq a \supset \exists y(y \in a \wedge Fy = x))$ from (16) and Q-rule,
(18) $\neg x \subseteq a, Gx \in a \wedge FGx = x$ from (1), (17),
(19) $\neg x \subseteq a, Gx \in a$ from (4), (18),
(20) $\neg x \subseteq a, FGx = x$ from (5), (18),
(21) $\neg Rz, \neg Qz, z \in \{u \mid (Ru \wedge Qu)\}$ from (6), (10), and $P \leftarrow \lambda u \cdot (Ru \wedge Qu) \cdot$,
(22) $Rz, \neg z \in \{u \mid (Ru \wedge Qu)\}$ from (4), (11), and $P \leftarrow \lambda u \cdot (Ru \wedge Qu) \cdot$,
(23) $Qz, \neg z \in \{u \mid (Ru \wedge Qu)\}$ from (5), (11), and $P \leftarrow \lambda u \cdot (Ru \wedge Qu) \cdot$,

[2] Actually, the parameter d should be given as a binary function (of x and y), but this simplified notation clearly follows as a special case.

(24)	$\neg x \subseteq a,\ \neg PFGx,\ Px$	from (12), (20),
(25)	$\neg\{u \mid (\neg u \in Fu \wedge Qu)\} \subseteq a,\ \neg QG\{u \mid (\neg u \in Fu \wedge Qu)\},\ G\{u \mid (\neg u \in Fu \wedge Qu)\} \in \{u \mid (\neg u \in Fu \wedge Qu)\}$	from (21), (24), $R \leftarrow \lambda u \cdot \neg u \in Fu \cdot$, $P \leftarrow \lambda u \cdot Gx \in u \cdot$, $z \leftarrow G\{u \mid (\neg u \in Fu \wedge Qu)\}$, $x \leftarrow \{u \mid (\neg u \in Fu \wedge Qu)\}$
(26)	$\neg\{u \mid (\neg u \in Fu \wedge Qu)\} \subseteq a,\ \neg QG\{u \mid (\neg u \in Fu \wedge Qu)\}$	from (22), (25), $R \leftarrow \lambda u \cdot \neg u \in Fu \cdot$, $z \leftarrow G\{u \mid (\neg u \in Fu \wedge Qu)\}$,
(27)	$\neg\{u \mid (\neg u \in Fu \wedge u \in a)\} \subseteq a$	from (19), (26), and $Q \leftarrow \lambda u \cdot u \in a \cdot$, $x \leftarrow \{u \mid (\neg u \in Fu \wedge u \in a)\}$
(28)	$d \in \{u \mid (\neg u \in Fu \wedge u \in a)\}$	from (13), (27),
(29)	$(\neg d \in Fd \wedge d \in a)$	from (11), (28), and $P \leftarrow \lambda u \cdot (\neg u \in Fu \wedge u \in a) \cdot$,
(30)	$d \in a$	from (5), (29),
(31)	$\neg d \in a$	from (14), (27),
(32)	$\square$	from (30), (31).

Some comments about this proof. In lines (16) to (20) an atomization of the original theorem is performed. Lines (21) and (22) are characteristic of many proofs where there is a necessity to introduce a predicate which is not an atomic formula (in this case it is $\lambda u \cdot (\neg u \in Fu \wedge u \in a) \cdot$). The derivation of line (25) is based on an interesting example of double second-order unification to obtain a merge. It should be noted that the above proof would be impossible to realize without use of some axioms of propositional calculus (axioms of conjunction (4) and (5)).

7. *Conclusion*

The method described above is clearly a proper generalization of the resolution principle for second-order logic. However, its application raises many problems which do not exist in the first-order case. We shall present some of them.

The main problem is due to the fact that growth of the resolution sets in the presence of second-order axioms is much faster. It focuses our attention on special strategies to accelerate the search. Most of the first-order general strategies can be directly applied (like set of support, merging, ancestry filters, and unit preference), but some, such as P_1-deduction, need modifications. However, it seems to be imperative to develop problem oriented strategies which will be suited to particular mathematical theories. The common feature of these strategies should be a possibility of dealing with much more complex atomic formulas than in first order.

Special consideration must be given to the fact that the unification of second-order literals generally involves many unifiers. This implies introducing some schemas of preference ordering of unifiers while producing resolvents. It becomes especially important when a general unifier is infinite.

The last group of problems is connected with the computer implementation of this method. Since the structure of the objects and the manipulation of them are more complex, there is clearly a need for some special data structure and manipulation rules which will suit the specific features of the higher order systems.

Appendix I

THEOREM 1. *For each object class $\bar{x}$ there corresponds a unique object class $\bar{y}$ such that y is a normal object and conv*(x, y) holds.*

PROOF. Proof of this theorem is partially provided in [1, pp. 8–13] (the existence of a normal object y) and the rest can be obtained from the works of Church and Rosser. However, we shall give here an independent proof which seems to be simpler. The proof is based on the following lemmas.

Let N^* be the set of all finite tuples of natural numbers, ordered by the relation "$>$" as follows: $\langle x_1, \cdots, x_n\rangle > \langle y_1, \cdots, y_m\rangle$ iff $n > m$ or $n = m$ and there exists k such that for all i $(0 \leq i \leq k)$, $x_{n-i} = y_{m-i}$ and $x_{n-k} > y_{m-k}$.

LEMMA 1. *If $\{X_i\}$ $(i = 1, 2, \cdots)$ is a sequence of elements of N^* such that $X_i > X_{i+1}$, then this sequence is finite.*

A proof of this lemma can be easily obtained by application of a double induction: on the length of X_i and on the value of its rightmost component.

We shall introduce some functions characterizing the type complexness of objects.

Let *lord* be a mapping of the set of λ-objects into the set of integers described as follows:

$$\text{lord}(\lambda u_1 \cdots u_n \cdot x \cdot w_1 \cdots w_m) =^{\text{def}} \max_{1 \leq i \leq m} \{\text{ord}\ (\tau(u_i))\} + 1$$

(where $1 \leq m \leq n$). And finally *Lord* is a mapping of $\mathbf{L}$ into N^* given by the formula

$$\text{Lord}(x) =^{\text{def}} (i_1, \cdots, i_k),$$

where x is an object and i_j = number of λ-subobjects y of x such that $\text{lord}(y) = j$.

A λ-object which has no proper λ-subobjects is called *minimal*.

LEMMA 2. *Let $\lambda u_1 \cdots u_n \cdot x \cdot w_1 \cdots w_m$ $(n \geq m \geq 1)$ be a minimal λ-object and let $\bar{y} = |\overline{\lambda u_1 \cdots u_n \cdot x \cdot w_1 \cdots w_m}|_\lambda$. If z is a λ-subobject of y then lord$(y) >$ lord(z).*

PROOF. Let $z = \lambda v_1 \cdots v_k \cdot s \cdot q_1 \cdots q_l$ $(1 \leq l \leq k)$. From the assumption it follows that all $x, w_1, \cdots, w_m$ have no λ-subobjects, and this means that there exist such i_0 that $w_{i_0} = \lambda v_1 \cdots v_k \cdot s \cdot$. Now from the definitions of functions *ord* and *lord*, and the fact that $\tau(u_i) = \tau(w_i)$ $(1 \leq i \leq n)$ combined with the above, we obtain the following:

$$\begin{aligned}
\text{lord(y)} &= \max_{1 \leq i \leq m} \{\text{ord}(\tau(u_i))\} + 1 \\
&= \max_{1 \leq i \leq m} \{\text{ord}(\tau(w_i))\} + 1 \geq \text{ord}(\tau(w_{i_0})) + 1 \\
&= \text{ord}((\tau(v_1), \cdots, \tau(v_k), \tau(s))) + 1 \\
&= \max_{1 \leq i \leq k} \{\text{ord}(\tau(v_i))\} + 2 \geq \max_{1 \leq i \leq l} \{\text{ord}(\tau(v_i))\} + 2 \\
&= \text{lord}(z) + 1 > \text{lord}(z).
\end{aligned}$$

This proves Lemma 2.

Now we shall proceed with the main part of the proof.

Let x, y be objects such that *conv*(x, y) holds and let *contr*(x, y) denote the λ-subobject of x which is replaced in y by its λ-contraction. Let us additionally assume that contr(x, y) is minimal. Let $\text{Lord}(x) = (i_1, \cdots, i_k, \cdots, i_m)$ and lord(contr $(x, y)) = k$ where $1 \leq k \leq m$. This means that y has one λ-subobject of lord $= k$ less and some new λ-subobjects. However, all these new λ-subobjects obviously

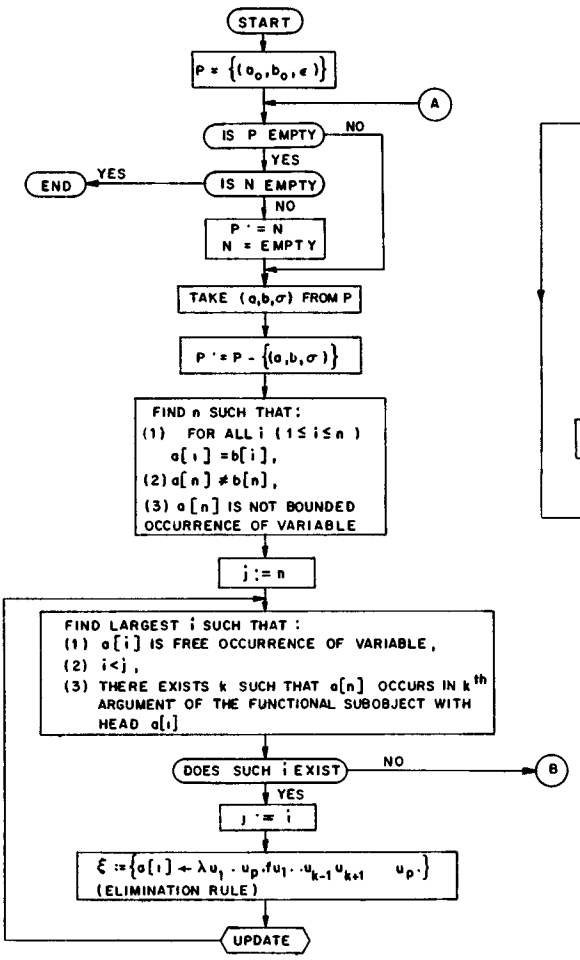
occur in contr(x, y) and by Lemma 2 their lord is smaller than k. These two results together with the definition of mapping Lord imply that

$$\text{Lord}(y) = (j_1, \cdots, j_{k-1}, i_k - 1, i_{k+1}, \cdots, i_m).$$

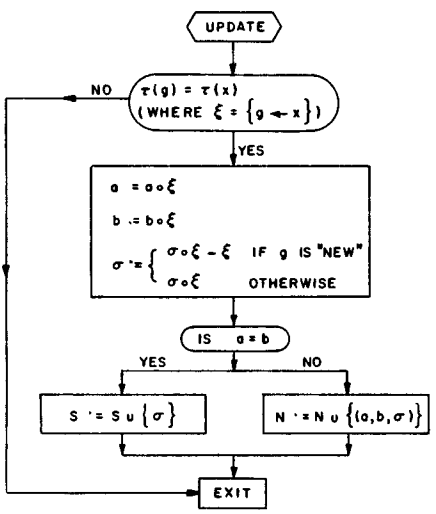
This means that Lord$(x) \succ$ Lord(y).

Now let $a_1, a_2, \cdots$, be a sequence of objects such that conv(a_i, a_{i+1}) $(i \geq 1)$ holds and contr(a_i, a_{i+1}) is minimal. It is easily noticed that if a_i is not a normal object then it has at least one minimal λ-subobject. Thus there also exists an object a_{i+1} such that conv(a_i, a_{i+1}). So, from the formula above it follows that Lord$(a_i) \succ$ Lord(a_{i+1}) for all $i \geq 1$. But Lemma implies that the sequence Lord(a_i) $(i = 1, 2, \cdots)$ must be finite. Thus there exists an n such that a_n is a normal object and obviously conv$^*(a_1, a_n)$ holds. Since a_1 is an arbitrary object it remains only to prove that the object class $\bar{a}_n$ uniquely corresponds to $\bar{a}_1$.

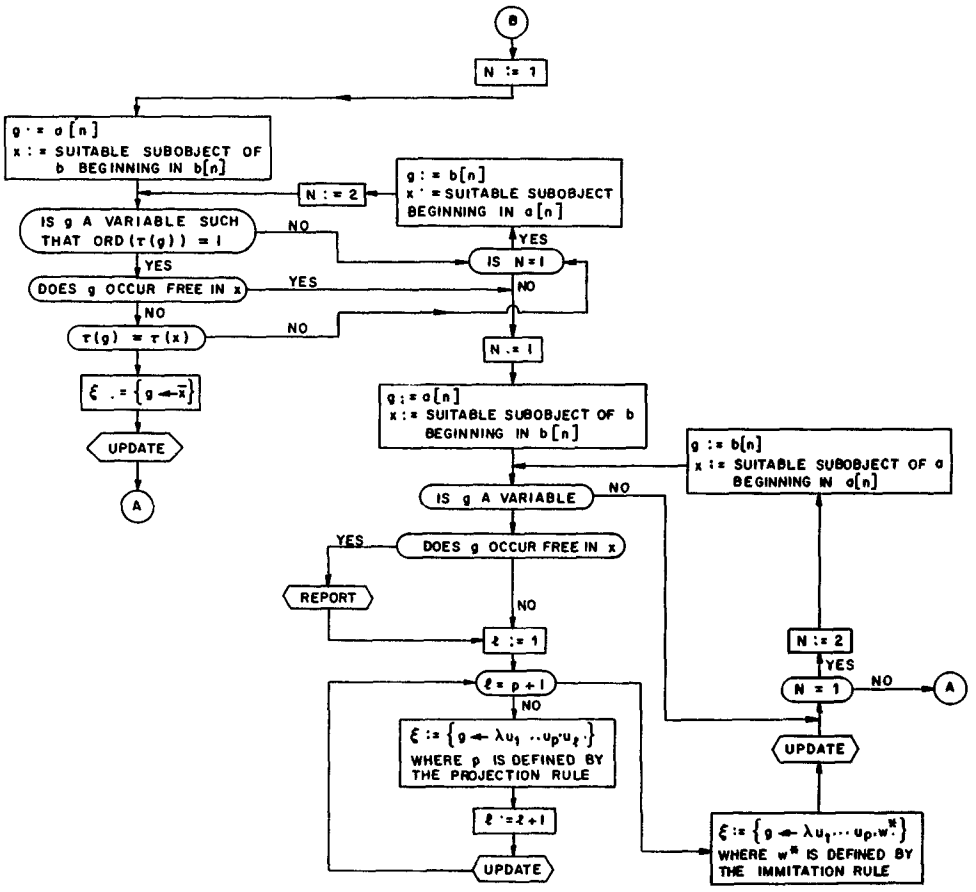
Let us assume inversely that there exists an object b such that b is normal, conv$^*(a_1, b)$ holds, and $\bar{b} \neq \bar{a}_n$. Let k be a number such that conv$^*(a_k, b)$ holds and conv$^*(a_{k+1}, b)$ is false. Obviously such a k always exists and $1 \leq k \leq n - 1$. Since conv$^*(a_k, b)$ holds, there exists a sequence $b_1, \cdots, b_m$ $(m \geq 2)$ such that



Flowchart: part (a)



Flowchart: part (b)



Flowchart: part (c)

$\mathrm{conv}(b_i, b_{i+1})$ holds for each i $(1 \leq i \leq m-1)$, $b_1 = a_k$ and $b_m = b$. Now, let $c = \mathrm{contr}(a_k, a_{k+1})$ and c_i be the subobject of b_i $(1 \leq i \leq p < m)$ which is the image of c under the replacement induced by the conversion from b_i to b_{i+1}. There must exist certain p $(1 \leq p < m)$ such that $c_p = \mathrm{contr}(b_p, b_{p+1})$ or c_p disappears from b_{p+1} (otherwise b_m would have a λ-subobject which contradicts with the assumption that b_m is a normal object). Now we shall define the sequence $b_1', \cdots, b_m'$ as follows:

$$b_i' = \begin{cases} \text{result of replacing } c_i \text{ in } b_i \text{ by an element of } |\bar{c}_i|_\lambda & (1 \leq i \leq p), \\ b_{i+1} & (p < i < m). \end{cases}$$

It is easily noticed that $\mathrm{conv}(b_i', b_{i+1}')$ holds for all i $(1 \leq i \leq m)$ and that $\bar{b}_{m-1}' = \bar{b}_m$. Moreover it can also be shown that $\mathrm{conv}(a_{k+1}, b_2')$. These two results imply that $\mathrm{conv}^*(a_{k+1}, b_m)$ holds, which contradicts the definition of k, and completes the proof of this theorem.

Appendix II

A flowchart of the Unification Algorithm is presented in three parts. The notation used here is as close as possible to the one used in the description of the algorithm;

however some additional conventions are necessary:
P denotes the recently completed level of the unification tree,
N denotes the level which is under construction,
$\acute{S}$ denotes the set of unifiers recently produced.
The box indicates the possibility of a message to the user that a singularity is detected (it may cause an infinite looping!).

If the algorithm terminates or if the growth of S terminates, S will be a most general unifier (in the case of the second-order language by Theorem 2). However, S may grow indefinitely (essential singularity).

ACKNOWLEDGMENT. I would like to express my sincere gratitude to Mr. D. A Forkes, whose discovery of a certain higher order unification algorithm inspired me to develop this method, and to Dr. A. Ehrenfeucht and Dr. A. Gabrielian for their comments and criticisms. But especially I feel in debt to Dr. D. C. Jensen, whose help, criticisms, and corrections allowed me to bring the paper to the present form.

REFERENCES

1. ANDREWS, P. Resolution in type theory. *J. Symbolic Logic 36*, 3 (1971), 414–432.
2. BLEDSOE, W. W. Splitting and reduction heuristics in automatic theorem proving. *Artificial Intelligence 2* (1971), pp. 57–78.
3. DE BRUIJN, N. G. The mathematical language AUTOMATH, its usage and some of its extensions. Symposium on Automatic Demonstration, Versailles, Dec. 1968, Springer-Verlag, 1970, pp. 29–61.
4. CHURCH, A. *The Calculi of Lambda-Conversion*. Princeton U. Press, Princeton, N. J., 1941.
5. DARLINGTON, J. L. Automatic theorem proving with equality substitutions and mathematical induction. In *Machine Intelligence 3*, D. Michie (Ed.), American Elsevier, New York, 1968, pp. 113–130.
6. DARLINGTON, J. L. A partial mechanization of second-order logic. In *Machine Intelligence 6*, B. Melzer and D. Michie (eds.), American Elsevier, New York, 1971, pp. 91–100.
7. GOULD, W. E. A matching procedure for omega-order logic. Ph.D. thesis, Princeton U., Princeton, N. J.; University Microfilms, Ann Arbor, Mich.
8. HENKIN, L. Completeness in the theory of types. *J. Symbolic Logic 15* (1960), 81–91.
9. JENSEN, D. C. A two-valued type theory model convenient for resolution theorem-proving. CSRR 2055, Dep. of Applied Analysis and Comput. Sci., U. of Waterloo, Waterloo, Ont., Canada, 1971.
10. MELTZER, B. Power amplification for theorem-provers. In *Machine Intelligence 5*, B. Meltzer and D. Michie (Eds.), American Elsevier, New York, 1970, pp. 165–180.
11. MONK, J. D. *Introduction to Set Theory*. McGraw-Hill, New York, 1969.
12. PIETRZYKOWSKI, T. A natural language for formal mathematical reasoning (TPL 2), Part I. CSRR 2015, Dep. of Applied Analysis and Comput. Sci., U. of Waterloo, Waterloo, Ont., Canada, 1970.
13. PIETRZYKOWSKI, T. A complete mechanization of second order skolemized logic. CSRR 2036, Dep. of Applied Analysis and Comput. Sci., U. of Waterloo, Waterloo, Ont., Canada, 1971.
14. ROBINSON, J. A. A machine-oriented logic based on the resolution principle. *J. ACM 12*, 1 (Jan. 1965), 23–41.
15. ROBINSON, J. A. New directions in mechanical theorem proving. Proc. IFIP Cong. 68, Vol. 1, North-Holland Pub. Co., Amsterdam, pp. 63–67.
16. ROBINSON, J. A. Mechanizing higher-order logic. In *Machine Intelligence 4*, B. Meltzer and D. Michie (Eds.), American Elsevier, New York, 1969, pp. 157–172.
17. SHOENFIELD, J. R. *Mathematical Logic*, Addison-Wesley, Reading, Mass., 1967.

RECEIVED MAY 1971; REVISED MAY 1972