Technical Report CS75024-R

AN INTRODUCTION TO THE CONCEPTS AND TECHNIQUES OF AUTOMATED THEOREM-PROVING

Deborah Young Macock

October 1975

Language Research Center
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061

The work represented here was supported by the National Science Foundation Grant No. DCR74-18108.

Abstract

This paper summarizes the theoretical foundations and basic methodology of automated theorem-proving. The material presented includes a review of the predicate calculus, the transformation of a first-order wff into clause normal form, J. A. Robinson's resolution principle, semantic resolution, significant resolution heuristics (search strategies), and paramodulation. The application of theorem-proving techniques to the problem of program correctness is also briefly discussed.

Keywords: automated theorem-proving, predicate calculus, program correctness, artificial intelligence

MR categories: 68A40, 02B10
CR categories: 5.21, 5.24, 3.6

Introduction

Proving theorems often requires, as all mathematicians will agree, a great deal of ingenuity and creativity. These two decidedly human traits have aroused the interest of many researchers in artificial intelligence, a field concerned with the mechanical implementation of "intelligent" tasks, and since the 1960's much effort has been expended on the problem of mechanical or automated theorem-proving. Although the fundamental concepts of mechanical theorem-proving are based on Herbrand's work in the first-order predicate calculus (1930), it was not until 1965 and the formulation of the resolution principle by J. A. Robinson [1], that the first major step was made toward achieving a computer implemented theorem-prover.

Almost all mechanical theorem-provers currently in operation are based on refinements of resolution, a single inference rule for first-order logic. The resolution principle is a partial decision rule. That is, given an arbitrary unsatisfiable formula in the first-order predicate calculus, a resolution prover will eventually halt and indicate unsatisfiability, however, if the formula is satisfiable, the process may not terminate. Since there is no general decision procedure to check the validity of a formula in first-order logic (a well-known result due to Church and Turing), partial decideability is the best that can be achieved.

This paper is a discussion of the concepts and techniques of automated theorem-proving. Section 1 contains a brief review of first-order logic and the definition of necessary terminology. Section 2 defines the special normal form for first-order wffs on which mechanical theorem-provers operate. The main result of Herbrand's work is presented in Section 3. Sections 4 and 5 are devoted to the resolution principle and refinements of it. Section 6

concerns the particular problems involved with expressing the equality relation and some proposed methods for dealing with them. Finally, Section 7 is a discussion of applications, primarily in the area of program correctness.

1. The First-Order Predicate Calculus

An expression in this logic system may consist of individual constants (a, b, c, ...) which have values in some domain D, individual variables (u, v, ...) whose values range over the same domain D, functional constants (f, g, ...) which represent functions of the form $D^n \rightarrow D$, and predicate symbols (P, Q, R, ...) which represent functions of the form $D^n \rightarrow$ {true (T), false (F)}. In addition, the logical quantifiers $\forall$ and $\exists$ may be used. These constants, variables, functional constants, predicates, and quantifiers, along with the logical constants T and F, are combined using $\wedge$, $\vee$, $\sim$, $\rightarrow$, $\equiv$ (conjunction, disjunction, negation, implication, and equivalence) to form well-formed formulas (wffs). The following definitions provide a more precise notion of a wff:

Def. 1 An expression is a term if and only if it is one of the following:

(1) a constant;
(2) a variable;
(3) an n-place function $f(t_1, t_2, \ldots t_n)$ where each argument t_i is a term.

Def. 2 An expression is an atom if and only if it is of the form $P(t_1, t_2, \ldots, t_n)$ where P is an n-place predicate symbol, and each argument t_i is a term. An atomic formula consists of atoms and the symbols T, F, $\wedge$, $\vee$, $\sim$, $\rightarrow$, $\equiv$, $\forall$, $\exists$.

Def. 3 An expression is a literal if and only if it is an atom or a negation of an atom.

Def. 4 An expression is a well-formed formula (wff) if and only if it is generated by a finite number of applications of the following rules:

(1) An atomic formula is a wff;

(2) If F and G are wffs, $\sim F$, $(F \wedge G)$, $(F \vee G)$, $(F \rightarrow G)$, and $(F \equiv G)$ are wffs;

(3) If F is a wff and x is an unquantified (free) variable in F, then $(\forall x)F$ and $(\exists x)F$ are wffs.

A wff W is given meaning by specifying a nonempty domain D, assigning an element of D to each constant and free variable in W, assigning a total mapping $D^n \rightarrow D$ to each n-place function symbol in W, and assigning a total mapping $D^n \rightarrow \{T,F\}$ to each n-place predicate symbol in W. This process constitutes an interpretation of W. A wff is said to be valid if and only if it is true under every interpretation over all possible domains, and unsatisfiable if and only if it is false under every interpretation over all possible domains. It is clear that a wff W is valid if and only if $\sim W$ is unsatisfiable.

In order to determine the validity of a wff, a deductive (inference) system is needed. Many deductive systems exist, and all have essentially two parts: (1) a set of valid expressions called axioms and (2) a finite set of rules of inference which are used to derive new valid expressions from old ones. A wff W is said to be deducible in an inference system if there exists a finite sequence of wffs such that W is the last wff in the sequence, and each wff is either an axiom in the system or is derived from previous wffs in the sequence using the rules of inference. Such a sequence constitutes a proof of W. Theorem-proving is concerned specifically with proving the validity of wffs of the form $A_1 \wedge A_2 \wedge \ldots A_n \rightarrow B$, where $A_1, A_2, \ldots, A_n$ represent the hypothesis and B represents the conclusion of the conjecture.

The resolution method is an inference system which proves theorems by refutation; ie., instead of proving a given wff is valid, it is shown that the negation of the wff is unsatisfiable. This means that instead of proving the validity of a conjecture $A_1 \wedge A_2 \wedge \ldots \wedge A_n \rightarrow B$, a resolution prover will attempt to prove the unsatisfiability of $\sim W$ or $A_1 \wedge A_2 \wedge .. \wedge A_n \wedge \sim B$. The system has only one rule of inference, and the system is complete; that is, a wff W is valid if and only if there exists a resolution refutation of $\sim W$. The property of completeness is extremely important in formulating deductive systems to be used in theorem-proving, since in an incomplete system there is no guarantee that a proof will be produced even if the theorem is indeed valid. The resolution method operates on wffs in a special normal form, introduced by Davis and Putnam [2], which involves no quantification and facilitates the manipulation of expressions. As will be shown in Section 2, any wff in the first-order predicate calculus can be expressed in this form (called clause form).

2. Clause Normal Form

The initial step in the transformation of a first-order wff into clause form is to express the wff in prenex conjunctive normal form.

Def. 5 A wff W in the first-order predicate calculus is in prenex conjunctive normal form if and only if W is expressed as

$$(Q_1 x_1)(Q_2 x_2)\ldots(Q_n x_n)M$$

where each Q_i $(1 \leq i \leq n)$ is either $\forall$ or $\exists$, and M is a quantifier-free conjunction of clauses (a clause is a disjunction of literals). The list of quantifiers is called the prefix of W, and M is called the matrix of W.

A detailed description of the rules for transforming an arbitrary first-order wff into prenex conjunctive normal form is given in [3]. (It will be assumed that all wffs are existentially closed; ie., that there are no free variables.) Essentially, the process involves successive replacement

of the wff by logically equivalent wffs to effectively:

(1) remove all $\rightarrow$ and $\equiv$ symbols;

(2) bring negation signs immediately in front of atoms;

(3) rename any variable that is quantified more than once;

(4) move all quantifiers to the left of the entire formula;

(5) distribute $\vee$ over $\wedge$.

Example 1

Transform $W = (\forall x)(\forall y)\Big((\exists z)(P(x,y) \vee Q(x,z) \rightarrow (\exists u)R(x,y,u)\Big)$ into prenex conjunctive normal form.

		Rule Applied
1.	$(\forall x)(\forall y)\Big(\sim((\exists z)(P(x,y) \vee (Q(x,z))) \vee (\exists u)R(x,y,u)\Big)$	(1)
2.	$(\forall x)(\forall y)\Big((\forall z)(\sim P(x,y) \wedge \sim Q(x,z)) \vee (\exists u)R(x,y,u)\Big)$	(2)
3.	$(\forall x)(\forall y)(\forall z)(\exists u)\Big((\sim P(x,y) \wedge \sim Q(x,z)) \vee R(x,y,u)\Big)$	(4)
4.	$(\forall x)(\forall y)(\forall z)(\exists u)\Big((\sim P(x,y) \vee R(x,y,u)) \wedge (\sim Q(x,z) \vee R(x,y,u))\Big)$	(5)

The next step in the transformation of a wff into clause form is the elimination of all existential quantifiers. Suppose $W = (Q_1x_1)(Q_2x_2)..$ $..(Q_nx_n)M$ is a first-order wff in prenex conjunctive normal form (each Q_i $(1 \leq i \leq n)$ is either $\forall$ or $\exists$), and let Q_r $(1 \leq r \leq n)$ be the leftmost occurrence of $\exists$. The elimination of (Q_rx_r) is accomplished according to one of two rules:

(1) if no $\forall$ appears to the left of Q_r, choose a constant c different from all constants appearing in M, replace all occurrences of x_r in M by c, and delete (Q_rx_r) from the prefix;

(2) if $Q'_1, Q'_2, \ldots Q'_m$ are all the $\forall$'s appearing before Q_r, $(1 \leq m < r)$, choose a new m-place function symbol f, replace all occurrences of x_r in M by $f(x'_1, x'_2, \ldots x'_m)$, and delete (Q_rx_r) from the prefix.

The appropriate rule is applied to each of the existential quantifiers (proceeding from left to right) until all are eliminated. The constants and functions used in the replacement are called Skolem functions, and the process is often referred to as Skolemization. This step completes the transformation, and the following theorem is an assurance that this form may be used in place of the original wff during the proof process:

Theorem 1 (Skolem) [cf. 3, p. 126] Every wff W can be transformed into a wff W' in clause form [using the procedure above] such that W is unsatisfiable if and only if W' is unsatisfiable.

All variables appearing in the clause form of a wff are universally quantified, hence throughout the proof process this universal quantification is assumed and all manipulations are performed only on the matrix (the $\forall$ symbols are dropped). The matrix is of the form

$$(L_{11} \vee L_{12} \vee \ldots \vee L_{1n_1}) \wedge (L_{21} \vee L_{22} \vee \ldots \vee L_{2n_2}) \wedge \ldots \wedge (L_{m1} \vee L_{m2} \vee \ldots \vee L_{mn_m})$$

where each L_{ij} is a literal. For notational purposes, this is expressed as $\{L_{11}, L_{12},.., L_{1n_1},\}, \{L_{21}, L_{22},.., L_{2n_2}\}, .. , \{L_{m1}, L_{m2},.., L_{mn_m}\}$. (Each $\{L_{i1}, L_{i2},..L_{in_i}\}$ is a clause.)

Example 2

Eliminate the existential quantifiers in

$W = (\exists x)(\forall y)(\forall z)(\exists u)(P(x,y,z,u)\wedge Q(x,u))$ using Skolem functions.

1. Eliminate $(\exists x)$ using rule (1): $(\forall y)(\forall z)(\exists u)(P(c,y,z,u)\wedge Q(c,u))$
2. Eliminate $(\exists u)$ using rule (2): $(\forall y)(\forall z)(P(c,y,z,f(x,z)\wedge Q(c,f(y,z))$

The clause form of W is $\{P(c,y,z,f(x,z))\}, \{Q(c,f(y,z))\}$.

The goal of a resolution theorem-prover is to prove that the negation of a theorem in clause form is unsatisfiable; ie., that it is false under all interpretations over all possible domains. This immense problem would be greatly simplified if a domain could be found with the property that an arbitrary wff W is unsatisfiable if and only if it is false under all interpretations over this domain. The discovery of such a domain, the Herbrand universe, constitutes the crux of Herbrand's contribution to automated theorem-proving.

3. Herbrand's Theorem

The Herbrand universe of a given first-order wff W, denoted H_W, is constructed as follows:

> Let H_0 be the set of constants in W. If W contains no constants, then set $H_0 = \{ a \}$ for an arbitrary constant a. For $i = 0, 1, \ldots$, let H_{i+1} be the union of H_i and the set of all terms of the form $f(t_1, t_2, \ldots, t_n)$, for all n-place functions f, such that $t_j \in H_i$. Then, $H = H_W$.

For example, the Herbrand universe of $P(a,f(x),g(y,z)) \vee Q(b)$ is $\{a,b,f(a), f(b), g(a,b), g(b,a), g(f(a),b)\ldots\}$. The essential characteristic of this domain is made clear in the following theorem:

Theorem 2 [4, p. 55] A wff W in clause form is unsatisfiable if and only if W is false under all interpretations over H_W.

The ability to restrict consideration to one domain is definitely an improvement, however, since the Herbrand universes of most wffs are infinitely large, a further result is necessary:

Def. 6 Let W be a wff in clause form (recall that a clause is a disjunction of literals). If C is a clause in W, and C' is a clause obtained by replacing all individual variables in C with members of H_W, then C' is called a ground instance of C.

Theorem 3 (Herbrand) [cf. 4, p. 61] A set S of clauses is unsatisfiable if and only if there is a finite unsatisfiable set S' of ground instances of clauses of S.

This result suggests the design of a process which continually generates ground instances and tests their conjunction for unsatisfiability, by applying rules of inference in an attempt to find a contradiction.

Such a process, called saturation, selects a sequence P_0, P_1, ... of finite subsets of the Herbrand universe of a wff W such that $P_j \subseteq P_{j+1}$ and $\bigcup_{j=o}^{\infty} P_j = H_W$. Then, $P_1(W)$, $P_2(W)$ ... are tested in turn, where $P_i(W)$ is the set of all ground instances obtained by substituting elements from P_i into W. This was the method used by Davis and Putnam [2], and although their work constitutes one of the most important early efforts, saturation is far too impractical for realistic implementation. The rules of inference that they devised for conjunctions of ground instances were generalized by Robinson to apply to any conjunction of clauses, not necessarily ground instances. This generalization is the basis of the resolution principle.

4. The Resolution Principle

Like the method employed by Davis and Putnam, resolution involves the substitution of terms for variables, however, these substitutions are chosen in a specific way. A substitution α will be denoted by a finite set of pairs $\{ \langle v_1,t_1\rangle, \langle v_2,t_2\rangle,\ldots,\langle v_n ,t_n\rangle\}$, where the v_i's are distinct variables and each t_i is a variable or element of the Herbrand universe different from v_i (the notation means that for each i, t_i is to be substituted for v_i). For any expression E, $E\alpha$ denotes the literal obtained by applying the substitution α to E, and is called an instance of E.

Def. 7 If $[E_1, E_2, \ldots E_n]$ is a set of expressions, then the substitution α is a unifier for the set if and only if $E_1\alpha = E_2\alpha = \ldots = E_n\alpha$. A set of expressions is said to be unifiable if there exists a unifier for it. A unifier β is a most general unifier if for each unifier α, $E_i\alpha$ is an instance of $E_i\beta$ for $1 \leq i \leq n$.

The resolution method restricts the use of substitutions to those which are most general unifiers. Given a unifiable set of expressions, the most general unifier can be determined by an algorithm which is

easily implemented mechanically, and which initially must locate the disagreement set of the expressions. The disagreement set of $[E_1, E_2, \ldots, E_n]$, denoted $[T_1, T_2, \ldots, T_n]$, is found by locating the leftmost symbol which is not common to every $E_i (1 \leq i \leq n)$ and setting T_i equal to the term in E_i which begins with this symbol. For example, the disagreement set of $[P(a,f(x,y),z), P(a,g(u),v)]$ is $[f(x,y), g(u)]$.

Unification Algorithm (Robinson) [1, p. 33]

Let S be a set of expressions and let ε denote the empty substitution.

(1) Set $k = 0$, $S_k = S$, $\sigma_k = \varepsilon$.

(2) If S_k contains one element, stop; σ_k is a most general unifier for S. Otherwise, find the disagreement set D_k of S_k.

(3) If there exist v_k and t_k in D_k such that v_k is a variable that does not occur in t_k, go to 4. Otherwise stop; S is not unifiable.

(4) Let $\sigma_{k+1} = \{\sigma_k, \langle v_k, t_k \rangle\}$ and $S_{k+1} = S\sigma_{k+1}$.

(5) Set $k = k+1$ and go to 2.

Example 3

Find a most general unifier for $S = [P(a,f(x),z) \wedge Q(y), P(u,v,b) \wedge Q(y)]$

$D_0 = [a,u]$;

$\sigma_1 = \{\langle u,a \rangle\}$; $S_1 = [P(a,f(x),z) \wedge Q(y), P(a,v,b) \wedge Q(y)]$

$D_1 = [f(x),v]$;

$\sigma_2 = \{\langle u,a \rangle, \langle v,f(x) \rangle\}$; $S_2 = [P(a,f(x),z) \wedge Q(y), P(a,f(x),b) \wedge Q(y)]$

$D_2 = [z,b]$;

$\sigma_3 = \{\langle u,a \rangle, \langle v,f(x) \rangle, \langle z,b \rangle\}$; $S_3 = [P(a,f(x),b) \wedge Q(y)]$.

S_3 contains only 1 element, hence the algorithm stops at step 2, and $\sigma_3 = \{\langle u,a \rangle, \langle v,f(x) \rangle, \langle z,b \rangle\}$ is the most general unifier for S.

Example 4

Find a most general unifier for S = [Q(a,b), Q(c,y)].

D_0 = [a,c] contains no variable, hence the algorithm will stop at step 3, indicating that S is not unifiable.

It is not difficult to see that the algorithm does indeed terminate in all cases, for, S contains only a finite number of distinct variables, and each time through the steps either the algorithm halts or the number of variables is decreased by 1. The algorithm will always halt at step 3 if no most general unifier has been found by the time all of the variables have been eliminated.

Unification is an essential element of the resolution process. The resolution prover accepts as input a conjunction of universally quantified clauses (which represent the negation of the theorem to be proved) and attempts to derive a contradiction from them (denoted □), which indicates unsatisfiability. Clearly, an expression such as $\sim Q(x) \wedge Q(x)$ is always false, hence from $\{\sim Q(x)\}$, $\{Q(x)\}$, □ could be logically deduced. In addition □ could be logically deduced from $\{\sim Q(x)\}$, $\{Q(a)\}$; for, if $\sim Q(x)$ is true for all x (universal quantification is assumed), then Q(a) can not possibly be true, which implies $\{\sim Q(x)\}$, $\{Q(a)\}$ is always false. Basically, a resolution prover attempts to find two clauses such that a literal in one clause is identical to the complement of a literal in another clause after unification. If this can be done, then a third clause (called a resolvent) is deduced from the original two (called parent clauses), and added to the set of clauses as a potential parent clause for further deductions. The proof is successful if, by this process, □ is deduced.

More precisely, let C_1 and C_2 be two clauses with no variables in common (variables may be renamed if necessary). If they contain literals

L_1 and L_2 respectively, such that L_1 and $\sim L_2$ have a most general unifier σ, then the resolvent $\{(C_1\sigma - L_1\sigma), (C_2\sigma - L_2\sigma)\}$ can be deduced from C_1 and C_2. For example, $\{P(a), Q(y,b)\}$ can be deduced from $\{P(x), R(x,b)\}$ and $\{Q(y,z), \sim R(a,z)\}$, using the unifier $\{ \langle x,a \rangle, \langle z,b \rangle \}$.

Robinson proved that this single inference rule is complete. Thus, if a given theorem is valid, there exists a refutation of the negation of the theorem which uses only the resolution principle as a rule of inference.

Example 5 In an associative system with left and right solutions, there is a right identity element. ($Q(x,y,z)$ represents $x \cdot y = z$)

C_1: $\{\sim Q(x,y,u), \sim Q(y,z,v), \sim Q(x,v,w), Q(u,z,w)\}$ } Associativity

C_2: $\{\sim Q(x,y,u), \sim Q(y,z,v), \sim Q(u,z,w), Q(x,v,w)\}$ }

C_3: $\{Q(f(x,y), x, y)\}$ left solution

C_4: $\{Q(x,h(x,y), y)\}$ right solution

C_5: $\{Q(x,y,k(x,y))\}$ closure

C_6: $\{\sim Q(\ell(x'), x', \ell(x'))\}$ negation of conclusion

Parent Clauses

C_1 & C_6 C_7: $\{\sim Q(x'',y'',\ell(x'), \sim Q(y'',x',v), \sim Q(x'',v,\ell(x')\}$

C_2 & C_7 C_8: $\{\sim Q(x,x',v), \sim Q(f(x,\ell(x')), v, \ell(f(x'))\}$

C_2 & C_8 C_9: $\{\sim Q(v,x',v)\}$

C_4 & C_9 C_{10}: $\square$

The property of completeness guarantees that given a valid theorem, there exists a resolution refutation of its negation. The mathematician, calling upon experience and intuition, is able to rule out many combinations of parent clauses as redundant or irrevelant to the proof, and is also able to easily determine which pairs have no unifiable literals. A computer, unless some of the concepts employed by the mathematician are part of the programmed proof procedure, randomly chooses combinations of clauses until a proof is completed, in the process generating many useless clauses which, even though they have no role in the actual proof, are potentially parent clauses for additional irrelevant resolvents. This "combinatorial explosion" causes the original resolution method to be highly impracticial for actual mechanical implementation given realistic time and space requirements. This has necessitated the development of several strategies to direct the computer's search for advantageous parent clauses, so that certain alternatives are deferred or omitted.

Such strategies are called heuristics. The most valuable heuristics are those which are complete (when used in conjunction with a deductive system), since otherwise it is possible eliminate all potential proofs by omitting certain alternatives. The next section presents some of the more important heuristics that have been incorporated into the resolution method.

5. Resolution Heuristics

Robinson suggested two strategies in his initial paper on resolution which he called the purity principle and subsumption [1]. Essentially, the purity principle states that if a clause C contains a literal L and no other clause contains a literal identical to $\sim L$ (after unification), then C may be eliminated from the list of clauses, since every resolvent

descended from C must contain an instance of L. ($\square$ will never be deduced using C.) Such a literal is said to be pure in the set of clauses. The purity principle is summarized in the following theorem:

Theorem 4 (Purity Theorem) [1, p. 37]

> If S is any finite set of clauses, and C is a clause in S which contains a literal L such that L is pure in S, then S is unsatisfiable if and only if S - {C} is unsatisfiable.

Subsumption is based on the concept that if a deduced clause is implied by another deduced clause, the former may be eliminated from the list of clauses to be used for inference. More precisely, a clause C subsumes a clause D if and only if there exists a substitution σ such that $C\sigma \subseteq D$. D is called a subsumed clause. Then, given a finite set of clauses S, if $D \in S$ is subsumed by another clause in S, D may be deleted from S.

Theorem 5 (Subsumption) [1, p. 38]

> 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 unsatisfiable if and only if S - {D} is unsatisfiable.

These two strategies do improve the efficiency of the resolution method, however, not to the degree necessary for a realistic implementation. Soon after the publication of Robinson's results, two more resolution heuristics, both shown to be complete, were developed by L. Wos, D. Carson, and G. Robinson. They are known as the unit preference strategy [5] and the set-of-support strategy [6].

The unit preference strategy attempts to resolve unit clauses (clauses of one literal) before any others and, if $\square$ is not derived, shorter clauses before longer ones. A detailed description of the search process,

as well as a proof of completeness, are presented in [5]. The rationale behind the method is that since the goal of the prover is to deduce a clause with no literals (□), an attempt should be made to generate clauses of steadily decreasing length. (If a unit clause is resolved against a clause with n literals, the resolvent clause will have no more than n-1 literals.) Most proofs will require some nonunit resolutions, however, this strategy has been effective in many cases.

The set-of-support strategy, although generalized in [6], is most often used in the following context. The goal of a resolution prover is to deduce a contradiction from a wff of the form $A_1 \wedge A_2 \wedge \ldots \wedge A_n \wedge \sim B$ where $A_1, \ldots, A_n$ represent the hypothesis of the conjecture and $\sim B$ is the negation of the conclusion. $A_1 \wedge A_2 \wedge \ldots \wedge A_n$ is, in most cases, a satisfiable wff. It seems reasonable, then, to avoid resolving two clauses from the set of clauses representing the hypothesis, since this will normally not lead to a contradiction. The strategy involves distinguishing the clauses which represent the negation of the conclusion from the rest of the clauses, and making no deductions which do not depend, directly or indirectly, on this distinguished set. The following definition is a more accurate description of the strategy:

Def. 7 If S is a set of clauses, then $S' \subseteq S$ is a set-of-support of S if S - S' is satisfiable. A set-of-support refutation is a resolution refutation in which there are no resolutions which involve two parent clauses from S - S'.

The proof in Example 5 is actually a set-of-support proof. Every resolution involves C_6 or a descendent of C_6, the clause which represents the negation of the conclusion.

The original resolution principle and all of the preceding heuristics have allowed inferences to be made from only two parent clauses. Robinson's hyper-resolution is a method which allows deductions to be made from sets

of clauses called clashes. [7 and 8]

Def. 8 A clash is a finite set of clauses $[C_1, C_2, \ldots, C_n, D]$ satisfying the following conditions:

(1) D contains at least $n > 0$ literals $L_1, L_2, \ldots L_n$;

(2) for each i, $1 \leq i \leq n$, C_i contains $\sim L_i$, but contains no complement of any other literal in D, nor the complement of any literal which occurs in C_j, $j \neq i$, $1 \leq j \leq n$.

The resolvent of a clash in the above form is $\{C_1-(\sim L_1), C_2-(\sim L_2), \ldots C_n-(\sim L_n), D-\{L_1, L_2, \ldots, L_n\}\}$.

Example 6

C_1: $\{Q(x)\}$

C_2: $\{R(y)\}$

C_3: $\{\sim Q(x'), \sim R(y'), S(x',y')\}$

C_4: $\{\sim S(u,v)\}$

The resolvent of the clash $[C_1, C_2, C_4, C_3]$ is $\square$.

The resolution of a clash $[C_1, C_2, \ldots, C_n, D]$ where the C_i's contain only positive literals is called a hyper-resolution. Robinson proved that an inference system using only hyper-resolution is complete [7]. Thus, there exists a proof in which every resolution involves at least one positive clause. This imposes additional selection criteria on candidates for

resolution, and helps to control the "combinatorial explosion" problem. Meltzer suggested that by renaming predicates, the size of the set of positive clauses could be reduced, further limiting the number of potential parent clauses [9].

The set-of-support strategy, Robinson's hyper-resolution, and Meltzer's renamable resolution were all shown by Slagle to be included in his semantic resolution [10]. As in the set-of-support strategy, the clauses are divided into two groups and no resolutions are made within the same group, however, in semantic resolution, an interpretation is used to divide the clauses. In addition, an ordering is imposed on the predicate symbols.

Def. 9 [4, p. 104]

> Let I be an interpretation and let P be an ordering of predicate symbols. A finite set of clauses $[E_1, E_2, ..E_n, N]$, $n \geq 1$, is called a semantic clash with respect to P and I (PI-clash) if and only if $E_1, E_2, ...E_n$ (called electrons) and N (called the nucleus) satisfy:
>
> (1) $E_1, E_2, ..., E_n$ are false in I;
>
> (2) Let $R_1 = N$. For each $i = 1, ..., n$, there is a resolvent R_{i+1} of R_i and E_i;
>
> (3) The literal which is resolved upon in E_i contains the largest predicate in E_i for $1 \leq i \leq n$;
>
> (4) R_{n+1} is false in I.
>
> R_{n+1} is called a PI-resolvent of the PI-clash.

Slagle proved that PI-resolution is complete [10]; ie., given a set of unsatisfiable clauses S, an ordering P of predicate symbols, and an interpretation I of S, there is a deduction of □ from S in which each clause is either in the original set or is a PI-resolvent of the preceding clauses. In addition, Slagle showed the equivalence of renamable resolution and semantic resolution.

The set-of-support strategy and hyper-resolution are actually special cases of semantic resolution. If there exists a set-of-support T of a set S (S - T is satisfiable), then there must exist an interpretation I that satisfies S - T. Under any ordering P, there exists a PI-deduction of □ corresponding to the set-of-support proof. Hyper-resolution involves a specific kind of interpretation I, one in which every literal in I is the negation of an atom. Thus, every electron and every PI-resolvent (which all must be false in I) contain only positive literals.

Example 7

C_1: {P(a,b), Q(c)}

C_2: {R(u), Q(v)}

C_3: {∿P(x,y), ∿R(b), Q(c)}

I = {∿P(a,b), ∿R(b), ∿Q(c)}; P: P > R > Q.

Then, [C_1, C_2, C_3] is a PI-clash, and Q(c) is the PI-resolvent.

The idea of ordering predicate symbols spawned the concept of the ordered clause (an ordered clause is considered as a sequence of literals rather than a set of literals) and ordered resolution, a strategy similar to PI-resolution with further limitations placed on parent clauses [11]. Lock resolution is another technique based on an ordering of symbols, which uses indices to order the literals in clauses [12].

Loveland [13] and Luckham [14] independently developed linear strategies for the resolution method which initially resolve two clauses, then resolve the result against another clause, and resolve this result against another clause, etc., keeping the proof in a linear format. They proved that all (valid) theorems have resolution refutations which can be arranged

in a straight line, such that every clause is either in the original set of clauses or can be partially deduced from its predecessor.

Many other heuristics have been developed for the resolution principle, but although these and the specific methods described above have proved helpful in many cases, contemporary theorem-provers are far from efficient. One proposed solution to the efficiency problem is the interactive or man-machine prover. In this type of a system, the user is allowed to interact with the theorem-prover by offering suggestions or hints when necessary. A research group at the University of Texas at Austin has been quite successful using an interactive system based on a "natural deduction" prover rather than a resolution prover [15 and 16]. This system has been used to prove theorems in set theory and topology, and limit theorems in calculus and analysis. Interactive systems based on resolution provers are also in operation currently [17]. Unless a major breakthrough occurs in the technology of mechanical theorem-proving, the interactive type of system seems to be the most promising in terms of efficient implementation.

The next section deals with the problem of expressing an essential element of Mathematics, the equality relation, in a form which can be easily manipulated by a mechanical theorem-prover.

6. The Equality Relation

The equality relation is not easily expressible in the first-order predicate calculus. The three axioms of reflexivity, symmetry, and transitivity, in addition to an indefinite number of axioms expressing functional and predicate reflexivity (in some cases as many as one for each function and predicate symbol) must be represented. As in the group theoretic proof of Example 5, many early theorem-provers avoided using equality

whenever possible.

Three main methods have been used to deal with the equality relation:

(1) Employing a set of first-order axioms for equality;

(2) Employing a smaller set of second-order axioms for equality;

(3) Employing a substitution rule for equals as a rule of inference.

The first method, as observed, is highly impractical. The second method, the use of second-order logic, has many advantages, however, the fact that no second-order inference system is complete (a result due to Gödel), in addition to its high degree of complexity, has discouraged its use in mechanical theorem-proving. Second-order logic allows quantification over predicate and function symbols, and greatly simplifies the representation of many mathematical concepts. For example, equality can be expressed as $[x=y] \equiv [(\forall P)(P(x) = P(y)) \wedge (\forall f)(f(x) = f(y))]$, and induction (which requires an infinite set of axioms in first-order logic) as $(\forall P)[P(0) \wedge (\forall x)(P(x) \rightarrow P(x+1)) \rightarrow (\forall x)P(x)]$. [16, p. 128]

Another reason that few attempts have been made to use second-order logic is the inadequacy of the resolution principle as an inference rule in this logic system. Darlington attempted to extend the matching process of unification to second-order logic by allowing substitutions to be made for predicate and functional variables, however, his method offers an improvement in only a small number a cases [17].

Paramodulation, an inference rule developed by G. Robinson and L. Wos, operates in a resolution system, and is an example of the third method [18].

Def. 10 Let C_1 and C_2 be two clauses (with variables renamed if necessary). If C_1 is of the form $\{L[t], C_1'\}$, where $L[t]$ is a literal L containing the term t, and C_2 is of the form $\{r=s, C_2'\}$, and if t and r have a most general unifier σ, then the clause $\{L\sigma[s\sigma], C_1'\sigma, C_2'\sigma\}$ may be inferred. This clause is called a paramodulant.

Superficially, one might describe paramodulation as a "substitution rule for equality", however, an essential element of this inference rule is the maximum generality it provides. In theorem-proving, generality refers to choosing to infer a clause C rather than a proper instance of C, when either could be inferred logically. Completeness for paramodulation in combination with resolution has been proved only for sets of clauses which include the axioms for functional reflexivity, but it is conjectured that the system is indeed complete without them.

Example 8 [15, p. 140]

Prove that if $x^2 = e$ for all x in a group, the group is abelian. (f(xy) represents $x \cdot y$)

	C_1: f(ex) = x	(left identity)
	C_2: f(xe) = x	(right identity)
	C_3: f(xf(yz)) = f(f(xy)z)	(associativity)
	C_4: f (xx) = e	
negation of conclusion	C_5: f(ab) = c	
negation of conclusion	C_6: $c \neq f(ba)$	
	C_7: f(xe) = f(f(xy)y)	C_4 into C_3
	C_8: x = f(f(xy)y)	C_2 into C_7 on f(xe)
	C_9: a = f(cb)	C_5 into C_8 on f (xy)
	C_{10}: f(yf(yz)) = f(ez)	C_4 into C_3 on f(xy)
	C_{11}: f(yf(yz)) = z	C_1 into C_{10} on f(ez)
	C_{12}: f(ca) = b	C_9 into C_{11} on f(yz)
	C_{13}: c = f(ba)	C_{12} into C_8 on f(xy)
	C_{14}: □	C_{13} resolved with C_6

7. Applications

As a result of the limited efficiency of current systems, the contribution of mechanical theorem-proving to Mathematics has thus far been minimal. (Although some useful counterexamples have been generated, only known results have been proved.) However, the techniques and concepts have been applied to several problems in artificial intelligence, including deductive question answering and problem solving [4], and more recently program synthesis and program verification.

Because of the increasing costs of discovering programming errors at the consumer level and the inability of traditional debugging techniques to guarantee the reliability of today's highly complex software, much interest has developed in the area of program correctness. Several methods have been proposed for constructing a correct program from a specified description; this process is called program synthesis. Dijkstra has suggested that simply by requiring a programmer to make explicit (within comments) the reasoning involved in writing an algorithm to meet desired specifications, reliability will be improved [21]. Floyd has proposed an interactive design procedure which uses a computer to check the program as it is being written for consistency with the specifications [22]. Manna and Waldinger have developed a method which involves expressing the program specifications in first-order logic, proving a theorem induced by the specifications, and extracting the desired program from the proof [23].

Program verification involves verifying that an already existing program satisfies certain specifications. It is required that the programmer provide a description of input restrictions and the desired relation between variables upon completion of execution. The program is said to be partially correct if it can be shown that for any input satisfying the input restrictions, the output relation is met whenever the program terminates; the program is said to be correct if for any input satisfying

the input restrictions, the program terminates and yields the desired result.

The assertion method, developed independently by Floyd [24] and Naur [25] is the only known method for constructing proofs about programs. It involves providing assertions about the state of the program variables after selected steps in the program. The verification procedure attempts to prove that each assertion A_i is implied by the conjunction of A_{i-1} and the transformation of the variables effected by the step in the program corresponding to A_i ($1 \leq i \leq n$ where A_0 represents the input restrictions and A_n represents the desired output relation). Manna has shown that this type of a program description corresponds to a first-order wff, and that the proof of correctness is equivalent to a proof of the unsatisfiability of the negation of this wff [26]. Detailed descriptions of the applications of theorem-proving techniques to the problem of program verification can be found in [3] and [4], while an excellent summary is presented in [19].

The problem of program correctness is becoming increasingly crucial, however, progress is severely hampered by the inadequacy of contemporary theorem-provers. Much work is needed in the development of more efficient heuristics and in the investigation of the applicability of other logic systems. Although substantial progress has been made in the development and actual implementation of mechanical theorem-proving techniques, a great deal more is necessary before the full potential of mechanical theorem-proving is realized.

Deborah Young Macock
Language Research Center
Dept. of Computer Science
Virginia Polytechnic Institute and
State University
Blacksburg, VA 24061

Bibliography

1. J. A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle", JACM 12: 23-41, 1965.

2. Martin Davis and Hilary Putnam, "A Computing Procedure for Quantification Theory", JACM 7: 201-215, 1960.

3. Zohar Manna, Mathematical Theory of Computation, McGraw-Hill Book Company, 1974.

4. C. L. Chang and R.C.T. Lee, Symbolic Logic and Mechanical Theorem Proving, Academic Press, 1974.

5. L. Wos, D. Carson, and G. Robinson, "The unit preference strategy in theorem proving", Proc. AFIPS 1964 FJCC: 615-621, 1964.

6. L. Wos, G. Robinson, and D. Carson, "Efficiency and Completeness of the Set-of-Support Strategy in Theorem Proving", JACM 12: 536-541, 1965.

7. J. A. Robinson, "Automatic Deduction with Hyper-Resolution", Int. J. of Computer Math. 1: 227-234, 1965.

8. J. A. Robinson, "A review of automatic theorem proving", Proc. Symp. in Applied Math. 19 (AMS):1-18, 1967.

9. B. Meltzer, "Theorem-proving for Computers: Some results on resolution and renaming", Computer Journal 8: 341-343, 1966.

10. James R. Slagle, "Automatic Theorem Proving with Renamable and Semantic Resolution", JACM 14: 687-697, 1967.

11. R. Reiter, "Two results on ordering for resolution with merging and linear format", JACM 18: 630-646, 1971.

12. R. S. Boyer, Locking: a restriction of resolution, Ph.D. Thesis, Univ. of Texas at Austin, 1971.

13. D. W. Loveland, "A Unifying View of Some Linear Herbrand Procedures", JACM 19: 366-384, 1972.

14. David Luckham, "Some Tree-paring Strategies for Theorem Proving", Machine Intelligence 3: 95-112, 1968.

15. W. W. Bledsoe and Peter Bruell, "A Man-Machine Theorem-Proving System", Artificial Intelligence 5: 51-72, 1974.

16. W. W. Bledsoe and Mabry Tyson, The UT Interactive Prover , Univ. of Texas Math. Dept. Memo ATP 17, 1975.

17. S. Igarashi, R. L. London, and D. C. Luckham, Automatic Program Verification I: Logical Basis and its Implementation , Computer Science Report 365, Stanford University, 1973.

18. G. Robinson and L. Wos, "Paramodulation and Theorem-proving in First-Order Theories with Equality", Machine Intelligence 4: 135-150, 1969.

19. B. Elspas, L. N. Levitt, R. J. Waldinger, and A. Waksman, "An Assessment of Techniques for Proving Program Correctness", ACM Computing Surveys 4,2: 97-147, 1972.

20. J. L. Darlington, "Automatic Theorem Proving with Equality Substitution and Mathematical Induction", Machine Intelligence 3: 113-127, 1968.

21. E. W. Dijkstra, "A Constructive Approach to the Problem of Program Correctness", BIT 8: 174-186, 1968.

22. R. W. Floyd, "Toward Interactive Design of Correct Programs", Proc. IFIP: 7-10, 1971.

23. Zohar Manna and Richard J. Waldinger, "Toward Automatic Program Synthesis", CACM 14,3: 151-165, 1971.

24. R. W. Floyd, "Assigning Meanings to Programs", Proc. Symp. Applied Math. 19 (AMS): 19-32, 1967.

25. Peter Naur, "Proof of algorithms by general snapshots", BIT 6: 310-316, 1966.

26. Zohar Manna, "The Correctness of Programs", Journal of Computer and System Sciences 3: 119-127, 1969.