# A Machine-Oriented Logic Based on the Resolution Principle

J. A. Robinson

*Argonne National Laboratory* and Rice University†*

*Abstract.* Theorem-proving on the computer, using procedures based on the fundamental theorem of Herbrand concerning the first-order predicate calculus, is examined with a view towards improving the efficiency and widening the range of practical applicability of these procedures. A close analysis of the process of substitution (of terms for variables), and the process of truth-functional analysis of the results of such substitutions, reveals that both processes can be combined into a single new process (called *resolution*), iterating which is vastly more efficient than the older cyclic procedures consisting of substitution stages alternating with truth-functional analysis stages.

The theory of the resolution process is presented in the form of a system of first-order logic with just one inference principle (the resolution principle). The completeness of the system is proved; the simplest proof-procedure based on the system is then the direct implementation of the proof of completeness. However, this procedure is quite inefficient, and the paper concludes with a discussion of several principles (called search principles) which are applicable to the design of efficient proof-procedures employing resolution as the basic logical process.

## 1. *Introduction*

Presented in this paper is a formulation of first-order logic which is specifically designed for use as the basic theoretical instrument of a computer theorem-proving program. Earlier theorem-proving programs have been based on systems of first-order logic which were originally devised for other purposes. A prominent feature of those systems of logic, which is lacking in the system described in this paper, is the relative *simplicity* of their inference principles.

Traditionally, a single step in a deduction has been required, for pragmatic and psychological reasons, to be simple enough, broadly speaking, to be apprehended as correct by a human being in a single intellectual act. No doubt this custom originates in the desire that each single step of a deduction should be indubitable, even though the deduction as a whole may consist of a long chain of such steps. The ultimate conclusion of a deduction, if the deduction is correct, follows logically from the premisses used in the deduction; but the human mind may well find the unmediated transition from the premisses to the conclusion surprising, hence (psychologically) dubitable. Part of the point, then, of the logical analysis of deductive reasoning has been to reduce complex inferences, which are beyond the capacity of the human mind to grasp as single steps, to chains of simpler inferences, each of which is within the capacity of the human mind to grasp as a single transaction.

Work performed under the auspices of the U. S. Atomic Energy Commission.

* Argonne, Illinois.

† Present address: Rice University, Houston, Texas.



Journal of the Association for Computing Machinery, Vol. 12, No. 1 (January, 1965), pp. 23–41

From the theoretical point of view, however, an inference principle need only be *sound* (i.e., allow only logical consequences of premisses to be deduced from them) and *effective* (i.e., it must be algorithmically decidable whether an alleged application of the inference principle is indeed an application of it). When the agent carrying out the application of an inference principle is a modern computing machine, the traditional limitation on the complexity of inference principles is no longer very appropriate. More powerful principles, involving perhaps a much greater amount of combinatorial information-processing for a single application, become a possibility.

In the system described in this paper, one such inference principle is used. It is called the *resolution principle*, and it is machine-oriented, rather than human-oriented, in the sense of the preceding remarks. The resolution principle is quite powerful, both in the psychological sense that it condones single inferences which are often beyond the ability of the human to grasp (other than discursively), and in the theoretical sense that it alone, as sole inference principle, forms a complete system of first-order logic. While this latter property is of no great importance, it is interesting that (as far as the author is aware) no other complete system of first-order logic has consisted of just one inference principle, if one construes the device of introducing a logical axiom, given outright, or by a schema, as a (premiss-free) inference principle.

The main advantage of the resolution principle lies in its ability to allow us to avoid one of the major combinatorial obstacles to efficiency which have plagued earlier theorem-proving procedures.

In Section 2 the syntax and semantics of the particular formalism which is used in this paper are explained.

## 2. *Formal Preliminaries*

The formalism used in this paper is based upon the notions of unsatisfiability and refutation rather than upon the notions of validity and proof. It is well known (cf. [2] and [5]) that in order to determine whether a finite set $S$ of sentences of first-order logic is satisfiable, it is sufficient to assume that each sentence in $S$ is in prenex form with no existential quantifiers in the prefix; moreover the matrix of each sentence in $S$ can be assumed to be a disjunction of formulas each of which is either an atomic formula or the negation of an atomic formula. Therefore our syntax is set up so that the natural syntactical unit is a finite set $S$ of sentences in this special form. The quantifier prefix is omitted from each sentence, since it consists just of universal quantifiers binding each variable in the sentence; furthermore the matrix of each sentence is regarded simply as the set of its disjuncts, on the grounds that the order and multiplicity of the disjuncts in a disjunction are immaterial.

Accordingly we introduce the following definitions (following in part the nomenclature of [2] and [5]):

2.1 *Variables.* The following symbols, in alphabetical order, are variables:

$$u,\ v,\ w,\ x,\ y,\ z,\ u_1,\ v_1,\ w_1,\ x_1,\ y_1,\ z_1,\ u_2,\ \cdots,\ \text{etc.}$$

2.2 *Function symbols.* The following symbols, in alphabetical order, are function symbols of degree $n$, for each $n \geqq 0$:

$$a^n, b^n, c^n, d^n, e^n, f^n, g^n, h^n, k^n, a_1{}^n, b_1{}^n, \cdots, \text{etc.}$$

When $n = 0$, the superscript may be omitted. Function symbols of degree 0 are *individual constants.*

2.3 *Predicate symbols.* The following symbols, in alphabetical order, are predicate symbols of degree $n$, for each $n \geqq 0$:

$$P^n, Q^n, R^n, P_1{}^n, Q_1{}^n, R_1{}^n, P_2{}^n, \cdots, \text{etc.}$$

The superscript may be omitted when $n$ is 0.

2.4 *The negation symbol.* The following symbol is the negation symbol: $\sim$

2.5 *Alphabetical order of symbols.* The set of all symbols is well ordered in alphabetical order by adding to the above ordering conventions the rule that variables precede function symbols, function symbols precede predicate symbols, predicate symbols precede the negation symbol, function symbols of lower degree precede function symbols of higher degree, and predicate symbols of lower degree precede predicate symbols of higher degree.

2.6 *Terms.* A variable is a term, and a string of symbols consisting of a function symbol of degree $n \geqq 0$ followed by $n$ terms is a term.

2.7 *Atomic formulas.* A string of symbols consisting of a predicate symbol of degree $n \geqq 0$ followed by $n$ terms is an atomic formula.

2.8 *Literals.* An atomic formula is a literal; and if $A$ is an atomic formula then $\sim A$ is a literal.

2.9 *Complements.* If $A$ is an atomic formula, then the two literals $A$ and $\sim A$ are said to be each other's complements, and to form, in either order, a complementary pair.

2.10 *Clauses.* A finite set (possibly empty) of literals is called a clause. The empty clause is denoted by: $\square$

2.11 *Ground literals.* A literal which contains no variables is called a ground literal.

2.12 *Ground clauses.* A clause, each member of which is a ground literal, is called a ground clause. In particular $\square$ is a ground clause.

2.13 *Well-formed expressions.* Terms and literals are (the only) well formed expressions.

2.14 *Lexical order of well-formed expressions.* The set of all well formed expressions is well ordered in lexical order by the rule that $A$ precedes $B$ just in case that $A$ is shorter than $B$ or, if $A$ and $B$ are of equal length, then $A$ has the alphabetically earlier symbol in the first symbol position at which $A$ and $B$ have distinct symbols.

In writing well-formed expressions for illustrative purposes, we follow the more readable plan of enclosing the $n$ terms following a function symbol or predicate symbol of degree $n$ by a pair of parentheses, separating the terms, if there are two or more, by commas. We can then unambiguously omit all superscripts from symbols. In writing finite sets, we follow the usual convention of

enclosing the members in a pair of braces and of separating the members by commas, with the understanding that the order of writing the members is immaterial.

2.15 *Herbrand universes.* With any set $S$ of clauses there is associated a set of ground terms called the Herbrand universe of $S$, as follows: let $F$ be the set of all function symbols which occur in $S$. If $F$ contains any function symbols of degree 0, the functional vocabulary of $S$ is $F$; otherwise it is the set $\{a\} \cup F$. The Herbrand universe of $S$ is then the set of all ground terms in which there occur only symbols in the functional vocabulary of $S$.

2.16 *Saturation.* If $S$ is any set of clauses and $P$ is any set of terms, then by $P(S)$ we denote the saturation of $S$ over $P$, which is the set of all ground clauses obtainable from members of $S$ by replacing variables with members of $P$—occurrences of the same variable in any one clause being replaced by occurrences of the same term.

2.17 *Models.* A set of ground literals which does not include a complementary pair is called a model. If $M$ is a model and $S$ is a set of ground clauses, then $M$ is a model of $S$ if, for all $C$ in $S$, $C$ contains a member of $M$. Then, in general, if $S$ is any set of clauses, and $H$ is the Herbrand universe of $S$, we say that $M$ is a model of $S$ just in case that $M$ is a model of $H(S)$.

2.18 *Satisfiability.* A set $S$ of clauses is satisfiable if there is a model of $S$; otherwise $S$ is unsatisfiable.

From the definition of satisfiability, it is clear that any set of clauses which contains $\square$ is unsatisfiable, and that the empty set of clauses is satisfiable. These two circumstances will appear quite natural as the development of our system proceeds. It is also clear that according to our semantic definitions each nonempty clause is interpreted, as explained in the informal remarks at the beginning of this section, as the universal closure of the disjunction of the literals which it contains.

2.19 *Ground resolvents.* If $C$ and $D$ are two ground clauses, and $L \subseteq C$, $M \subseteq D$ are two singletons (unit sets) whose respective members form a complementary pair, then the ground clause: $(C - L) \cup (D - M)$ is called a ground resolvent of $C$ and $D$.

Evidently any model of $\{C, D\}$ is also a model of $\{C, D, R\}$, where $R$ is a ground resolvent of $C$ and $D$. Not all pairs of ground clauses have ground resolvents, and some have more than one; but in no case, as is clear from the definition, can two ground clauses have more than a finite number of ground resolvents.

2.20 *Ground resolution.* If $S$ is any set of ground clauses, then the ground resolution of $S$, denoted by $\mathcal{R}(S)$, is the set of ground clauses consisting of the members of $S$ together with all ground resolvents of all pairs of members of $S$.

2.21 *N-th ground resolution.* If $S$ is any set of ground clauses, then the $n$th ground resolution of $S$, denoted by $\mathcal{R}^n(S)$, is defined for each $n \geqq 0$ as follows: $\mathcal{R}^0(S) = S$; and for $n \geqq 0$, $\mathcal{R}^{n+1} = \mathcal{R}(\mathcal{R}^n(S))$.

This completes the first batch of definitions. The next sections are concerned with the various forms that Herbrand's Theorem takes on in our system. To each such form, there is a type of refutation procedure which that form sug-

gests and justifies. The basic version is stated as follows (cf. [2, 4]):

HERBRAND'S THEOREM. *If $S$ is any finite set of clauses and $H$ its Herbrand universe, then $S$ is unsatisfiable if and only if some finite subset of $H(S)$ is unsatisfiable.*

## 3. *Saturation Procedures*

Is was noted in an earlier paper [5] that one can express Herbrand's Theorem in the following form:

THEOREM 1. *If $S$ is any finite set of clauses, then $S$ is unsatisfiable if and only if, for some finite subset $P$ of the Herbrand universe of $S$, $P(S)$ is unsatisfiable.*

This version of Herbrand's Theorem suggests the following sort of refutation procedure, which we call a *saturation procedure*: given a finite set $S$ of clauses, select a sequence $P_0, P_1, P_2, \cdots$, of finite subsets of the Herbrand universe $H$ of $S$, such that $P_j \subseteq P_{j+1}$ for each $j \geqq 0$, and such that $\bigcup_{j=0}^{\infty} P_j = H$. Then examine in turn the sets $P_0(S), P_1(S), P_2(S), \cdots$, for satisfiability. Evidently, for any finite subset $P$ of $H$, $P \subseteq P_j$ for some $j$, and therefore $P(S) \subseteq P_j(S)$. Therefore, by Theorem 1, if $S$ is unsatisfiable then, for some $j$, $P_j(S)$ is unsatisfiable.

Of course, any specific procedure of this sort must make the selection of $P_0, P_1, P_2, \cdots$, uniformly for all finite sets of clauses. A particularly natural way of doing this is to use the so-called levels $H_0, H_1, H_2, \cdots$, of the Herbrand universe $H$; where $H_0$ consists of all the individual constants in $H$, and $H_{n+1}$, for $n \geqq 0$, consists of all the terms in $H$ which are in $H_n$, or whose arguments are in $H_n$. In [5] we called procedures using this method *level-saturation procedures*. It was there remarked that essentially the procedures of Gilmore [4] and Davis-Putnam [2] are level-saturation procedures.

The major combinatorial obstacle to efficiency for level-saturation procedures is the enormous rate of growth of the finite sets $H_j$ and $H_j(S)$ as $j$ increases, at least for most interesting sets $S$. These growth rates were analyzed in some detail in [5], and some examples were there given of some quite simple unsatisfiable $S$ for which the earliest unsatisfiable $H_j(S)$ is so large as to be absolutely beyond the limits of feasibility.

An interesting heuristic remark is that, for every finite set $S$ of clauses which is unsatisfiable and which has a refutation one could possibly construct, there is at least one reasonably small finite subset of the Herbrand universe of $S$ such that $P(S)$ is unsatisfiable and such that $P$ is *minimal* in the sense that $Q(S)$ is satisfiable for each proper subset $Q$ of $P$. Such a $P$ was called a *proof set for* $S$ in [5]. If only, then, a benevolent and omniscient demon were available who could provide us, in reasonable time, with a proof set $P$ for each unsatisfiable finite set $S$ of clauses that we considered, we could simply arrange to saturate $S$ over $P$ and then extract a suitable refutation of $S$ from the resulting finite unsatisfiable set $P(S)$ of ground clauses. This was in fact the underlying scheme of a computer program reported in [5], in which the part of the demon is played, as best his ingenuity allows, by the mathematician using the program. What is really

wanted, to be sure, is a simulation of the proof set demon on the computer; but this would appear, intuitively, to be out of the question.

It turns out that it is not completely out of the question. In fact, the method developed in the remainder of this paper seems to come quite close to supplying the required demon as a computing process. In Section 4 we take the first major step towards the development of this method by proving more versions of Herbrand's Theorem. We also give a preliminary informal account of the rest of the argument, pending a rigorous treatment in succeeding sections.

## 4. *The Resolution Theorems and the Basic Lemma*

As a specific method for testing a finite set of ground clauses for satisfiability, the method of Davis-Putnam [4] would be hard to improve on from the point of view of efficiency. However, we now give another method, far less efficient than theirs, which plays only a theoretical role in our development, and which is much simpler to state: given the finite set $S$ of ground clauses, form successively the sets $S$, $\mathfrak{R}(S)$, $\mathfrak{R}^2(S)$, $\cdots$, until either some $\mathfrak{R}^n(S)$ contains $\square$, or does not contain $\square$ but is equal to $\mathfrak{R}^{n+1}(S)$. In the former case, $S$ is unsatisfiable; in the latter case, $S$ is satisfiable. One or other of these two terminating conditions must eventually occur, since the number of distinct clauses formable from the finite set of literals which occur in $S$ is finite, and hence in the nested infinite sequence:

4.1 $S \subseteq \mathfrak{R}(S) \subseteq \mathfrak{R}^2(S) \subseteq \cdots \subseteq \mathfrak{R}^n(S) \subseteq \cdots$,

not all of the inclusions are proper, since resolution introduces no new literals.

In view of the finite termination of the described process we can prove its correctness, as stated above, in the form of the ground resolution theorem.

Ground Resolution Theorem. *If $S$ is any finite set of ground clauses, then $S$ is unsatisfiable if and only if $\mathfrak{R}^n(S)$ contains $\square$, for some $n \geqq 0$.*

Proof. The "if" part is immediate. To prove the "only if" part, let $T$ be the terminating set $\mathfrak{R}^n(S)$ in the sequence (4.1) above, so that $T$ is closed under ground resolution. We need only show that if $T$ does not contain $\square$, then $T$ is satisfiable, and hence $S$ is satisfiable since $S \subseteq T$. Let $L_1, \cdots, L_k$ be all the distinct atomic formulas which occur in $T$ or whose complements occur in $T$. Let $M$ be the model defined as follows: $M_0$ is the empty set; and for $0 < j \leqq k$, $M_j$ is the set $M_{j-1} \cup \{L_j\}$, unless some clause in $T$ consists entirely of complements of literals in the set $M_{j-1} \cup \{L_j\}$; in which case $M_j$ is the set $M_{j-1} \cup \{\sim L_j\}$. Finally, $M$ is $M_k$. Now if $T$ does not contain $\square$, $M$ satisfies $T$. For otherwise there is a least $j$, $0 < j \leqq k$, such that some clause (say, $C$) in $T$ consists entirely of complements of literals in the set $M_j$. By the definition of $M_j$, therefore, $M_j$ is $M_{j-1} \cup \{\sim L_j\}$. Hence by the leastness of $j$, $C$ contains $L_j$. But since $M_j$ is $M_{j-1} \cup \{\sim L_j\}$, there is some clause (say, $D$) in $T$ which consists entirely of complements of literals in the set $M_{j-1} \cup \{L_j\}$. Hence by the leastness of $j$, $D$ contains $\sim L_j$. Then the clause $B = (C - \{L_j\}) \cup (D - \{\sim L_j\})$ consists entirely of complements of literals in the set $M_{j-1}$, unless $B$ is $\square$. But $B$ is a

ground resolvent of $C$ and $D$, hence is in $T$, hence is not $\square$. Thus the leastness of $j$ is contradicted and the theorem is proved.

The Ground Resolution Theorem now allows us to state a more specific form of Theorem 1, namely,

THEOREM 2. *If $S$ is any finite set of clauses, then $S$ is unsatisfiable if and only if, for some finite subset $P$ of the Herbrand universe of $S$, and some $n \geqq 0$, $\mathfrak{R}^n(P(S))$ contains $\square$.*

It is now possible to state informally the essential steps of the remaining part of the development. We are going to generalize the notions of ground resolvent and ground resolution, respectively, to the notions of resolvent and resolution, by removing the restriction that the clauses involved be only ground clauses. Any two clauses will then have zero, one or more clauses as their resolvents, but in no case more than finitely many. In the special case that $C$ and $D$ *are* ground clauses, their resolvents, if any, are precisely their ground resolvents as already defined. Similarly, the notations $\mathfrak{R}(S)$, $\mathfrak{R}^n(S)$ will be retained, with $S$ allowed to be any set of clauses. $\mathfrak{R}(S)$ will then denote the resolution of $S$, which is the set of clauses consisting of all members of $S$ together with all resolvents of all pairs of members of $S$. Again, $\mathfrak{R}(S)$ is precisely the ground resolution of $S$, already defined, whenever $S$ happens to be a set of ground clauses.

The details of how this generalization is done must await the formal definitions in Section 5. However, an informal grasp of the general notion of resolution is obtainable now, prior to its exact treatment, from simply contemplating the fundamental property which it will be shown to possess: *resolution is semicommutative with saturation*. More exactly, this property is as stated in the following basic Lemma, which is proved in Section 5:

LEMMA. *If $S$ is any set of clauses, and $P$ is any subset of the Herbrand universe of $S$, then: $\mathfrak{R}(P(S)) \subseteq P(\mathfrak{R}(S))$.*

The fact is, as will be shown here, that any ground clause which can be obtained by *first* instantiating over $P$ a pair $C$, $D$ of clauses in $S$, and *then* forming a ground resolvent of the two resulting instances, can also be obtained by instantiating over $P$ one of the finitely many resolvents of $C$ and $D$.

It is an easy corollary of the basic Lemma that $n$th resolutions are also semicommutative with saturation:

COROLLARY. *If $S$ is any set of clauses and $P$ is any subset of the Herbrand universe of $S$, then: $\mathfrak{R}^n(P(S)) \subseteq P(\mathfrak{R}^n(S))$ for all $n \geqq 0$.*

PROOF. By induction on $n$. $\mathfrak{R}^0(P(S)) = P(S) = P(\mathfrak{R}^0(S))$, so that the case $n = 0$ is trivial. And if, for $n \geqq 0$, $\mathfrak{R}^n(P(S)) \subseteq P(\mathfrak{R}^n(S))$, then:

$$\begin{aligned}
\mathfrak{R}^{n+1}(P(S)) &= \mathfrak{R}(\mathfrak{R}^n(P(S))) && \text{by definition of } \mathfrak{R}^{n+1},\\
&\subseteq \mathfrak{R}(P(\mathfrak{R}^n(S))) && \text{by the induction hypothesis, as } \mathfrak{R} \text{ preserves inclusion},\\
&\subseteq P(\mathfrak{R}(\mathfrak{R}^n(S))) && \text{by the Lemma},\\
&= P(\mathfrak{R}^{n+1}(S)) && \text{by definition of } \mathfrak{R}^{n+1},
\end{aligned}$$

and the Corollary is proved.

Now by the above Corollary to the basic Lemma we may immediately obtain a third version of Herbrand's Theorem from Theorem 2:

THEOREM 3. *If $S$ is any finite set of clauses, then $S$ is unsatisfiable if and only if, for some finite subset $P$ of the Herbrand universe of $S$, and some $n \geqq 0$, $P(\mathcal{R}^n(S))$ contains* $\square$.

Here, the order of the saturation and $n$th resolution operations is reversed. Now a rather surprising simplification of Theorem 3 can be made, on the basis of the remark that mere replacement of variables by terms cannot produce $\square$ from a nonempty clause. Hence $P(\mathcal{R}^n(S))$ will contain $\square$ if and only if $\mathcal{R}^n(S)$ contains $\square$. From Theorem 3, therefore, we immediately obtain our final version of Herbrand's Theorem, which is the main result of this paper, and which we call:

RESOLUTION THEOREM. *If $S$ is any finite set of clauses, then $S$ is unsatisfiable if and only if $\mathcal{R}^n(S)$ contains* $\square$, *for some $n \geq 0$.*

The statement of the Resolution Theorem is just that of the Ground Resolution Theorem with the word "ground" omitted. Apart, therefore, from the somewhat more complex way in which the resolvents of two clauses are computed (described in Section 5) the method suggested by the Resolution Theorem for testing a finite set $S$ of clauses for unsatisfiability is exactly like that given earlier for the case that $S$ is a set of ground clauses, and indeed it automatically reverts to that method when it is applied to a finite set of ground clauses. However, it is no longer the case in general that the nested sequence

$$S \subseteq \mathcal{R}(S) \subseteq \mathcal{R}^2(S) \subseteq \cdots \subseteq \mathcal{R}^n(S) \subseteq \cdots$$

must terminate for all finite $S$. By Church's Theorem this could not be so, for otherwise we would have a decision procedure for satisfiability for our formulation of first-order logic.

Consider now the "proof set demon" discussed in Section 3. We there supposed that if we were given a proof set $P$ for an unsatisfiable set $S$ of clauses, all we would have to do would be to compute until we encountered the first $\mathcal{R}^n(P(S))$ which contains $\square$, in order to obtain from it a formal refutation of $S$. But the Resolution Theorem assures us that by the time we had computed $\mathcal{R}^n(S)$, if not before, we would have turned up $\square$, despite our ignorance of $P$. In this sense the Resolution Theorem makes the proof set demon's role unnecessary.

In Section 5 we introduce a little more formal apparatus by a second batch of definitions, and pay off our debts by defining the general notion of resolution and proving the basic Lemma.

## 5. *Substitution, Unification and Resolution*

The following definitions are concerned with the operation of instantiation, i.e. substitution of terms for variables in well-formed expressions and in sets of well-formed expressions, and with the various auxiliary notions needed to define resolution in general.

5.1 *Substitution components.* A substitution component is any expression of

the form $T/V$, where $V$ is any variable and $T$ is any term different from $V$. $V$ is called the *variable of* the component $T/V$, and $T$ is called the *term of* the component $T/V$.

5.2 *Substitutions.* A substitution is any finite set (possibly empty) of substitution components none of the variables of which are the same. If $P$ is any set of terms, and the terms of the components of the substitution $\theta$ are all in $P$, we say that $\theta$ is a substitution over $P$. We write the substitution whose components are $T_1/V_1, \cdots, T_k/V_k$ as $\{T_1/V_1, \cdots, T_k/V_k\}$, with the understanding that the order of the components is immaterial. We use lower-case Greek letters to denote substitutions. In particular, $\epsilon$ is the *empty substitution.*

5.3 *Instantiation.* If $E$ is any finite string of symbols and

$$\theta = \{T_1/V_1, \cdots, T_k/V_k\}$$

is any substitution, then the instantiation of $E$ by $\theta$ is the operation of replacing each occurrence of the variable $V_i$, $1 \leqq i \leqq k$, in $E$ by an occurrence of the term $T_i$. The resulting string, denoted by $E\theta$, is called the instance of $E$ by $\theta$. I.e., if $E$ is the string $E_0V_{i_1}E_1 \cdots V_{i_n}E_n$, then $E\theta$ is the string $E_0T_{i_1}E_1 \cdots T_{i_n}E_n$. Here, none of the substrings $E_j$ of $E$ contain occurrences of the variables $V_1, \cdots, V_k$, some of the $E_j$ are possibly null, $n$ is possibly 0, and each $V_{i_j}$ is an occurrence of one of the variables $V_1, \cdots, V_k$. Any string $E\theta$ is called an instance of the string $E$. If $C$ is any set of strings and $\theta$ a substitution, then the instance of $C$ by $\theta$ is the set of all strings $E\theta$, where $E$ is in $C$. We denote this set by $C\theta$, and say that it is an instance of $C$.

5.4 *Standardizations.* If $C$ is any finite set of strings, and $V_1, \cdots, V_k$ are all the distinct variables, in alphabetical order, which occur in strings in $C$, then the $x$-standardization of $C$, denoted by $\xi_C$, is the substitution $\{x_1/V_1, \cdots, x_k/V_k\}$ and the $y$-standardization of $C$, denoted by $\eta_C$, is the substitution

$$\{y_1/V_1, \cdots, y_k/V_k\}.$$

5.5 *Composition of substitutions.* If $\theta = \{T_1/V_1, \cdots, T_k/V_k\}$ and $\lambda$ are any two substitutions, then the set $\theta' \cup \lambda'$, where $\lambda'$ is the set of all components of $\lambda$ whose variables are not among $V_1, \cdots, V_k$, and $\theta'$ is the set of all components $T_i\lambda/V_i$, $1 \leqq i \leqq k$, such that $T_i\lambda$ is different from $V_i$, is called the composition of $\theta$ and $\lambda$, and is denoted by $\theta\lambda$.

It is straightforward to verify that $\epsilon\theta = \theta\epsilon = \theta$ for any substitution $\theta$. Also, composition of substitutions enjoys the associative property $(\theta\lambda)\mu = \theta(\lambda\mu)$, so that we may omit parentheses in writing multiple compositions of substitutions.

The point of the composition operation on substitutions is that, when $E$ is any string, and $\sigma = \theta\lambda$, the string $E\sigma$ is just the string $E\theta\lambda$, i.e. the instance of $E\theta$ by $\lambda$.

These properties of the composition of substitutions are established by the following propositions.

5.5.1. *$(E\sigma)\lambda = E(\sigma\lambda)$ for all strings $E$ and all substitutions $\sigma$, $\lambda$.*

PROOF. Let $\sigma = \{T_1/V_1, \cdots, T_k/V_k\}$, $\lambda = \{U_1/W_1, \cdots, U_m/W_m\}$ and $E = E_0V_{i_1}E_1 \cdots V_{i_n}E_n$ as explained in (5.3) above. Then by definition $E\sigma =$

$E_0T_{i_1}E_1 \cdots T_{i_n}E_n$, and $(E\sigma)\lambda = \bar{E}_0\bar{T}_{i_1}\bar{E}_1 \cdots \bar{T}_{i_n}\bar{E}_n$, where each $\bar{T}_{i_j}$ is $T_{i_j}\lambda$, and each $\bar{E}_j$ is $E_j\lambda'$, where $\lambda'$ is the set of all components of $\lambda$ whose variables are not among $V_1, \cdots, V_k$ (since none of these variables occur in any $E_j$). But $\sigma\lambda = \sigma' \cup \lambda'$, where each component of $\sigma'$ is just $\bar{T}_i/V_i$ whenever $\bar{T}_i$ is different from $V_i$. Hence $E(\sigma\lambda) = \bar{E}_0\bar{T}_{i_1}\bar{E}_1 \cdots \bar{T}_{i_n}\bar{E}_n$.

5.5.2. *For any substitutions* $\sigma$, $\lambda$: *if* $E\sigma = E\lambda$ *for all strings* $E$, *then* $\sigma = \lambda$.

PROOF. Let $V_1, \cdots, V_k$ include all the variables of the components of $\sigma$ and $\lambda$; then $V_j\sigma = V_j\lambda$, for $1 \leqq j \leqq k$. Then all the components of $\sigma$ and $\lambda$ are the same.

5.5.3. *For any substitutions* $\sigma$, $\lambda$, $\mu$: $(\sigma\lambda)\mu = \sigma(\lambda\mu)$.

PROOF. Let $E$ be any string. Then by 5.5.1,

$$\begin{aligned} E((\sigma\lambda)\mu) &= (E(\sigma\lambda))\mu \\ &= ((E\sigma)\lambda)\mu \\ &= (E\sigma)(\lambda\mu) \\ &= E(\sigma(\lambda\mu)). \end{aligned}$$

Hence $(\sigma\lambda)\mu = \sigma(\lambda\mu)$ by (5.5.2).

We shall also have occasion to use the following distributive property.

5.5.4. *For any sets* $A$, $B$ *of strings and substitution* $\lambda$: $(A \cup B)\lambda = A\lambda \cup B\lambda$.

5.6 *Disagreement sets.* If $A$ is any set of well-formed expressions, we call the set $B$ the disagreement set of $A$ whenever $B$ is the set of all well-formed subexpressions of the well-formed expressions in $A$, which begin at the first symbol position at which not all well-formed expressions in $A$ have the same symbol. *Example*:

$$A = \{P(x, h(x, y), y), P(x, k(y), y), P(x, a, b)\},$$

Disagreement set of $A = \{h(x, y), k(y), a\}$.

Evidently, if $A$ is nonempty and is not a singleton, then the disagreement set of $A$ is nonempty and is not a singleton.

5.7 *Unification.* If $A$ is any set of well-formed expressions and $\theta$ is a substitution, then $\theta$ is said to unify $A$, or to be a unifier of $A$, if $A\theta$ is a singleton. Any set of well-formed expressions which has a unifier is said to be unifiable.

Evidently, if $\theta$ unifies $A$, but $A$ is not a singleton, then $\theta$ unifies the disagreement set of $A$.

5.8 *Unification Algorithm.* The following process, applicable to any finite nonempty set $A$ of well-formed expressions, is called the Unification Algorithm:

*Step* 1. Set $\sigma_0 = \epsilon$ and $k = 0$, and go to step 2.
*Step* 2. If $A\sigma_k$ is not a singleton, go to step 3. Otherwise, set $\sigma_A = \sigma_k$ and terminate.
*Step* 3. Let $V_k$ be the earliest, and $U_k$ the next earliest, in the lexical ordering of the disagreement set $B_k$ of $A\sigma_k$. If $V_k$ is a variable, and does not occur in $U_k$, set $\sigma_{k+1} = \sigma_k\{U_k/V_k\}$, add 1 to $k$, and return to step 2. Otherwise, terminate.

This definition requires justification in the form of a proof that the given process is in fact an algorithm. In fact the process always terminates for any

finite nonempty set of well-formed expressions, for otherwise there would be generated an infinite sequence $A$, $A\sigma_1$, $A\sigma_2$, $\cdots$ of finite nonempty sets of well-formed expressions with the property that each successive set contains one less variable than its predecessor (namely, $A\sigma_k$ contains $V_k$ but $A\sigma_{k+1}$ does not). But this is impossible, since $A$ contains only finitely many distinct variables.

5.9 *Most general unifiers.* If $A$ is a finite nonempty set of well-formed expressions for which the Unification Algorithm terminates in step 2, the substitution $\sigma_A$ then available as output of the Unification Algorithm is called the most general unifier of $A$, and $A$ is then said to be most generally unifiable.

5.10 *Key triples.* The ordered triple $\langle L, M, N\rangle$ of finite sets of literals is said to be a key triple of the ordered pair $\langle C, D\rangle$ of clauses just in case the following conditions are satisfied.

5.10.1. $L$ and $M$ are nonempty, and $L \subseteq C$, $M \subseteq D$.

5.10.2. $N$ is the set of atomic formulas which are members, or complements of members, of the set $L\xi_C \cup M\eta_D$ (cf. definition (5.4)).

5.10.3. $N$ is most generally unifiable, with most general unifier $\sigma_N$.

5.10.4. The sets $L\xi_C\sigma_N$ and $M\eta_D\sigma_N$ are singletons whose members are complements.

Evidently, a pair $\langle C, D\rangle$ of clauses has at most a finite number of key triples, and possibly none at all.

5.11 *Resolvents.* A resolvent of the two clauses $C$ and $D$ is any clause of the form: $(C - L)\xi_C\sigma_N \cup (D - M)\eta_D\sigma_N$ where $\langle L, M, N\rangle$ is a key triple of $\langle C, D\rangle$.

By the remark following definition (5.10) it is clear that two clauses $C$ and $D$ can have at most finitely many resolvents, and possibly none at all.

5.12 *Resolutions.* If $S$ is any set of clauses then the resolution of $S$, denoted by $\mathfrak{R}(S)$, is the set of all clauses which are members of $S$ or resolvents of members of $S$.

5.13 *$N$-th resolution.* The $n$th resolution of $S$, where $S$ is any set of clauses, is denoted by $\mathfrak{R}^n(S)$ and is defined for all $n \geqq 0$ exactly analogously to definition (2.21).

This completes our second group of definitions. The definition of $\mathfrak{R}(S)$ as given is adequate for our theoretical argument, but in practice one would not include in it both the resolvents of $\langle C, D\rangle$ and the resolvents of $\langle D, C\rangle$, since these are in fact identical up to a change of variables. When $C$ and $D$ are both ground clauses, the resolvents of $\langle C, D\rangle$ are actually identical with those of $\langle D, C\rangle$, and are precisely the ground resolvents of $C$ and $D$, as is easily verified.

It now remains to prove the basic Lemma, which will be done after we have first proved the following theorem establishing the basic property of unification, which we need in the proof of the Lemma and elsewhere in our theory:

UNIFICATION THEOREM. *Let $A$ be any finite nonempty set of well-formed expressions. If $A$ is unifiable, then $A$ is most generally unifiable with most general unifier $\sigma_A$; moreover, for any unifier $\theta$ of $A$ there is a substitution $\lambda$ such that $\theta = \sigma_A\lambda$.*

PROOF. It will suffice to prove that under the hypotheses of the theorem the Unification Algorithm will terminate, when applied to $A$, at step 2; and that for each $k \geqq 0$ until the Unification Algorithm so terminates, the equation

5.14. $\theta = \sigma_k\lambda_k$

holds at step 2 for some substitution $\lambda_k$. For $k = 0$, (5.14) holds with $\lambda_0 =$ since $\sigma_0 = \epsilon$. Now assume that, for $k \geqq 0$, (5.14) holds at step 2 for some s stitution $\lambda_k$. Then either $A\sigma_k$ is a singleton, in which case the Unification gorithm terminates at step 2 with $\sigma_A = \sigma_k$ the most general unifier of $A$ $\lambda = \lambda_k$ the required substitution; or the Unification Algorithm transfers to s 3. In the latter case, since $\lambda_k$ unifies $A\sigma_k$, (by (5.14), since $\theta$ unifies $A$) $\lambda_k$ m also unify the disagreement set $B_k$ of $A\sigma_k$. Hence the $V_k$ and $U_k$ defined in s 3 of the Unification Algorithm satisfy the equation

5.15. $V_k\lambda_k = U_k\lambda_k$.

Since $B_k$ is a disagreement set, the well-formed expressions in $B_k$ cannot begin with the same symbol; hence they cannot all begin with symbols which not variables, since $B_k$ is unifiable. Therefore at least one well-formed express in $B_k$ begins with a variable, and hence is a variable, since it is well-form Since variables precede all other well-formed expressions in the lexical order, since $V_k$ is the earliest well-formed expression in $B_k$, it follows that $V_k$ i variable. Now if $V_k$ occurs in $U_k$, $V_k\lambda_k$ occurs in $U_k\lambda_k$, but since $V_k$ and $U_k$ distinct well-formed expressions this is impossible because of (5.15). Theref $V_k$ does not occur in $U_k$. Hence the Unification Algorithm will not terminate step 3, but will return to step 2 with $\sigma_{k+1} = \sigma_k\{U_k/V_k\}$. Now let $\lambda_{k+1} = \lambda_k$ $\{V_k\lambda_k/V_k\}$. Then:

$$\begin{aligned}
\lambda_k &= \{V_k\lambda_k/V_k\} \cup \lambda_{k+1} && \text{by definition of } \lambda_{k+1}, \\
&= \{U_k\lambda_k/V_k\} \cup \lambda_{k+1} && \text{by (5.15)}, \\
&= \{U_k\lambda_{k+1}/V_k\} \cup \lambda_{k+1} && \text{since } V_k \text{ does not occur in } U_k, \\
&= \{U_k/V_k\}\lambda_{k+1} && \text{by definition (5.5)}.
\end{aligned}$$

Hence by (5.14) $\theta = \sigma_{k+1}\lambda_{k+1}$. Thus (5.14) holds for all $k \geqq 0$ until the U fication Algorithm terminates in step 2, and the theorem is proved.

We are now in a position to prove the basic Lemma, which we state here ag for convenience.

LEMMA. *If $S$ is any set of clauses and $P$ is any subset of the Herbrand unive of $S$, then:* $\mathfrak{R}(P(S)) \subseteq P(\mathfrak{R}(S))$.

PROOF. Assume that $A \in \mathfrak{R}(P(S))$. Then either $A \in P(S)$, in which c $A \in P(\mathfrak{R}(S))$ since $S \subseteq \mathfrak{R}(S)$; or $A$ is a ground resolvent of two ground clau $C\alpha$, $D\beta$, where $C \in S$, $D \subset S$, $\alpha = \{T_1/V_1, \cdots, T_k/V_k\}$, where $V_1, \cdots, V_k$ all the distinct variables of $C$ in alphabetical order and $T_1, \cdots, T_k$ are in and $\beta = \{U_1/W_1, \cdots, U_m/W_m\}$, where $W_1, \cdots, W_m$ are all the distinct va ables of $D$ in alphabetical order and $U_1, \cdots, U_m$ are in $P$. In that case, $A$ $(C - L)\alpha \cup (D - M)\beta$, where $L \subseteq C$, $M \subseteq D$, $L$ and $M$ are nonempty, a $L\alpha$, $M\beta$ are singletons whose members are complements. Let

$$\theta = \{T_1/x_1, \cdots, T_k/x_k, U_1/y_1, \cdots, U_m/y_m\}.$$

Then it follows that $A = (C - L)\xi_C\theta \cup (D - M)\eta_D\theta$ and that $L\xi_C\theta = L\alpha$ a $M\eta_D\theta = M\beta$. Therefore $\theta$ unifies the set $N$ of atomic formulas which are me

bers, or complements of members, of the set $L\xi_C \cup M\eta_D$. Hence by the Unification Theorem $N$ has a most general unifier $\sigma_N$, and there is a substitution $\lambda$ over $P$ such that $\theta = \sigma_N\lambda$. Hence $L\xi_C\sigma_N\lambda = L\alpha$ and $M\eta_D\sigma_N\lambda = M\beta$, and therefore $L\xi_C\sigma_N$ and $M\eta_D\sigma_N$ are singletons whose members are complements. It follows that $\langle L, M, N\rangle$ is a key triple of $\langle C, D\rangle$, and hence that the clause

$$B = (C - L)\xi_C\sigma_N \cup (D - M)\eta_D\sigma_N$$

is a resolvent of $C$ and $D$; hence $B \in \mathfrak{R}(S)$. But since $\theta = \sigma_N\lambda$, it follows by (5.5.4) that $A = B\lambda$ and therefore that $A \in P(\mathfrak{R}(S))$. The proof is complete.

The hypotheses of the Lemma do not entail the opposite inclusion $P(\mathfrak{R}(S)) \subseteq \mathfrak{R}(P(S))$. As a simple counterexample, consider:

$$S = \{\{Q(x, f(y))\}, \{\sim Q(g(y), x)\}\}, \qquad P = \{a\}.$$

A short investigation shows that $P(\mathfrak{R}(S))$ contains $\square$ (since $\mathfrak{R}(S)$ does) while $\mathfrak{R}(P(S))$ does not. Thus $S$ is unsatisfiable, but $P$ is not a proof set for $S$.

## 6. *The Resolution Principle: Refutations*

The single inference principle of our system of logic, mentioned in Section 1, is the *resolution principle*, namely: *From any two clauses $C$ and $D$, one may infer a resolvent of $C$ and $D$.*

By a *refutation* of the set $S$ of clauses we mean a finite sequence $B_1, \cdots, B_n$ of clauses such that (a) each $B_i$, $1 \leqq i \leqq n$, is either in $S$ or is a resolvent of two earlier clauses in the sequence, and (b) $B_n$ is $\square$.

It is immediate from the Resolution Theorem that a finite set $S$ of clauses is unsatisfiable if and only if there is a refutation of $S$. Thus the Resolution Theorem is the completeness theorem for our system of logic.

Two examples of refutations will illustrate the workings of the system.

*Example* 1. The set containing just the two clauses $C_1$ and $C_2$, where

$$C_1 = \{Q(x, g(x), y, h(x, y), z, k(x, y, z))\}$$

$$C_2 = \{\sim Q(u, v, e(v), w, f(v, w), x)\}$$

has the refutation $C_1, C_2, \square$. Note that $\langle C_1, C_2\rangle$ has the key triple $\langle C_1, C_2, N\rangle$, where $N$ is the set

$$\{Q(x_1, g(x_1), x_2, h(x_1, x_2), x_3, k(x_1, x_2, x_3)),$$
$$Q(y_1, y_2, e(y_2), y_3, f(y_2, y_3), y_4)\}.$$

The reader can verify in a few minutes of computation with the Unification Algorithm that $\sigma_N$ is the substitution with the components:

$$y_1/x_1, \qquad h(y_1, e(g(y_1)))/y_3$$
$$g(y_1)/y_2, \qquad f(g(y_1), h(y_1, e(g(y_1))))/x_3$$
$$e(g(y_1))/x_2, \qquad h(y_1, e(g(y_1)), f(g(y_1), h(y_1, e(g(y_1)))))/y_4$$

and that then $C_1\xi_{C_1}\sigma_N$ and $C_2\eta_{C_2}\sigma_N$ are singletons whose members are complements.

This example illustrates the way in which a proof set is automatically computed as a by-product of the resolution operation. The terms of the above substitution components become those of a proof set for $\{C_1, C_2\}$ when the variable $y_1$ is replaced throughout by any term of the Herbrand universe of $\{C_1, C_2\}$, e.g. by the individual constant "$a$." It is interesting to note that the earliest level of this Herbrand universe $H$ to include such a proof set is $H_5$, which has of the order of $10^{64}$ members. Consequently $H_5(\{C_1, C_2\})$ has of the order of $10^{256}$ members. A level-saturation procedure would not find this example feasible.

*Example* 2. A more interesting example is one which was discussed in [5]. It arises from the following algebraic problem.

*Prove that in any associative system which has left and right solutions $x$ and $y$ for all equations $x \cdot a = b$ and $a \cdot y = b$, there is a right identity element.*

To formalize this problem in our logic, we deny the alleged conclusion, and try to refute the set containing the clauses (where $Q(x, y, z)$ is to mean $x \cdot y = z$):

| | |
|---|---|
| $C_1$ : $\{\sim Q(x, y, u), \sim Q(y, z, v), \sim Q(x, v, w), Q(u, z, w)\}$ | Associativity |
| $C_2$ : $\{\sim Q(x, y, u), \sim Q(y, z, v), \sim Q(u, z, w), Q(x, v, w)\}$ | Associativity |
| $C_3$ : $\{Q(g(x, y), x, y)\}$ | Existence of left and right |
| $C_4$ : $\{Q(x, h(x, y), y)\}$ | solutions |
| $C_5$ : $\{Q(x, y, f(x, y))\}$ | Closure under $\cdot$ |
| $C_6$ : $\{\sim Q(k(x), x, k(x))\}$ | No right identity. |

By adding the following resolvents, we get a refutation:

$C_7$ : $\{\sim Q(y_1, x_6, y_1), Q(y_2, x_6, y_2)\}$

$C_8$ : $\{\sim Q(y_2, y_2, y_1)\}$

$C_9$ : $\square$

*Commentary.* $C_7$ is the resolvent of the pair $\langle C_1, C_3\rangle$ for the key triple

$$\langle\{\sim Q(x, y, u), \sim Q(x, v, w)\}, \{Q(g(x, y), x, y)\}, N\rangle$$

where $N$ is the set $\{Q(x_4, x_5, x_1), Q(x_4, x_2, x_3), Q(g(y_1, y_2), y_1, y_2)\}$. The $\sigma_N$ computed for this $N$ by the Unification Algorithm is

$$\{y_2/x_1, y_1/x_2, y_2/x_3, g(y_1, y_2)/x_4, y_1/x_5\},$$

as is easily verified. $C_8$ is the only resolvent of $\langle C_6, C_7\rangle$, and $\square$ is the only resolvent of $\langle C_4, C_8\rangle$.

This example illustrates the way in which the single steps in a refutation made with the resolution principle go beyond, in their complexity, the capacity of the human mind to apprehend their correctness in one single intellectual act. By taking larger bites, so to speak, the resolution principle in this case permits a very compact, not to say elegant, piece of reasoning. $C_2$ and $C_5$ are not used as premisses in the refutation, although this has nothing to do with the resolution principle. Hence a nonredundant refutation for this example is the sequence: $C_1, C_3, C_4, C_6, C_7, C_8, \square$.

7. *Refutation Procedures, Search Principles*

The foregoing discussion was intended only to establish the theoretical framework, in the form of a special system of logic, for the design of theorem-proving programs, i.e. in the present case, refutation procedures. No attempt has been made thus far to discuss the question of developing efficient refutation procedures, and in this final section of the paper we briefly discuss this question.

The raw implementation of the Resolution Theorem would produce a very inefficient refutation procedure, namely, the procedure would consist of computing, given the finite set $S$ of clauses as input, the sequence of sets $S$, $\mathfrak{R}(S)$, $\mathfrak{R}^2(S)$, $\cdots$, until one is encountered, say, $\mathfrak{R}^n(S)$, which either contains $\square$ or else does not contain $\square$ but is equal to its successor $\mathfrak{R}^{n+1}(S)$. In the former case, a refutation of $S$ is obtained by tracing back the genesis of $\square$; in the latter case the conclusion is that $S$ is satisfiable. By Church's Theorem [1] we know that for some inputs $S$ this procedure, and in general all correct refutation procedures, will not terminate in either of these two ways but will continue computing indefinitely.

In some cases we can foresee the nonterminating behavior. Consider the example of the set $S$ whose members are:

$$C_1: \ \{Q(a)\}, \qquad C_2: \ \{\sim Q(x), Q(f(x))\}.$$

(The reader will recognize this as the formulation, in our logic, of a fragment of Peano's postulates for the natural numbers, with "$Q(x)$" for "$x$ is a natural number," "$a$" for "0," and "$f(x)$" for "the successor of $x$".) It is easy to see that for this $S$ the procedure described above would generate successively the resolvents $\{Q(f(a))\}$, $\{Q(f(f(a)))\}$, $\{Q(f(f(f(a))))\}$, $\cdots$, etc., *ad infinitum*.

This example suggests our attempting to formulate a principle which would allow us effectively to recognize the particular indefinite continuation phenomenon which it exhibits, so that we might incorporate the principle into a refutation procedure and cause it to terminate for this $S$ and for other similar examples. Such a principle, which we call the *purity principle*, is available, based on the notion of a literal being pure in a set $S$ of clauses. We define this notion as follows.

7.1 *Pure literals.* If $S$ is any finite set of clauses, $C$ a clause in $S$, and $L$ a literal in $C$ with the property that there is no key triple $\langle \{L\}, M, N\rangle$ for any pair $\langle C, D\rangle$ of clauses, where $D$ is any clause in $S - \{C\}$, then $L$ is said to be pure in $S$.

The purity principle is then based on the following theorem.

PURITY THEOREM. *If $S$ is any finite set of clauses, and $L \in C \in S$ is a literal which is pure in $S$, then $S$ is satisfiable if and only if $S - \{C\}$ is satisfiable.*

PROOF. If $S$ is satisfiable then so is $S - \{C\}$ since it is a subset of $S$. If $S - \{C\}$ is satisfiable, then there is a model $A$ of $S - \{C\}$, every literal in which occurs in some clause of $H(S)$, where $H$ is the Herbrand universe of $S$. Let $N$ be the set of all ground literals $L\theta$, where $\theta$ is a substitution over $H$, and let $K$ consist of all complements of members of $N$. Then the set $P = N \cup (A - K)$ is a

model; moreover it is a model of $S$, since every clause in $H(\{C\})$ contains a member of $P$ (namely a member of $N$), and every clause in $H(S - \{C\})$ contains a member of $P$, namely a member of $A - K$; for no clause in $H(S - \{C\})$ contains a member of $K$, since otherwise, if $D\beta$ were such a clause, with $D \in S - \{C\}$, then there would be an $M \subseteq D$ such that $M\beta$ would be a singleton containing a member of $K$. Then there would be some substitution $\alpha$ over $H$ such that $\{L\}\alpha$, $M\beta$ contained complementary singletons. Hence by the same argument as in the proof of the Lemma, there would be a key triple $\langle \{L\}, M, N\rangle$ of the pair $\langle C, D\rangle$, contradicting the purity of $L$ in $S$. The theorem is proved.

The *purity principle* is then simply the following: *One may delete, from a finite set $S$ of clauses, any clause containing a literal which is pure in $S$.*

When $S$ is the little Peano example given earlier, i.e., is the set containing just the two clauses

$$C_1: \quad \{Q(a)\}, \qquad C_2: \quad \{\sim Q(x), \underline{Q(f(x))}\},$$

we see that the underlined literal in $C_2$ is pure in $S$. Hence we may delete $C_2$, obtaining the set $S - \{C_2\}$ whose only clause is

$$C_1: \quad \{\underline{Q(a)}\}.$$

But of course the underlined literal is, trivially, pure in $S - \{C_2\}$; hence we may delete $C_1$, obtaining the set $S - \{C_1\} - \{C_2\}$, which is empty, hence satisfiable. Hence by the Purity Theorem, $S$ is satisfiable.

Thus a refutation procedure incorporating the purity principle as well as the resolution principle "converges" for more finite sets of clauses than a procedure based on the resolution principle alone. Such principles as the purity principle we call *search principles*, to distinguish them from inference principles.

There is another search principle which, though not increasing the range of convergence, does help to increase the rate of convergence, of refutation procedures. We call this principle the *subsumption principle* and base it on the following definition.

7.2 *Subsumption.* If $C$ and $D$ are two distinct nonempty clauses, we say that $C$ subsumes $D$ just in case there is a substitution $\sigma$ such that $C\sigma \subseteq D$.

The following theorem establishes the basic property of subsumption.

SUBSUMPTION THEOREM. *If $S$ is any finite set of clauses, and $D$ is any clause in $S$ which is subsumed by some clause in $S - \{D\}$, then $S$ is satisfiable if and only if $S - \{D\}$ is satisfiable.*

PROOF. We need only show that if $M$ is a model of $S - \{D\}$, then $M$ is a model of $S$. Let $M$ be a model of $S - \{D\}$, and suppose that $C \in S - \{D\}$ subsumes $D$. Then there is a substitution $\sigma$ such that $C\sigma \subseteq D$. Since $D \in S$, the terms of the components of $\sigma$ must be formed from function symbols in the functional vocabulary of $S$, together possibly with variables. Hence every ground instance of $C\sigma$ over $H$ is a ground instance of $C$ over $H$, and hence contains a member of $M$. But every ground instance $D\lambda$ of $D$ includes the ground instance $C\sigma\lambda$ of $C$, and hence contains a member of $M$. So $M$ is a model of $S$ and the theorem is proved.

The *subsumption principle* is then simply the following: *One may delete, from a finite set $S$ of clauses, any clause $D$ which is subsumed by a clause in $S - \{D\}$.*

In order to make the subsumption principle available for incorporation into a refutation procedure, we must give an algorithm for deciding whether one clause $C$ subsumes another clause $D$. Such an algorithm is the following Subsumption Algorithm:

*Step* 1. Let $V_1, \cdots, V_m$ be all the distinct variables, in alphabetical order, of $D$. Let $J_1, \cdots, J_m$ be distinct individual constants, none of which occur in $C$ or $D$. Let $\theta = \{J_1/V_1, \cdots, J_m/V_m\}$. Compute $D\theta$ and go to step 2.

*Step* 2. Set $A_0 = \{C\}$, $k = 0$, and go to step 3.

*Step* 3. If $A_k$ does not contain $\square$, let $A_{k+1}$ be the set of all clauses of the form $(K\sigma_N - M\sigma_N)$, where $K \in A_k$, $M \subseteq K$, $N = M \cup \{P\}$, for some $P \in D\theta$, and $N$ is most generally unifiable with most general unifier $\sigma_N$; and go to step 4. Otherwise, terminate.

*Step* 4. If $A_{k+1}$ is nonempty, add 1 to $k$ and return to step 3. Otherwise, terminate.

That this is an algorithm is clear from the fact that each clause in $A_{k+1}$ is smaller, by at least one literal, than the clause in $A_k$ from which it was obtained. Hence, since the only clause in $A_0$ has but finitely many literals, the sequence $A_0, A_1, \cdots$, must eventually contain a set which contains $\square$ or is empty.

That the Subsumption Algorithm is correct is shown by the following argument that it terminates in step 3 if and only if $C$ subsumes $D$.

If $C$ subsumes $D$, then $C\sigma \subseteq D$ for some $\sigma$. Hence $C\sigma\theta \subseteq D\theta$. Hence $C\mu \subseteq D\theta$, for some $\mu$. Now assume, for $k \geqq 0$, that $K \in A_k$ and that, for some $\mu$, $K\mu \subseteq D\theta$. If $K$ is not $\square$, let $P$ be a literal in $K\mu \cap D\theta$. Then there is an $M \subseteq K$ such that $N = (M \cup \{P\})$ is unified by $\mu$. Therefore by the Unification Theorem $N$ has a most general unifier $\sigma_N$, and the clause $G = (K\sigma_N - M\sigma_N)$ is in $A_{k+1}$. But by the Unification Theorem $\mu = \sigma_N\lambda$, for some $\lambda$, hence $K\sigma_N\lambda \subseteq D\theta$. Therefore $G\lambda \subseteq D\theta$. Since $C \in A_0$, this shows that each $A_k$, $k \geqq 0$, either contains $\square$ or is nonempty. Hence the Subsumption Algorithm does not terminate in step 4. Therefore it terminates in step 3.

If the Subsumption Algorithm terminates in step 3, for $C$ and $D$ as input, then there is a finite sequence $C_0, C_1, \cdots, C_{n+1}$ of clauses such that $C_0 = C$, $C_{n+1} = \square$, and, for $0 \leqq j \leqq n$, $C_{j+1} = C_j\sigma_j - M_j\sigma_j$, where $M_j \subseteq C_j$, and $\sigma_j$ is the most general unifier of $M_j \cup \{P\}$, where $P \in D\theta$. It follows that (since $M_j\sigma_j$ contains no variables, $0 \leqq j \leqq n$) we have

$$C_{n+1} = \square = C\sigma_0\sigma_1 \cdots \sigma_n - M_0\sigma_0 - M_1\sigma_1 - \cdots - M_n\sigma_n,$$

i.e. that $C\sigma_0\sigma_1 \cdots \sigma_n \subseteq (M_0\sigma_0 \cup M_1\sigma_1 \cup \cdots \cup M_n\sigma_n) \subseteq D\theta$. Hence, for some $\lambda$, $C\lambda \subseteq D\theta$. Let $\sigma$ be the substitution obtained from $\lambda$ by the replacement, in each component of $\lambda$ of $J_i$ by $V_i$, for $1 \leqq i \leqq m$. Then $C\sigma \subseteq D$.

A particularly useful application of the subsumption principle is the following: Suppose a resolvent $R$ of $C$ and $D$ subsumes one of $C$, $D$; then in adding $R$ by the resolution principle we may simultaneously delete, by the subsumption principle, that one of $C$, $D$ which $R$ subsumes. This combined operation amounts to replacing $C$ or $D$ by $R$; accordingly we name the principle involved the *replacement principle.*

The following example, used by Gilmore [4], Davis-Putnam [2] and Friedman [3], illustrates the utility of these search principles in speeding up convergence. Consider the set $S$ whose members are:

$C_1$ : $\underset{(6)}{\{P(x_1, x_2)\}}$

$C_2$ : $\{\underset{(1)}{\sim P(y_2, f(y_1, y_2))}, \underset{(2)}{\sim P(f(y_1, y_2), f(y_1, y_2))}, Q(y_1, y_2)\}$

$C_3$ : $\{\underset{(3)}{\sim P(y_2, f(y_1, y_2))}, \underset{(4)}{\sim P(f(y_1, y_2), f(y_1, y_2))}, \underset{(5)}{\sim Q(y_1, f(y_1, y_2))}, \sim Q(f(y_1, y_2), f(y_1, y_2))\}$

and we obtain the set $S'$ whose only members are:

$C_4$ : $\{Q(y_1, y_2)\}$

$C_5$ : $\{\sim Q(f(y_1, y_2), f(y_1, y_2))\}$

in six stages which may be followed through by deleting the underlined literals and the underlined clause, in the order indicated. This gives the set of clauses at each stage. Deletions (1) through (5) are by virtue of the replacement principle; deletion (6), of the entire clause $C_1$, is by virtue of the purity principle. The set $S'$ in turn is found immediately to be unsatisfiable, since $C_4$ and $C_5$ have $\square$ as their only resolvent.

Gilmore's 704 program failed to converge after 21 minutes' running time, when given this example. The more efficient procedure of Davis and Putnam converges, for this example, in 30 minutes of hand computation.

The application, to a finite set $S$ of clauses, of any of the three search principles we have described, produces a set $S'$ which either has fewer clauses than $S$ or has the same number of clauses as $S$ but with one or more shorter clauses. An obvious method of exploiting these principles in a refutation procedure is therefore never to add new clauses, by the resolution principle, except to a set to which the three principles are no longer applicable. We might call such sets *irreducible*. The way in which such a procedure would terminate, for satisfiable $S$ within its range of convergence, would then be with a set which is either empty (as in the Peano example) or nonempty, irreducible, and with the property that each resolvent of any pair of its clauses is subsumed by some one of its clauses.

There are further search principles of this same general sort, which are less simple than those discussed in this section. A sequel to the present paper is planned in which the theoretical framework developed here will be used as the basis for a more extensive treatment of search principles and of the design of refutation procedures. This section has been merely a sketch of the general nature of the problem, and a brief view of some of the approaches to it.

*Acknowledgments.* I should like to express my indebtedness to my colleagues Dr. George A. Robinson and Dr. Lawrence T. Wos, of the Argonne National Laboratory, and to Professor William Davidon of Haverford College, for their

invaluable insights and criticisms concerning the basic concepts of this paper. My thanks are also due to the ACM referees, and to Dr. T. Hailperin of the Sandia Corporation, whose comments on and criticisms of a prior version of the paper greatly facilitated the writing of the present complete revision.

RECEIVED SEPTEMBER, 1963; REVISED AUGUST, 1964

## REFERENCES

1. CHURCH, A. A note on the Entscheidungsproblem. *J. Symb. Logic 1* (1936), 40–41. Correction, ibid., 101–102.
2. DAVIS, M., AND PUTNAM, H. A computing procedure for quantification theory. *J. ACM 7* (Mar. 1960), 201–215.
3. FRIEDMAN, J. A semi-decision procedure for the functional calculus. *J. ACM 10* (Jan. 1963), 1–24.
4. GILMORE, P. C. A proof method for quantification theory. *IBM J. Res. Develop. 4* (1960), 28–35.
5. ROBINSON, J. A. Theorem-proving on the computer. *J. ACM 10* (Apr. 1963), 163–174.