A Proof Procedure Using Connection Graphs

ROBERT KOWALSKI

University of Edinburgh, Edinburgh, Scotland

ABSTRACT. Various deficiencies of resolution systems are investigated and a new theorem-proving system designed to remedy those deficiencies is presented The system is notable for eliminating redundancies present in SL-resolution, for incorporating preprocessing procedures, for liberalizing the order in which subgoals can be activated, for incorporating multidirectional searches, and for giving immediate access to pairs of clauses which resolve Examples of how the new system copes with the deficiencies of other theorem-proving systems are chosen from the areas of predicate logic programming and language parsing. The paper emphasizes the historical development of the new system, beginning as a supplement to SL-resolution in the form of classification trees and incorporating an analogue of the Waltz algorithm for picture interpretation The paper ends with a discussion of the opportunities for using look-ahead to guide the search for proofs

KEY WORDS AND PHRASES: theorem-proving, logic, programming, connection graph

CR CATEGORIES: 5 21

Introduction

Comparison of proof procedures is not an easy task. Mathematical analysis of efficiency is rarely attempted and experimentation with actual programs is generally inconclusive, when not misleading.

In [21] we presented arguments in support of SL-resolution. In this paper we introduce a new proof procedure, and argue that it is superior to SL-resolution.

The paper begins with an informal definition of SL-resolution and with an analysis of its deficiencies. The new proof procedure is introduced by describing its evolution from the classification trees used in SL-resolution and from the filtering algorithm used in Waltz's picture interpretation program. We argue that the new proof procedure using connection graphs solves the problems associated with the deficiencies of SL-resolution.

In this paper we concentrate on those arguments for the connection graph proof procedure which are based on the comparison with SL-resolution Elsewhere [20] we advance the arguments based on the comparison with Bledsoe's theorem-provers [2–4] and with interpreters for high level programming languages like PLANNER [13].

We have not concerned ourselves in this paper with investigations of the completeness of the new proof procedure. Such investigations have been pursued by Frank Brown.

This paper is a slightly revised version of an earlier memorandum [18]. In the previous version, connection graphs were called classification graphs because of their evolution from the classification trees used in SL-resolution.

Based on the earlier memorandum, the connection graph proof procedure has been implemented by several researchers. Particularly noteworthy are the implementations of

This research was supported by a Science Research Council grant to Bernard Meltzer.

Author's present address: Department of Computing and Control, Imperial College of Science and Technology, 48 Prince's Gardens, London SW7 1LU, England.

Tor Amble, University of Trondheim, Norway, applying the structure sharing techniques of Boyer and Moore [5] and of Gianesini, University of Aix-Marseille, France, written in predicate logic and interpreted by PROLOG [8], a theorem-prover related to SL-resolution.

More recently David Warren has convinced us that the connection graph proof procedure can usefully be regarded as an elaboration of the cancellation system developed by Colmerauer [7] and described in this paper.

SL-Resolution

This section gives an informal description of SL-resolution. The description is incomplete and also imprecise in certain respects. Our objective is to describe SL-resolution in sufficient detail for the later investigations of its deficiencies. The example which follows the description should help to clarify the meaning of such notions as *most recent*, *selected*, *active*, and *passive*, which are mentioned, but not defined, below. A precise definition of SL-resolution, if necessary, can be found in [21]. We assume that the reader is familiar with the notions of resolution and factoring. Nilsson's book [27] and Robinson's review paper [32] provide adequate explanations of these concepts. The following description is less accurate for the general case than it is for the ground case where clauses contain no variables. This is sufficient for our purposes since all of the deficiencies of SL-resolution which we shall investigate apply to the ground case.

For a sequence $C_1, \cdots, C_n$ of clauses to be an *SL-derivation* from a set of clauses S, it is *necessary* that

1. $C_1 \in S$ (called the *top clause* of the derivation),
2. C_{i+1} is obtained from C_i by resolving on a selected, most recent literal in C_i with a clause B, where (a) $B \in S$ or (b) B is an *ancestor* C_j $(j < i)$ of C_i.

Each clause in an SL-derivation can be regarded as a set of goals: one goal, for each literal, of refuting the literal by establishing its negation. The literals in each clause are arranged in a stack and the achievement of the individual goals located in the stack is attempted on a last-in-first-out basis. Associated with every input clause B in S and with every literal L in B is an *operator* (or rule of inference) which, applied to the goal of refuting $\bar{L}$, replaces that goal by the subgoals $B - \{L\}$. Application of such an operator is an instance of case 2(a) in the description of SL-derivation.

There is a simple notational device for encoding into each clause information about which of its subgoals are currently active, which subgoals are passively awaiting later activation, and what are the hierarchical relationships between various goals and subgoals. The same notation makes it easy to recognize and to reject clauses which contain contradictory subgoals, to suppress loops caused by activating a goal as a subgoal of itself, to merge distinct passive occurrences of a single subgoal, and to use proof by contradiction for the achievement of subgoals which are contradicted by higher level goals. Case 2(b) in the description of SL-derivation deals with the achievement of subgoals by contradiction and is called *ancestor resolution*. The notation which facilitates these operations is illustrated and described in Figure 1.

Example. In Figure 1 the initial set of goals C_1 is to refute both L and K. Both goals have equal priority. We select one, K, to solve before the other. But B_1 can be read as saying that K is solved if M is. Regarding B_1 as an operator and applying it to the stack of subgoals C_1, we obtain (by case 2(a)) the resolvent of B_1 and C_1 which is a new stack of subgoals C_2. The subgoal K changes status from passive to active and the new subgoal M is added to the top of the stack. Of the two subgoals L and M in C_2, only M is most recent and candidate for selection C_3 is obtained by applying to C_2 the operator $\bar{M}L$, which replaces the subgoal M by L. Now both K and M are active, with M as a subgoal of K. There are two passive occurrences of L. These occurrences are *merged* together. The earlier occurrence of the subgoal dominates and the most recent occurrence of L is re-

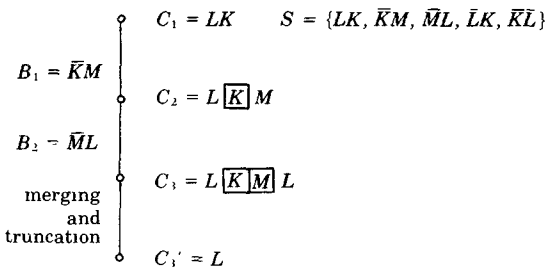

FIG. 1. Notational conventions

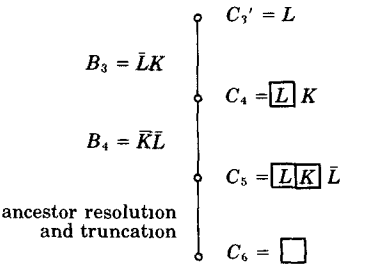

FIG. 2 Continuation of Figure 1

In our notation, the commas and curly brackets associated with the set notation for clauses are omitted. Thus LK stands for $\{L, K\}$. Stacks of literals are written with more recent literals following, i.e. to the right of, less recent literals. The selected literal resolved upon in a clause C_i is retained in the resolvent C_{i+1}, but is enclosed in a box to indicate that it has been resolved upon Such sequences of literals are called *chains* in [21]. The boxed literals in a chain stand for subgoals actively pursued in the chain. One active subgoal is a *subgoal* of another active subgoal in a chain if the first occurs after, i e. to the right of, the second As a *terminological convention*, we give the *name* L to the *subgoal* of refuting L Thus we say that the clause LK contains two subgoals L and K, which consist of refuting L and K, respectively

garded as achieved, conditionally, upon the later achievement of L. Both subgoals K and M are now conditionally achieved subject to the later achievement of the first occurrence of subgoal L. C_3' is obtained from C_3 by merging the two occurrences of L and by *truncation*, a bookkeeping operation which recognizes and removes accomplished subgoals from the top of the stack.

In Figure 2, C_4 is obtained by replacing the goal L by the subgoal K. C_5 is obtained by replacing goal K by subgoal $\bar{L}$. But now $\bar{L}$ is a subgoal of L. Proof by contradiction allows us to assume that L is true during the course of trying to refute L. But then, by assuming L, we achieve the subgoal $\bar{L}$ of refuting $\bar{L}$.

The definition of SL-derivation determines, for a given set of input clauses, for a given top clause, and for a given criterion for selecting most recent subgoals, a *search space* of all possible SL-derivations. Figure 3 illustrates the entire search space of all SL-derivations determined by the choice of S and C_1 in the preceding example. The space is arranged in the form of a tree whose derivations share common initial subderivations. In this example, terminal nodes not labeled by the mark of success, $\square$, violate the *admissibility* restriction, which rejects a chain whenever it contains distinct occurrences of the same atomic formula, unless merging or ancestor resolution can be performed and the literal removed by the operation has just been entered at the top of the stack.

The definition of SL-resolution in no way prejudices the order in which derivations can be generated by a search strategy. Breadth-first, depth-first, and various ordering strategies guided by merit orderings or by evaluation functions can be used to sequence the generation of derivations in the search space. The search strategy can reach its decisions autonomously as a result of its own deliberations or it can execute search strategic advice formulated by a user in a language especially provided for that purpose. This paper is concerned primarily with search spaces, although aspects of search strategy are dealt with in the last section.

Deficiencies of SL-Resolution

PROBLEM 1. CONTRADICTORY UNITS. Figure 3 illustrates what is possibly the most obvious defect of SL-resolution. that contradictory unit clauses, K and $\bar{K}$, can appear on different branches of the search space, while the generation of an explicit contradiction is prohibited by the restriction that clauses resolve only with axioms or ancestors.

This problem has an obvious and easy solution: Supplement SL-resolution with a procedure which checks, whenever a unit clause is generated, whether it contradicts some

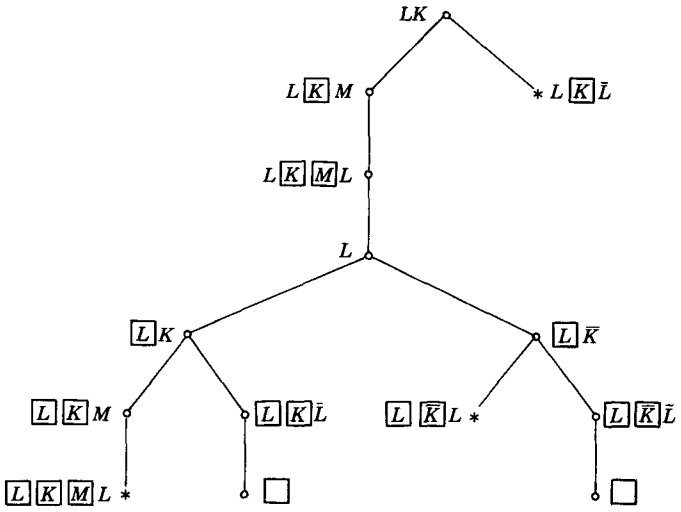

FIG 3. The search space for the example of Figures 1 and 2. The selection function activates in the top clause LK, the subgoal K before L. Nodes denoted by an asterisk violate the admissibility restriction, because their labeling chains contain distinct occurrences of the same atomic formula L and neither merging nor ancestor resolution is possible.

previously generated unit. This solution has the advantage of solving the problem in the least demanding manner, bringing immediate short-term benefits. However it is an ad hoc solution which resolves none of the many other related problems. It shares with other ad hoc solutions the danger of camouflaging the original problem and of inhibiting its satisfactory solution.

PROBLEM 2. REDUNDANCY. More worrisome and less obvious than the problem of contradictory units is the more general problem of redundancy in SL-search spaces. SL-resolution eliminates the $n!$ redundancy present in systems like set-of-support and linear resolution which, without a selection restriction, consider all the $n!$ ways of sequencing for solution the n subgoals associated with n literals in a clause. However, other redundancies can be generated by SL-resolution when more than one operator applies to a subgoal and when more than one is necessary for its solution. The occurrence of contradictory units in Figure 3 is a consequence of this redundancy. Figure 4 illustrates the same redundancy in a simplified example.

Considered in isolation from the many other problems of SL-resolution, redundancy is an easy problem admitting a variety of solutions. One, possibly the most straightforward, solution involves ordering operators when more than one applies to the same subgoal. In Figure 4, for example, order B_1 before B_2 when both are used as operators which apply to subgoals of the form L. Between the activation and eventual achievement of a subgoal, operators may occur only in increasing order. Thus the second branch in Figure 4, labeled by operator B_2 followed by B_1, violates this restriction. Unfortunately, in this example, the solution to problem 2 does not solve the associated problem 1.

A second solution attributes the problem to the loss of information which occurs when sentences are converted into clausal form. The operators $\bar{L}K$ and $\bar{L}\bar{K}$ share the literal $\bar{L}$. The two operators might usefully be replaced by the single sentence $\bar{L}(K \,\&\, \bar{K})$ and resolution of clauses might usefully be extended to resolution of such sentences. This approach merits serious consideration if only because it is a popular objection against clausal form that it destroys useful information implicit in nonclausal formulations of problems. The subject of clausal form, like that of the unnaturalness of resolution, is a controversial one. Here we note only that our solution to the problem of redundancy in SL-resolution retains the use of clausal form. A well-reasoned argument for liberalizing clausal form and

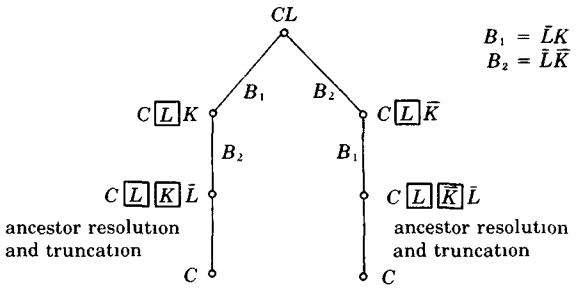

FIG 4. Redundancy in SL-resolution

an extension of SL-resolution to nonclausal sentences has been made by Wilkins [34]. In [20] we dealt in detail with the arguments for and against the naturalness of both clausal form and resolution

PROBLEM 3. MORE FLEXIBLE ACTIVATION OF SUBGOALS The first two problems concern the most obvious faults of SL-resolution. Although one can argue about the respective merits of alternative solutions, it is more difficult to argue that these problems are not worth serious consideration The situation is different for the remaining problems which we wish to consider. These problems concern not so much identifiable inefficiencies as much as they concern limitations which restrict problem-solving capabilities. The restriction, in SL-resolution, to the activation and achievement of subgoals in a last-in-first-out manner is such a problem.

In the propositional case, without ancestor resolution, SL-resolution search spaces can be regarded as search trees obtained from and-or trees by imposing a depth-first search on and-branches. Although in many cases such a strategy is desirable, in other cases a more flexible rule is useful. The ability to attempt the achievement of several subgoals simultaneously is especially important in the general case when subgoals are not independent. The reduction of problems to dependent subproblems and the simultaneous consideration of several subproblems are topics dealt with elsewhere [20].

Example. Clauses (1)–(11) define a nondeterministic predicate logic program for sorting lists whose elements are among the numbers 1, 2, 3. Clause (1) asserts that y is a sorted version of x if y is a permutation of x and y is ordered. Clauses (2) and (3) recursively define permutation in terms of deletion Del(x, y, z) states that z is obtained by deleting x from y and is defined by (4) and (5). Clauses (6), (7), and (8) define orderedness. Clauses (9), (10), and (11) list parts of the "less than" relation necessary for sorting lists whose elements contain just the numbers 1, 2, or 3.

(1) $S(x, y)\ \bar{P}(x, y)\ \bar{O}(y)$,
(2) $P(\text{nil}, \text{nil})$,
(3) $P(z, \text{cons}(x, y))\ \overline{\text{Del}}(x, z, z')\ \bar{P}(z', y)$,
(4) $\text{Del}(x, \text{cons}(x, y), y)$,
(5) $\text{Del}(x, \text{cons}(y, z), \text{cons}(y, z'))\ \overline{\text{Del}}(x, z, z')$,
(6) $\bar{O}(\text{nil})$,
(7) $O(\text{cons}(x, \text{nil}))$,
(8) $O(\text{cons}(x, \text{cons}(y, z)))\ \overline{LE}(x, y)\ \bar{O}(\text{cons}(y, z))$,
(9) $LE(1, 2)$,
(10) $LE(1, 3)$,
(11) $LE(2, 3)$.

We choose this example not because it is an especially good example of programming in predicate logic, but because it provides an illustration of the utility of more flexibly activating subgoals than is allowed by SL-resolution. Figure 5 illustrates part of the SL-search space determined by the problem of sorting the list cons(2, cons(1, cons(3, nil))). Execution of the program (1)–(11) is initiated by the addition of the top clause $\bar{S}(\text{cons}(2,$

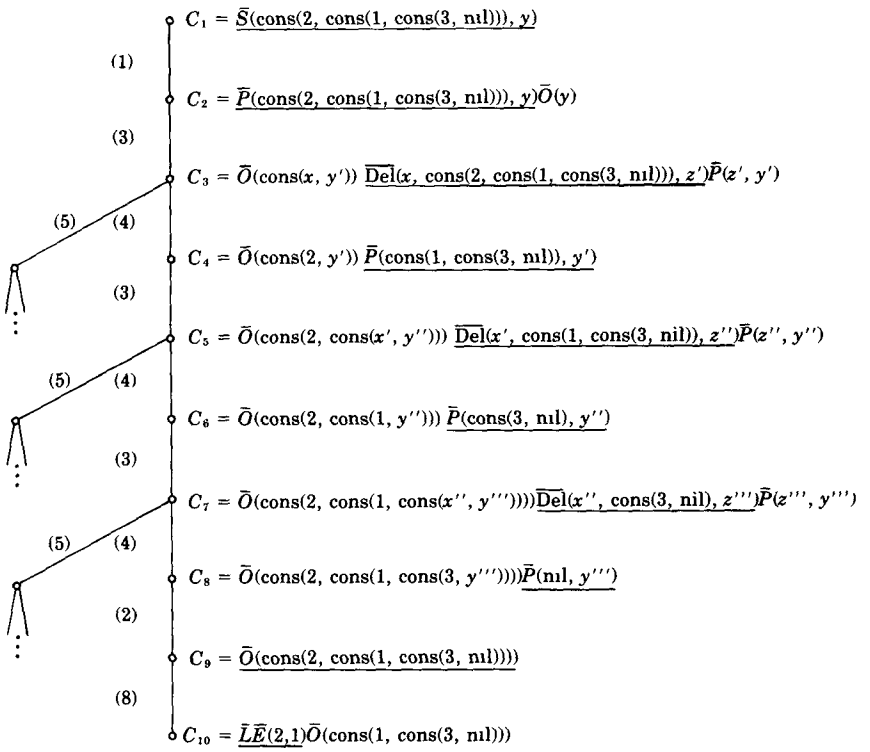

FIG. 5. Part of the search space for the sorting problem. Selected literals are underlined. Active subgoals are not illustrated for lack of space.

cons(1, cons(3, nil))), y), which denies that the list can be sorted. This contradicts what is implicit in (1)–(11) and provokes the theorem-prover into deriving a contradiction which is explicit. This example is also considered in [19]. Van Emden [9] compares this representation of the sorting problem with another representation corresponding to the Quicksort algorithm.

For this set of clauses, and for an appropriate selection function, SL-resolution executes the following algorithm for sorting lists: Generate permutations of the original list and test each permutation for orderedness. A permutation which survives the test is a solution. Since we are not concerned here with search strategy, the algorithm is nondeterministic in the sense that it does not specify whether permutations should be generated in parallel, by backtracking or by some other sequencing strategy.

For the same set of clauses, more flexible selection of subgoals results in the following more efficient, nondeterministic algorithm: Generate sublists cons(x, y) of the original list and test each such sublist for orderedness, assuming that the tail, y, is ordered and checking only that the head, x, is less than the first element in y. If the sublist is ordered, then add a new element to the front of the list and test that orderedness is preserved; otherwise reject the sublist and consider no extensions of it.

The second algorithm is obtained from the first by activating subgoals of the form $\bar{O}(\text{cons}(x, \text{cons}(y, z)))$ as soon as it can be tested whether $\overline{LE}(x, y)$. Thus the unsuccessful derivation illustrated in Figure 5 could be terminated earlier by activating and recognizing the unsolvability of the passive subgoal $\bar{O}(\text{cons}(2, \text{cons}(1, y'')))$. For SL-resolution, activation of this subgoal is impossible because it is not most recent.

Considered in isolation from the other problems of SL-resolution, it is not difficult to find systems which liberalize the order in which subgoals can be activated and achieved.

What is more difficult is to refrain from constructing locally optimal solutions to individual subproblems and to work instead toward the construction of globally optimal solutions to the many problems which need eventually to be solved.

PROBLEM 4. SUPPORT SETS AND MULTIDIRECTIONAL SEARCH. Suppose that a set of clauses S can be partitioned into two sets, one containing the axioms and the hypotheses of the theorem to be proved and the other containing a single clause which is the negation of the conclusion of the theorem. If the axioms and hypotheses are consistent and if the conclusion of the theorem is a logical consequence of the axioms and hypotheses, then there exists an SL-refutation of S with a top clause which is the negation of the conclusion of the theorem. Similarly, if the axioms of S are consistent and if all of the hypotheses of the theorem are necessary to establish the conclusion, then if S is unsatisfiable, for every clause which is a hypothesis of the theorem there exists an SL-refutation of S having that hypothesis as top clause. More generally, it suffices for completeness to generate only SL-derivations whose top clause belongs to some subset S' of the input set of clauses. What is required is that the subset, called *set-of-support*, contains at least one clause necessary for a refutation, i e. that $S - S'$ be satisfiable [37].

In general there exist several alternative support sets for a given set of input clauses. In SL-resolution it is possible to choose and employ just a single set-of-support. We need a proof procedure which, like those employed by Bledsoe and his colleagues [2–4], allows us to tackle a problem simultaneously from several points of view: chaining backward from the conclusion of the theorem and forward from the hypotheses.

Example. $S = \{P(0), \bar{P}(x)P(s(x)), \bar{P}(s(s(s(0))))\}$. It is easy to show that, on purely syntactic grounds, both $P(0)$ and $\bar{P}(s(s(s(0))))$ must occur in any refutation of S (because every unsatisfiable set of clauses must contain at least one positive clause, containing no negative literals, as well as at least one negative clause, containing no positive literals). Alternatively on "semantic" grounds it can be argued that the negative clause corresponds to the conclusion of the theorem and the positive clause to its hypothesis. Therefore attention can be limited either to SL-derivations with top clause $P(0)$ or to those with top clause $\bar{P}(s(s(s(0))))$. Either choice gives us effectively a unidirectional search, either "forward" from $P(0)$ or "backward" from $\bar{P}(s(s(s(0))))$. SL-resolution provides no facilities for bidirectional search.

Example. Sentences (and other strings of objects) can be represented by directed graphs whose edges are labeled by words, e.g. "(1) The (2) girl (3) guides (4) fish (5)." Axioms (1)–(4) below assert that "the girl guides fish" is a string of four words. Axioms (5)–(11) define the grammatical categories of the words in the string. A grammar adequate to determine the two ambiguous parses of the string as a sentence is given by axioms (12)–(17). Clause (18) is the negation of the theorem: that the string between points 1 and 5 is a sentence. This axiomatization of the parsing problem was developed by Colmerauer and the author. It is based upon Colmerauer's Q-systems [6] for writing and parsing grammars. Minker and Van der Brug [26] have independently investigated the application of theorem-provers to the parsing problem. Van Emden [9] compares the theorem-proving approach with more conventional methods. In [8] Colmerauer and his colleagues describe their use of predicate logic as a programming language for writing a natural language question-answering system.

(1) The(1, 2)
(2) girl(2, 3)
(3) guides(3, 4)
(4) fish(4, 5)
(5) $\overline{\text{The}}(x, y)$ det(x, y)
(6) $\overline{\text{girl}}(x, y)$ adj(x, y)
(7) $\overline{\text{girl}}(x, y)$ noun(x, y)
(8) $\overline{\text{guides}}(x, y)$ vb(x, y)
(9) $\overline{\text{guides}}(x, y)$ noun(x, y)
(10) $\overline{\text{fish}}(x, y)$ vb(x, y)
(11) $\overline{\text{fish}}(x, y)$ noun(x, y)
(12) $\overline{\text{det}}(x, y)$ $\overline{\text{noun}}(y, z)$ np(x, z)
(13) $\overline{\text{det}}(x, y)$ $\overline{\text{adj}}(y, z)$ $\overline{\text{noun}}(z, v)$ np(x, v)
(14) $\overline{\text{noun}}(x, y)$ np(x, y)
(15) $\overline{\text{vb}}(x, y)$ vp(x, y)
(16) $\overline{\text{vb}}(x, y)$ $\overline{\text{up}}(y, z)$ vp(x, z)
(17) $\overline{\text{up}}(x, y)\overline{\text{vp}}$ (y, z) S(x, z)
(18) $\bar{\text{S}}(1, 5)$

Corresponding to every refutation of (1)–(18) is a parse of the sentence. It is obvious, therefore, that every refutation must contain all five of the clauses (1)–(4), (18). To each such clause there corresponds an SL-refutation having that clause as top clause. Using the first clause (1), the search space corresponds to a bottom-up, left-to-right analysis of the sentence. Using the fourth clause (4), we obtain a bottom-up, right-to-left analysis. Using the negation of the theorem (18) results in a top-down parse. (In fact, the criterion for selecting literals determines, in all of these cases, whether the choice is uniformly one direction or another.) Although SL-resolution allows a certain degree of flexibility in that it permits an initial choice of different top clauses, it does not provide for the simultaneous exploitation of the different possibilities. We will see later that the new proof procedure provides us with the ability to execute a multidirectional search, with the search strategy determining which initial focus of attention to search from and when to switch from one focus to another. We will see how the problem of intersecting different directions of search, discussed by Pohl [28], is solved without introducing any mechanisms not already required for implementing the solution of the other problems we consider.

PROBLEM 5. OTHER RESOLUTION SYSTEMS. It is tempting to argue that SL-resolution is more efficient than all other resolution systems. Indeed, in the cases of set-of-support [37], A-ordering [14], resolution with merging [1], and general linear resolution [22, 24, 38], such arguments can be pursued a long way. But at least two other resolution systems need to be treated more seriously.

Hyperresolution [31] and its implementation by means of selective P_1-deduction [14] sometimes behaves more efficiently than SL-resolution. In the parsing problem, for example, selective P_1-deduction generates a bottom-up analysis which is completely flexible as to direction. The choice of left-to-right, right-to-left, or parallel bottom-up is left entirely to the discretion of the search strategy. Other examples can be cited where this kind of P_1-deduction admits no obvious redundancies and determines, in other respects, a satisfactory search space.

A second alternative to SL-resolution is Colmerauer's cancellation system [7]. It bears a greater resemblance to the new resolution system than does either SL-resolution or hyperresolution. It has the advantage of solving the first two problems of SL-resolution as well as of incorporating the two preprocessing procedures discussed below in connection with problem 6. Cancellation can be regarded as a variation of SL-resolution in which ancestor resolution is replaced by a restricted resolution of clauses occurring on different branches of an SL-search space. Without going into details, the system is easy to describe: Let S be the input set of clauses and let S' be an initial set-of-support. Let all literals in input clauses be equally most recent. Repeat the following procedure until $\square \in S'$: Pick a clause $C \in S'$. Pick a most recent literal $L \in C$. Add to both S and S' all resolvents D obtained by resolving on L in C with some clause $B \in S$. All literals in D descending from B are more recent than literals descending from C Delete C from S and S'.

It might be argued that SL-resolution, selective P_1-deduction, and cancellation ought to be regarded as alternatives, each of which is a useful member of a larger repertoire of solution methods Special problems require special methods. For one problem, one method might be useful, whereas for another problem a different method might be more useful. Our approach to the problem of reconciling competing resolution systems is less tolerant of differences.

Our goal is to design a single system which incorporates the best features of other systems but avoids the worst of their faults. The new theorem-proving system allows us to simulate SL-resolution, P_1-deduction, and cancellation. It allows us, moreover, to combine the different systems during the course of searching for a proof. In the course of simulating SL-resolution we avoid the inefficiencies and redundancies discussed in connection with problems 1 and 2.

PROBLEM 6. PREPROCESSING PROCEDURES. Preprocessing procedures transform one set of clauses into another set more highly instantiated, containing fewer clauses, or in

some other way better suited for input to a theorem-prover. On the other hand, various preprocessing procedures themselves resemble certain proof procedures. Colmerauer's cancellation is such a proof procedure. It can be used to good effect as a preprocessing procedure to delete clauses C which resolve with a small number of other clauses in the input set. It seems to us a deficiency of SL-resolution that various preprocessing procedures need to supplement the basic proof procedure. The theorem-proving system described later in this paper satisfactorily integrates preprocessing and proof procedure. In particular, preprocessing is not limited to an initial operation performed only on the set of input clauses, but is applicable at all stages in the course of searching for a proof.

The purity principle [30] asserts that a clause C may be deleted from an unsatisfiable set of clauses S if it contains some literal L such that no clause in S resolves with C on the literal L. The purity principle is one of the more useful preprocessing tools incorporated in the new theorem-proving system.

The purity principle is also incorporated in Colmerauer's cancellation: If L is pure in C, then pick C, pick $L \in C$. The set of resolvents obtained by resolving on L in C is empty. Adding the empty set of resolvents to the current unsatisfiable set of clauses S and deleting the parent clause C result overall in the deletion from S of the clause C containing the pure literal L.

More generally, cancellation incorporates a second preprocessing procedure which allows, under certain circumstances, one or both parents of a resolvent to be deleted when the resolvent is created. The second preprocessing procedure is just the operation which is the basis of the cancellation system: Select a clause C in S. Select a literal L in C. Generate and add to S all clauses which result from resolving on L in C with a clause $B \in S$. Delete C from S.

A similar deletion procedure has been investigated by Gelperin [10]. Related deletion procedures, reported by other authors [15, 25, 29], are also incorporated in the connection graph proof procedure.

Example. In the parsing problem, select for cancellation the single literals in the unit clauses (1)–(4), which specify the words occurring in the input string. These literals resolve only with clauses (5)–(11), which define the grammatical categories of the words in the input string. The resolvents obtained are the following clauses.

det(1, 2), adj(2, 3), noun(2, 3), vb(3, 4), noun(3, 4), vb(4, 5), noun(4, 5).

Clauses (1)–(4) are now deleted. If in clauses (5)–(11) we now select the first literals for cancellation, we notice that these literals have become pure and that therefore clauses (5)–(11) can also be deleted. Thus, in this example, the resolvents have replaced both of their parents in the input set of clauses. As in cancellation we aim for a proof procedure in which such a deletion procedure is an integral part.

PROBLEM 7. ACCESSING RELEVANT OPERATORS. A distressing characteristic of resolution proof procedures is the excessive amount of search which can be involved in accessing clauses which resolve with a selected literal. In the worst case, in nonlinear systems without some scheme for classifying clauses, it is necessary to search through the entire set of clauses before finding the subset of all clauses which resolve with the given literal. At any given time, a discriminating search strategy might examine only a small subset of the whole. But eventually it too examines the entire set of clauses. It is typical of resolution proof procedures that they occupy most of their time testing for resolvability of clauses which do not resolve. A recent paper concerned with this problem is devoted almost entirely to the task of speeding up the recognition that a pair of clauses fails to resolve [33].

With SL-resolution and with linear resolution systems generally, the situation improves significantly. For a given selected literal, it is necessary to search only through the set of input clauses for operators which resolve with the given literal. This set is fixed in size and does not grow during the course of searching for a proof. The situation can be further

improved by storing input operators according to the predicate letters they contain. In order to access operators it is necessary then to search only among those input clauses which contain the same predicate symbol as the selected literal but opposite in sign. It was the elaboration of this idea, in order to take into account the syntactic structure of the terms in literals, which resulted in the notion of classification tree Classification trees eventually evolved into matrices and finally into connection graphs. These structures were found useful not only for accessing operators but also for facilitating look-ahead. Although they were originally intended to supplement SL-resolution, they eventually assumed all the functions of a complete proof procedure. Thus it was the solution of the last of our seven problems which resulted in the solution of the other six.

Classification Trees

Classification trees [21] store input operators according to the syntactic structure of the literals with which they resolve. To any given selected literal there corresponds a unique branch of the tree. At the tip of the branch are located pointers to all the input operators which resolve with the literal scheme associated with the branch. The selected literal is an instance of this scheme and resolves only with input operators pointed to at the tip of the branch.

Example. The set of clauses classified by the tree in Figure 6 consists of five clauses, each containing two literals. Each input clause, therefore, has two associated operators. We employ the convention that an operator, written PQ, is resolved only upon its first literal P.

The leftmost branch in the figure has the associated literal scheme $L(v)$, where v is any variable. The rightmost branch is associated with literals of the form $\bar{M}(f(t))$, where t is any term. Such literals resolve only with the single operator 2′ listed at the tip of the branch.

Connection Graphs

The idea that matrices or graphs might provide a useful alternative to classification trees was suggested by Boyer. Figure 7 illustrates the connection graph for the example of Figure 6.

To every occurrence of a literal in an input clause there corresponds a node in the graph, labeled by the literal. Two nodes are connected by an undirected arc (called a *link*) if their literals are potentially complementary (i.e. if their literals can be made

(1) $L(x)K(x)$
(2) $\bar{K}(f(y))M(y)$
(3) $\bar{M}(u)L(f(a))$
(4) $\bar{L}(x)K(x)$
(5) $\bar{K}(f(x))\bar{L}(f(x))$

(1′) $K(x)\bar{L}(x)$
(2′) $M(y)\bar{K}(f(y))$
(3′) $L(f(a))\bar{M}(u)$
(4′) $K(x)\bar{L}(x)$
(5′) $\bar{L}(f(x))\bar{K}(f(x))$

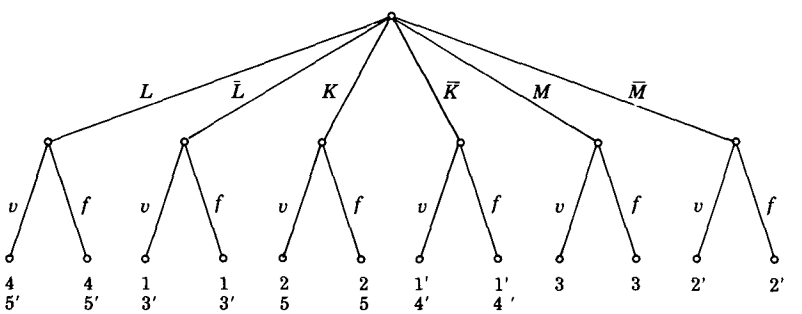

FIG. 6 A classification tree for operators (1)–(5) and (1′)–(5′)

complementary by applying some substitution, after renaming variables so that different clauses contain different variables). Nodes corresponding to literals which belong to the same clause are "grouped together" in the graph Since, in order to construct the graph, it is necessary to test pairs of literals to determine whether they are potential complements, it involves little additional computational effort to label each link with its associated most general unifying substitution.

Connection graphs can be used to store the entire set of clauses as it is generated. With each link in the graph is associated the single resolvent obtained by resolving away the two literals at the opposite ends of the link. When the resolvent is generated, it is a simple matter to add it and its associated links to the graph. The new links connected to literals L' in the resolvent are constructed without searching by examining the old links connected to the literals L from which L' descends: A new link connects L' to K if (1) an old link connects L to K, and (2) the substitution associated with the old link is compatible with the substitution associated with the link whose activation generated the resolvent. The substitution associated with the new link can be computed directly from the two compatible substitutions. An algorithm for computing these substitutions has been developed by Jophien van Vaalen at the Mathematical Centre in Amsterdam. Figure 8 illustrates the operations involved in adding a resolvent to a connection graph.

Given a selected link in a connection graph, a new connection graph is obtained by adding the associated resolvent to the graph, adding the new links connected to its literals, and deleting certain links and clauses from the old connection graph:

1. When a resolvent is generated and added to a graph, the link which generated the resolvent is deleted from the graph.

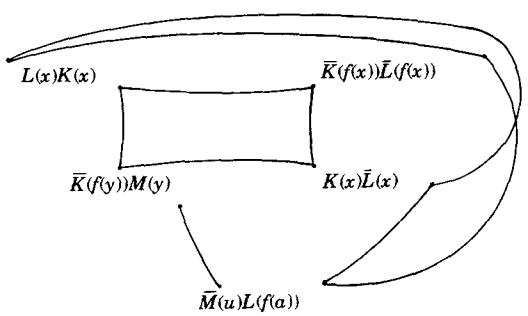

FIG. 7. The connection graph for the example of Figure 6

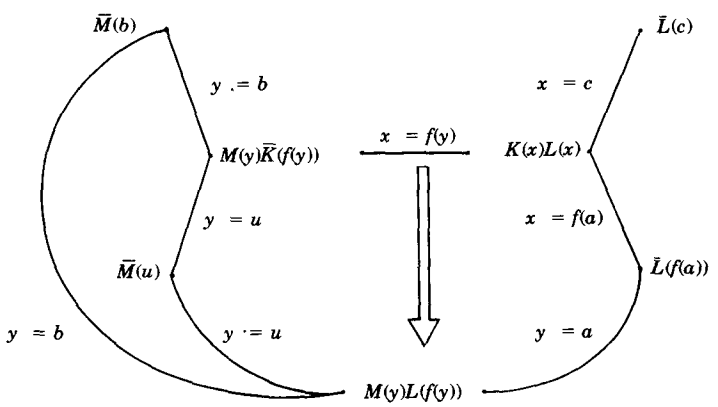

FIG. 8 The addition of a resolvent to a connection graph. The arrow points from the link which generates the resolvent to the resolvent itself. The links connected to the literals $M(y)$ and $L(f(y))$ in the resolvent descend from the links connected to the literals $M(y)$ and $L(x)$ from which they descend in the parent clauses.

2. If a node has no links (i.e if its labeling literal is pure), then the clause in which the node occurs and all links attached to nodes in the clause are deleted from the graph.

3. If a clause is a tautology, then it is deleted together with all its nodes and their links.

The new proof procedure consists of the application of the following nondeterministic procedure to the initial connection graph for the input set of clauses:

1. Terminate successfully if the graph contains the empty clause.

2. Otherwise, select a link in the graph, generate the associated resolvent, and construct the connection graph for the new set of clauses. Apply the procedure recursively to the new connection graph.

Much of the flexibility and power of the proof procedure owes to its nondeterminism: the fact that any link can be selected to generate the new connection graph from the old one.

In order to make the definition precise it is necessary to pay attention to details, taking special care with operations for self-resolving clauses (i.e. clauses which resolve with a copy of themselves) and with the merging and factoring operations. Ignoring these details for the moment, Figure 9 illustrates a sequence of transformations of a connection graph which detects the unsatisfiability of the set of clauses in Figures 1–3.

Historically, the step from connection graphs to the connection graph proof procedure was suggested by Winston's description [35], in 1972, of Waltz's picture interpretation algorithm.

Connection Graphs and the Waltz Algorithm

Waltz is concerned with the problem of constructing from local information a global interpretation of a two-dimensional line drawing as a scene. In order to do so, he constructs a graph whose nodes represent vertices in the picture and whose edges represent lines joining vertices. With each node in the graph is associated a large set of possible local interpretations of the corresponding vertex A single global interpretation of the entire drawing is obtained by selecting, for every node in the graph, a single interpretation from among the complete set of possible interpretations associated with the node. Nodes joined by an edge must be labeled by interpretations which satisfy certain compatibility restrictions If all pairs of nodes joined by an edge are labeled by compatible pairs of interpretations, then the labeling determines a single global interpretation of the drawing.

The algorithm starts with each node in the graph labeled by the complete set of all its possible interpretations. It proceeds by consecutively examining pairs of nodes joined by an edge, deleting an interpretation from the labeling of one of the nodes if it is incompatible with all interpretations in the labeling of the other node. Later certain interpretations in the labeling of the second node may become deleted because of incompatibilities associated with other edges in the graph. At such a time it will be necessary to

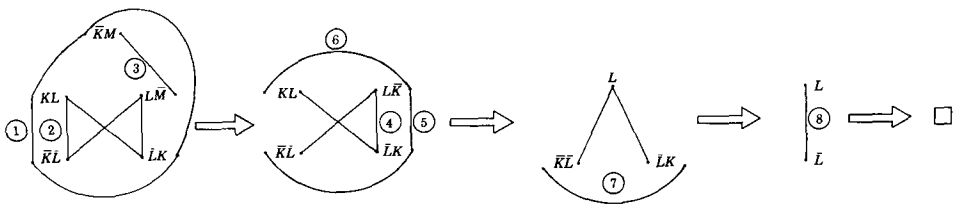

FIG. 9 A sequence of transformations of the connection graph for the set of clauses in Figures 1–3. Links ①, ②, ④, and ⑤ have resolvents which are tautologies These links may be deleted from the graph as soon as they are generated, without waiting until the tautologies are added and deleted. Links ⑥ and ⑦ have merge resolvents which involve identifying literals from different parents. The resolvents corresponding to ③, ⑥, ⑦, and ⑧ replace both of their parents, because once the resolvents are generated and the links deleted, both parent clauses contain unlinked literals.

reconsider the original edge and to test whether any other interpretations associated with the first node can be deleted because they have become incompatible with all interpretations in the diminished set associated with the second node. The deletion of incompatible interpretations continues until eventually no node has an associated interpretation which is incompatible with all interpretations associated with some neighboring node. Winston claimed that by this time there usually remained only a very small number of interpretations associated with every node.

It was immediately clear that a procedure analogous to Waltz's could be applied to diminish the number of links in a connection graph. Suppose that L and K are distinct literals belonging to the same clause. A link connected to the literal L can be deleted from the graph if its associated substitution is incompatible with all substitutions associated with the links connected to K. The consequences of deleting a link propagate throughout the connection graph in the same way that they do for deleting an interpretation in Waltz's picture graph. Figure 10 illustrates the transformation of a connection graph effected by deleting incompatible links.

That incompatibility deletion occupies the central role in Waltz's algorithm suggested that the analogous operation might play an important role in a complete proof procedure based on connection graphs. From this suggestion it was an easy step to the definition of the operation which deletes a link and adds to the connection graph the associated resolvent and the new links connected to its literals. It is interesting to note that this operation together with the operation of deleting clauses containing pure literals subsumes the operation of deleting incompatible links: If a link connected to a literal L is incompatible with all links connected to a literal K in the same clause, then generate the resolvent corresponding to the incompatible link, delete the link, and add the resolvent to the graph. But notice that the incompatibility of all links connected to K with the deleted link connected to L means that the descendant of the literal K in the resolvent is now pure and therefore the entire resolvent may be deleted from the graph. The total

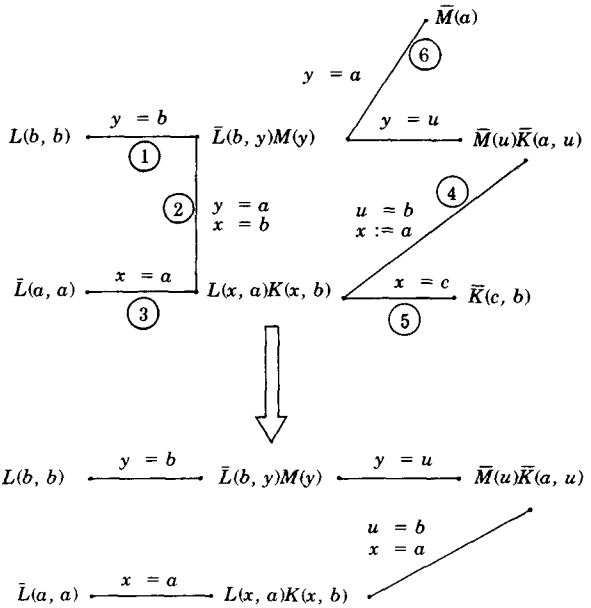

FIG 10. Deletion of incompatible links Link ⑤ is incompatible with both ② and ③. Deletion of ⑤ makes clause $\bar{K}(c, b)$ contain a pure literal This clause may consequently be deleted from the graph Link ② is incompatible with ④. Deletion of ② makes ⑥ incompatible with all links connected to $\bar{L}(b, y)$. Deletion of ⑥ makes $\bar{M}(a)$ pure and consequently this clause is also deleted from the graph.

effect of this sequence of operations is simply to delete the original incompatible link connected to the literal L from the connection graph

Notice that Waltz's incompatibility deletion is used simply as an initial preprocessing procedure. In connection graphs, incompatibility deletion is an integral part of the proof procedure and can be applied at any time during the course of searching for a proof.

Connection Graphs and the New Proof Procedure

It is necessary to supplement the resolution operation with a factoring operation which merges potentially identical literals in the same clause. An instance $C\theta$ of a clause C is a *factor* of C if θ is a most general unifier of some subset of literals in C Like the resolution operation, the operation of generating factors leads to excessive redundancy unless restrictions are placed on its use. Such restrictions have been imposed upon the factoring operation in SL-resolution and in other systems [16]. In order to simplify the definition of the connection graph proof procedure, we shall ignore restrictions on the factoring rule. The *m-factoring* method employed in SL-resolution and other forms of factoring employed in other proof procedures can easily be adapted to the new proof procedure On the other hand, it is likely that new factoring methods which take account of the special bookkeeping structure of connection graphs will be more suitable for the new proof procedure.

The *proof procedure* is the following nondeterministic program:

(1) In order to determine that a set S of clauses is unsatisfiable:
  (a) Construct and
  (b) Process the initial connection graph for S
(2) In order to construct the initial connection graph for S
  (a) Generate and include in the graph all factors of clauses in S.
  (b) For every pair of potentially complimentary literals in distinct clauses in the graph, insert a link connecting the literals.
  (c) Label each link by the most general unifying substitution which makes the literals complimentary.
(3) In order to process a connection graph.
  If the graph contains the empty clause,
  Terminate successfully,
  If the graph does not contain the empty clause:
  (a) Select a link in the graph,
    Generate the associated resolvent,
    Add its factors to the graph, and
    Delete the selected link
  (b) Add links connecting literals in the newly generated factors to other literals in the graph.
    (i) If $L\theta$ in a new clause descends from L in a parent clause,
      If a link labeled by σ connects L to K, and
      If θ and σ are compatible, then
      Add to the graph a link connecting $L\theta$ and K labeled by the substitution $\theta * \sigma$ which is the most general unifier of $L\theta$ and K.
    (ii) If $L\theta$ in a new clause is potentially complimentary to a literal occurrence K in another factor of the resolvent or in a parent clause, then
      Add a link connecting $L\theta$ and K, labeled by the most general unifier of $L\theta$ and K.
  (c) Delete a clause and all links connected to its literals
    (i) If the clause is a tautology, or
    (ii) If the clause contains a pure literal (one not connected by a link to any other literal in the graph).
  (d) Process the new connection graph

Figures 11–13 illustrate various details of the algorithm.

The algorithm as it has just been defined is nondeterministic. For implementation on a deterministic machine it needs the specification of a search procedure which determines the selection of links. There are two approaches to the specification of search procedures. One approach, advocated by Hayes [12], is to devise a control language with facilities for the axiom writer to specify the criteria for the selection of links. The other, more conven-

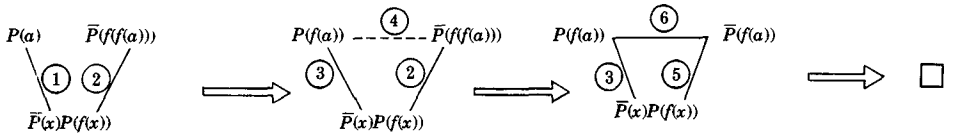

FIG. 11 Bidirectional search with a self-resolving clause A self-resolving clause, more precisely, is one which resolves with a copy of itself The first transformation is obtained by selecting link ①. The resolvent is its own only factor. Its parent $P(a)$ is deleted because it contains a pure literal after deletion of ① The new link ③ is added by step (3-b-ii). According to step (3-b-i) an attempt is made to add link ④ This attempt fails The next link selected is ② The parent $\bar{P}(f(f(a)))$ of the new resolvent is deleted because it now contains a literal which is pure. New links ⑤ and ⑥ are added to the graph. The null clause results from the selection of link ⑥.

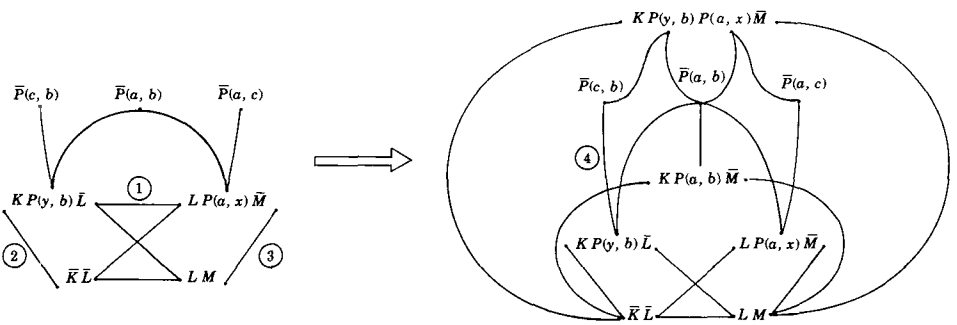

FIG 12. The addition of factors to a connection graph. Link ① is selected The two factors of the associated resolvent are added to the graph. Notice that selection of link ② or link ③, instead of link ①, would have resulted in the generation of a much simpler graph. The graph in Figure 13 is obtained by selecting link ④ in the new graph

tional, approach is to devise an autonomous sequencing algorithm which decides for itself. We shall have more to say about search procedures in the final section of this paper. But first we shall explain the criteria used for selecting links in the examples of the next section. For purposes of simplicity we ignore the factoring operation altogether.

In general it is desirable to select links whose activation simplifies the graph by reducing the total number of clauses, literal occurrences, or links. The ideal situation occurs when a link connects a pair of literals and no other link is connected to either literal In such a case the resolvent completely replaces its parents (because after deletion of the link, both parents contain pure literals) The graph is further simplified when the resolvent merges literals and links descending from different parents. The selection of links ⑥ and ⑦ in the graph of Figure 9 illustrates such a situation. The next most desirable circumstance is the one in which a literal occurrence in a clause is connected to just a single link. In this case the one clause is replaced by its resolvent. In the special subcase where the other parent is a unit clause or where all literals and connecting links descending from the undeleted parent merge into those descending from the deleted parent, selection of the link simplifies the graph by removing one literal occurrence and one link. In situations where the activation of no link simplifies the graph, it is necessary to select a link whose activation least complicates it. In such cases preference should be given to links whose resolvents contain the fewest literals. Both the criterion of selecting links which least complicate the graph and the criterion of generating clauses containing fewest literals can be improved by using the look-ahead procedure described in the last section of this paper.

When it is necessary to complicate the graph, the selected link should belong to a set-of-support, i.e. it should descend either from the negation of the conclusion of the theorem or from one of its hypotheses. Such a goal-oriented selection of links simulates the forward- and backward-chaining theorem-provers of Bledsoe and his colleagues [2–4].

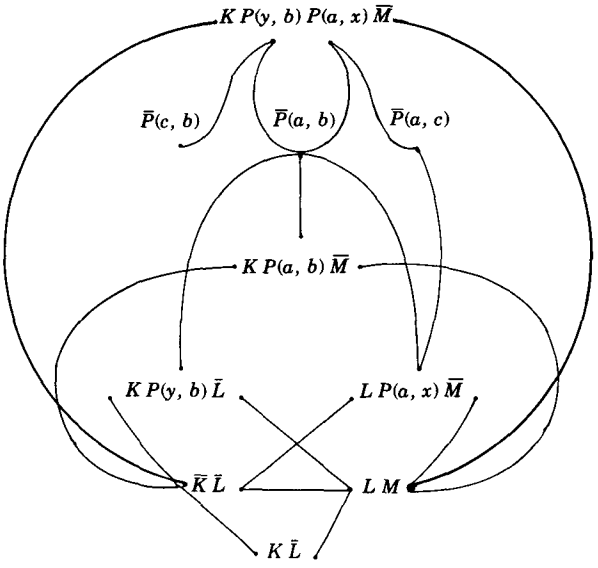

FIG. 13 The addition of links to a connection graph. The clause $K\bar{L}$ results from the selection of link ④ in the graph of Figure 12. Notice that no link is added connecting the occurrence of $\bar{L}$ in $K\bar{L}$ to the occurrence of L in $L\ P(a, x)\ \bar{M}$, even though these two literals are complementary. This is because the graph contains no link connecting $\bar{L}$ in the parent of the resolvent to L in $L\ P(a, x)\ \bar{M}$. That link has already been activated and deleted.

The New Proof Procedure and the Deficiencies of SL-Resolution

We shall investigate how the new proof procedure copes with the deficiencies of SL-resolution.

Recall that problem 7, the problem of accessing relevant operators, has been solved by using connection graphs to store clauses. Indeed, immediately after a clause is generated, all resolvents which can be obtained from the clause are implicitly represented by the newly added links which connect literals in the new clause to other literals in the graph. The new links are generated by copying them from the old links connected to literals in the parent clauses. Moreover, the substitutions associated with the new links can be computed directly from the substitutions associated with the parent links together with the substitution associated with the link whose activation generated the resolvent

Operations on connection graphs solve problem 6, the problem of incorporating preprocessing procedures into the proof procedure itself. This is obvious for the purity principle which plays a central role in the new theorem-proving system. The new system also incorporates the preprocessing procedure of Colmerauer's cancellation system In connection graphs cancellation is accomplished by simultaneously selecting at every stage all the links connected to some selected literal occurrence in a given clause. When the corresponding resolvents are generated the selected literal occurrence becomes pure and the clause in which it occurs is deleted from the graph.

The new theorem-proving system reconciles problem 5, the problem of competing resolution systems by incorporating them (or rather improved variations of them) as special ways of determining the selection of links. We have just remarked how links can be selected in order to simulate Colmerauer's cancellation. To simulate selective P_1-deduction it suffices always to select a link connected to a literal in a positive clause. Such a clause always exists in any unsatisfiable set of clauses. To simulate SL-resolution with a given set of top clauses, it suffices always to select a link connected to a most recent literal in some clause descending from a top clause. In the first example at the end of this section, we see how the new system solves problems 1 and 2, the problem of dealing with contra-

dictory units and the problem of avoiding redundancy when the selection of links is used to simulate SL-resolution.

The solution of problem 4, the problem of support sets and multidirectional search, has already been illustrated in the example of Figure 11. Notice in that example how the test for intersection of the bidirectional searches is accomplished automatically by the attempt to add new links to the graph whenever a new clause is generated. Pohl recommends a scheme of hash coding to alleviate the computational difficulties of intersecting bidirectional searches [28]. It is significant that the use of connection graphs to solve the problem of accessing operators also solves the problem of intersecting multidirectional searches. Multidirectional search for the parsing problem is illustrated in the second of the three examples which follow.

Problem 3, the problem of more flexible activation of subgoals, is solved by the absence of restrictions on the selection of links in the connection graph proof procedure. Not only can the selection of links be constrained so as to simulate SL-resolution, but selection can be liberalized so as to perform a generalization of SL-resolution which allows the simultaneous consideration of several subgoals. The third example below illustrates such a selection of links for the sorting example considered earlier.

Example. This example deals with the solution of problem 1, the problem of contradictory units, and of problem 2, the problem of redundant derivations. Figure 14 illustrates a redundant SL-resolution search space and Figure 15 illustrates a connection graph simulation of SL-resolution with a breadth-first search strategy for the same set of input clauses. The connection graph simulation generates the upper half of the SL-search space. But then it finds (in link ⑦) the contradictory pair of units $\bar{S}$ and S. After the contradiction has been generated, nothing remains in the connection graph, corresponding to the lower half of the SL-search space. The connection graph simulation avoids both problems 1 and 2 of SL-resolution.

It is interesting to perform, for this set of clauses, a connection graph simulation of SL-resolution with other search strategies. SL-resolution with backtracking (depth-first search) directly generates one of the two redundant refutations in the SL-search space. After the first refutation has been generated, the second refutation remains a candidate for generation in the SL-search space. The connection graph simulation of SL-resolution

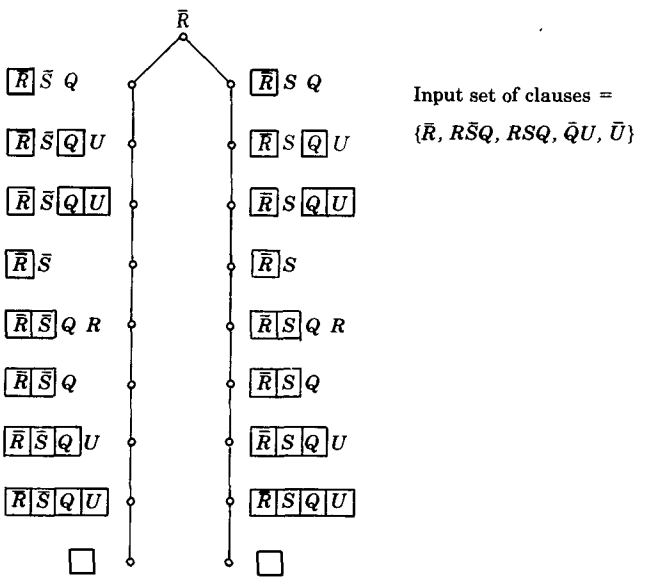

FIG. 14. A redundant SL-search space, with unrecognized contradictory units $\bar{S}$ and S.

with backtracking generates the same first refutation but, having done so, leaves no other candidate resolution in the search space.

The new theorem-proving system is interesting for the maximal confusion it makes between search space and search strategy. It is especially interesting how an unnecessary search in an irrelevant part of a connection graph alters the search space in general. In particular, an unnecessary search often simplifies the graph and makes it easier to find the necessary parts of the search space.

Example. The graph in Figure 16 is obtained from the initial connection graph for the

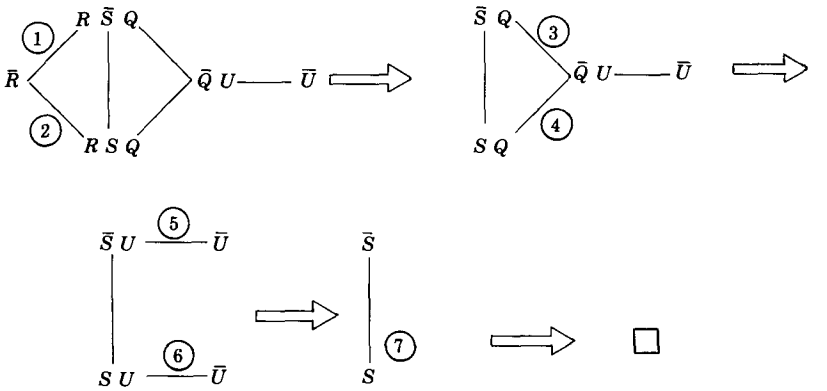

FIG. 15 A connection graph simulation of SL-resolution with breadth-first search, for the set of clauses and selection function of Figure 14 The numbering of links indicates the order in which they are selected for deletion.

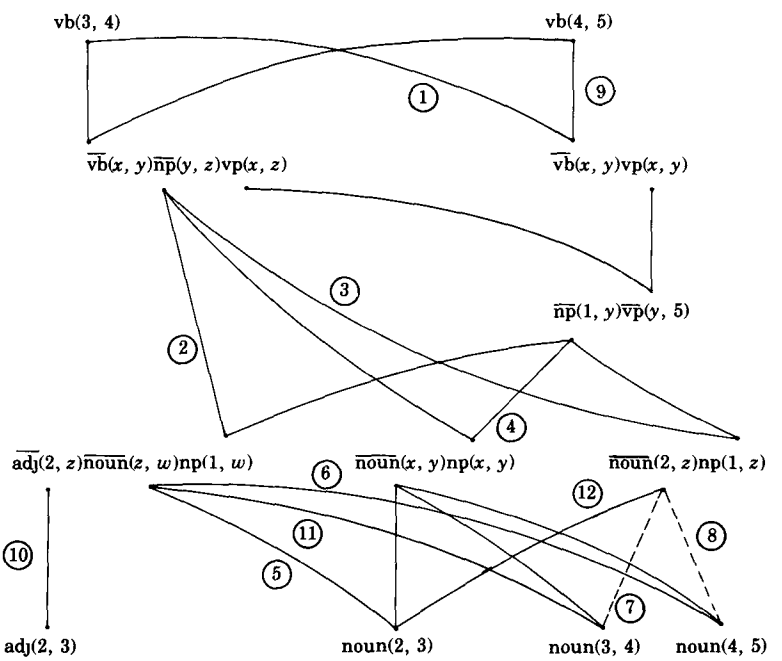

FIG. 16. The connection graph for the parsing problem, after initial preprocessing. The next transformation of the graph is obtained by first deleting the incompatible links ①–⑧, and then selecting for activation links ⑨–⑫ The resolvent associated with link ⑩ replaces both its parents The resolvents associated with ⑨, ⑪, and ⑫ replace and are simpler than their nonunit parents. Thus activation of each of links ⑨–⑫ simplifies the graph.

parsing problem by generating most of the resolvents which replace both their parents because the associated link is the only link connected to the literals resolved upon in the parents. Notice that this initial "preprocessing" of the graph is an integral part of the proof procedure Notice too that during preprocessing the connection graph operations simulate both SL-resolution and hyperresolution. The parsing method pursued so far has been a multidirectional bottom-up and top-down analysis. Figures 17–19 illustrate suc-

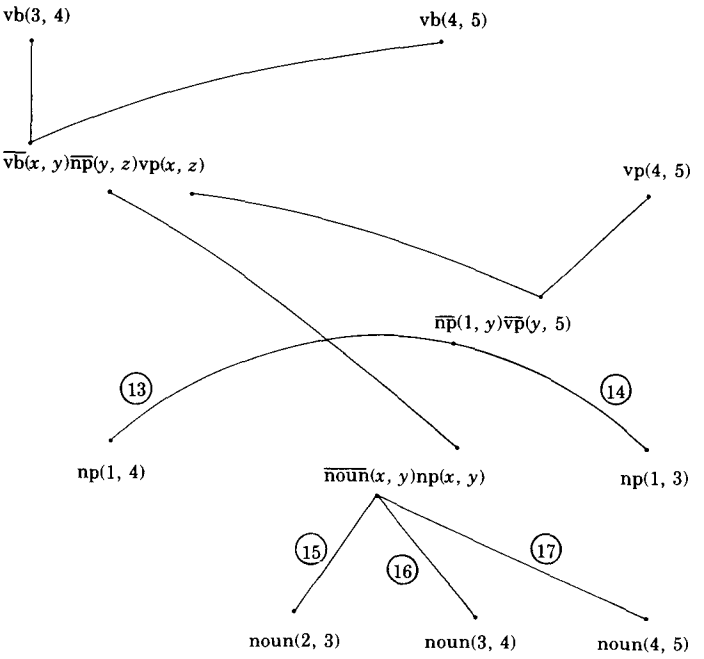

FIG 17. The connection graph which results from deleting links ①–⑫ in Figure 16. The next transformation of the graph results from the activation of links ⑬–⑰ The associated resolvents all replace their parents. Notice that deletion of ⑬–⑰ simulates Colmerauer's cancellation.

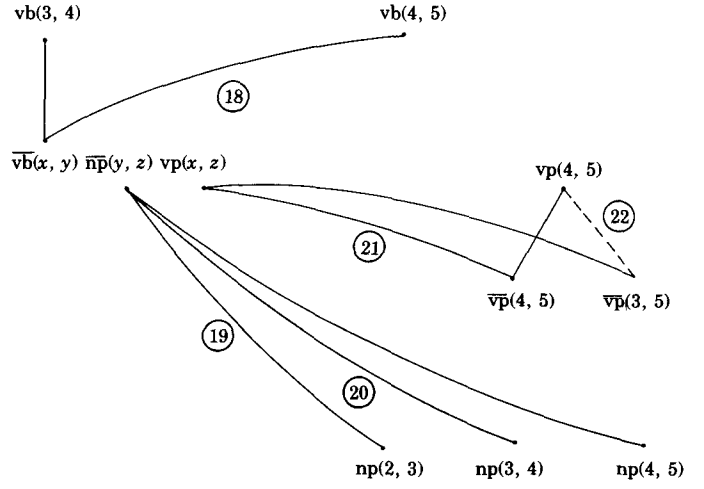

FIG 18 The connection graph resulting from the deletion of links ⑬–⑰ in Figure 17. Links ⑱–㉒ are now incompatible and can be deleted

cessive transformations of the original graph. In this example, the remainder of the parsing is done primarily in a bottom-up manner. But notice that deletion of links ⑬ and ⑭ can be interpreted as either bottom-up or top-down In fact, deletion of these links effects the intersection of the bottom-up and top-down analyses.

Example. Figures 20–22 illustrate the connection graph simulation of the SL-derivation of Figure 5 for the sorting problem. The connection graph in Figure 22, however, provides the opportunity to activate the new instance $\bar{O}(\text{cons}(2, \text{cons}(1, y)))$ of an old subgoal. The new instance is easily recognized as unsolvable. This example illustrates the ability to consider subgoals in a more flexible sequence than is allowed by SL-resolution. In the programming interpretation, the example illustrates the important ability of connection graphs to implement coroutines.

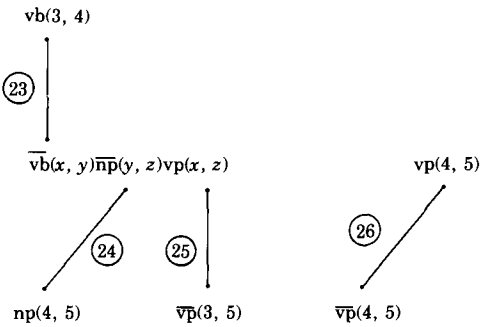

FIG 19 The connection graph which results from deleting the incompatible links ⑱–㉒. In this graph, deletion of links ㉓–㉕ results in a refutation corresponding to one parse of the ambiguous sentence, and deletion of link ㉖ results in a different refutation corresponding to the other parse

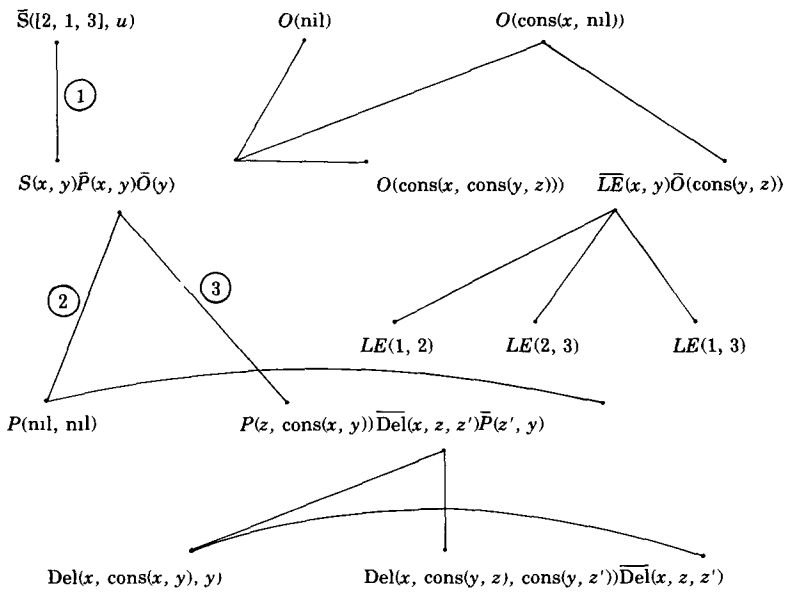

FIG 20 The connection graph for the sorting problem Here [2, 1, 3] is an abbreviation for the longer cons(2, cons(1, cons(3, nil))) Notice that three of the clauses in this graph are self-resolving. Thus, although some of their literals appear to have just single links connecting them to other literals in the graph, these clauses cannot be deleted when the links are deleted, because the literals do not become pure but rather become linked to other literals in the resolvent. Keeping this in mind, it is clear that initially only deletion of link ① results in simplification of the graph But then the descendant of link ② can be deleted because it is incompatible. The descendant of link ③ is then the only link whose deletion results in a literal becoming pure with the consequent deletion of a parent clause from the graph.

It is worth remarking that, although the sequence of transformations in Figures 20–22 has been chosen in order to simulate SL-resolution, the same sequence of transformations is determined by the autonomous selection procedure outlined in the preceding section. At every stage the selected link is one connected to a clause descending from the negation $\overline{S}([2, 1, 3], u)$ of the conclusion of the theorem, whose activation least complicates the graph and whose resolvent contains fewest literals.

Connection Graphs and Search Strategy

In connection graphs, the problem of search strategy is the problem of deciding at every stage which link should be selected. This problem is the same as the problem of sequencing the generation of resolvents in more conventional resolution systems. In particular, the usual search strategies for resolution systems, such as unit preference [36] or diagonal search [17], apply without complication to the new theorem-proving system. Moreover, search strategic advice of the kind employed in programming languages such as PLANNER [13] and supplied, for example, in the form of recommendation lists and filters (for sequencing the application of operators to subgoals) can also be employed to guide the search in connection graphs. What distinguishes connection graphs is that they facilitate the implementation of methods for obtaining useful information about ungenerated parts of the search space. This information can be used in a variety of ways, one of the most interesting of which is for the calculation of lower bounds on the complexity of a simplest refutation containing a given resolvent. Present diagonal search strategies use a very crude and approximate lower bound which is just the number of literals in the resolvent. A well-known property of search strategies which are guided by lower bounds is that their efficiency improves significantly with a significant improvement in the accuracy of the lower bound. The number of literals in a clause is equivalent to the lower bound obtained by just a one-level look-ahead in the connection graph. One-level look-ahead assumes that all literals in a clause might be linked to unit clauses. The lower bound can be improved significantly by n-level look-ahead.

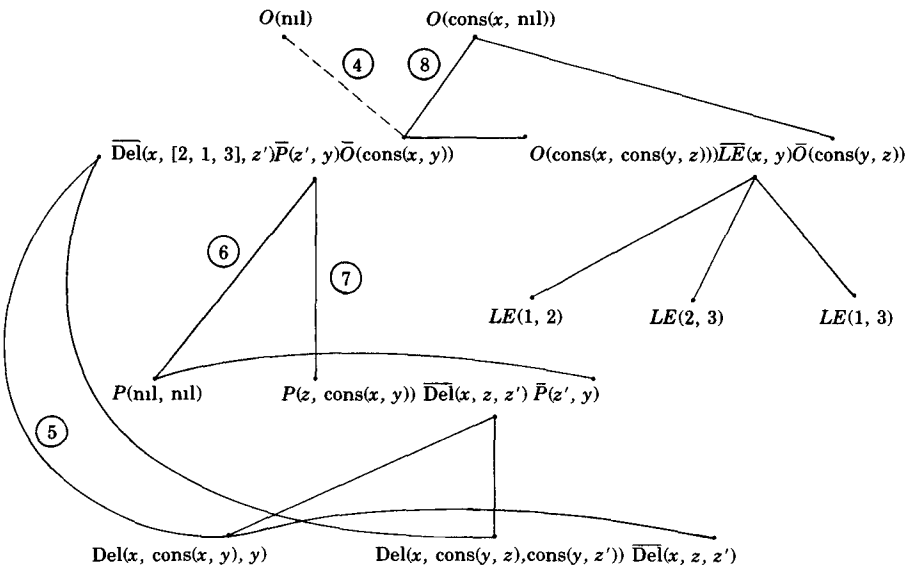

FIG. 21. The connection graph which results from deleting ①–③ in Figure 20. Link ④ can be deleted because it is incompatible Link ⑤ is deleted, simulating the SL-derivation of Figure 5. But then the descendant of link ⑥ in the resolvent is incompatible and the descendant of link ⑦ can be activated with the associated resolvent replacing one of its parents In the new resolvent, the descendant of link ⑧ is not present because links ⑧ and ⑦ are incompatible.

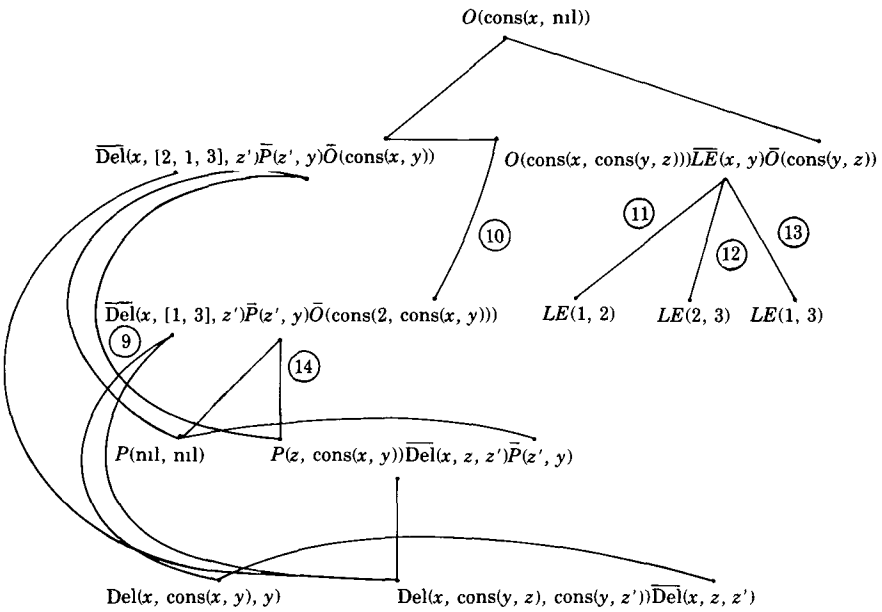

FIG 22. The connection graph which results from deleting ④–⑧ in Figure 21 Continuing the simulation of the SL-derivation in Figure 5, we delete link ⑨ But now the descendant of link ⑩ in the resolvent is incompatible with links ⑪, ⑫, and ⑬ The descendant of link ⑩ can therefore be deleted But then the resolvent contains a pure literal and the entire resolvent together with all its links can be deleted from the graph The total effect of this sequence of operations is simply to delete link ⑨ from the graph Ordinary SL-resolution, with its rigid last-in-first-out activation of subgoals, would generate the resolvent associated with link ⑨ and then generate the resolvent associated with the descendant of link ⑭.

In the example below, we show how to compute the quantities which estimate, by looking ahead n-levels in the connection graph, the minimum number of links which need to be activated in order to achieve the goal L. Notice below that the computational effort needed to compute these estimates is just a linear function of n, even though the size of the search space n-levels deep is generally an exponential function of n. Notice too that the look-ahead ignores the possibility of merging, factoring, and deleting tautologies. Moreover, it assumes that all substitutions are, and will be, compatible. For these reasons our look-ahead method resembles the relaxation method for fathoming subgoals and for computing lower bounds. A useful discussion of such applications of relaxation can be found in Geoffrion and Marsten's survey of integer programming [11].

Figure 23 illustrates the calculation of the quantities h_n for each literal occurrence in the connection graph of Figure 24. $h_n(L)$ estimates the minimum number of links which need to be activated in order to achieve the subgoal L. $h_1(L)$ is calculated by assuming that each literal in every clause directly connected by a link to L can be achieved in a single step by activating a single link. Thus $h_1(L)$ is the minimum, over all clauses connected by a link to L, of the number of literals in such clauses. More generally, $h_n(L)$ is one plus the minimum, over all clauses $\bar{L}K_1 \cdots K_m$ connected by a link to L, of the sums $\sum_{i=1,m} h_{n-1}(K_i)$. Notice that, ignoring situations which involve merging, $h_n(L)$ is a lower bound on the number of resolvents which need to be generated in order to achieve the goal of refuting L.

Look-ahead can be usefully employed to compute quantities other than h_n. In particular, it can be used to estimate the size of the entire search space n-levels deep which is generated by attempting to achieve in n steps a subgoal in all possible ways. We have already observed the utility of employing the one-level look-ahead estimate of this quan-

	h_1	h_2	h_3	h_4	h_5
A	3	6	7	7	7
B	1	1	1	1	1
C	3	4	4	4	4
D	2	2	2	2	2
E	1	1	1	1	1
F	3	4	4	4	4
G	2	2	2	2	2
H	3	5	6	6	6
I	1	1	1	1	1
J	2	4	6	7	7

FIG 23 The matrix of the values h_n for the literal occurrences in the graph of Figure 24. $h_n(L) = h_5(L) = h_4(L)$ for all $n \geq 5$ and for all $L = A, B, \cdots, J$.

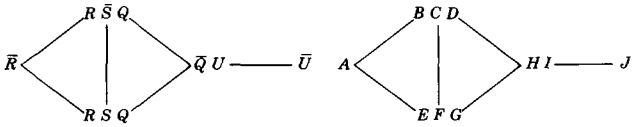

FIG 24 The assignment of unique names to distinct occurrences of literals in a connection graph. For example, the occurrence of R in $R\ \bar{S}\ Q$ is called B, whereas the occurrence of R in $R\ S\ Q$ is called E This assignment is used in Figure 23 for computing the function h_n

tity, namely the total number of links connected to a literal Good use can be made of more accurate n-level look-ahead estimates We shall not pursue further in this paper the details of these computations, nor their employment for the guidance of the search strategy.

More elaborate look-ahead computation can be performed in order to estimate the symbol complexity [17] of a simplest refutation containing a given clause, rather than to estimate its size (number of clauses). We have argued in this section that connection graphs provide new opportunities for improving search strategies by exploiting efficient methods for looking ahead in the ungenerated part of the search space. We do not claim to have solved the general problem of designing intelligent search strategies. We believe, however, that the employment of connection graphs and look-ahead will contribute to its eventual solution.

ACKNOWLEDGMENTS. Thanks are due to Jophien van Vaalen and Alain Colmerauer for useful interaction during the early part of this work. We have benefited also from the interest and encouragement of Aaron Sloman.

REFERENCES

(Note Reference [23] is not cited in the text.)

1 ANDREWS, P B. Resolution with merging. *J ACM 15*, 3 (July 1968), 367–381.
2. BLEDSOE, W. W. Splitting and reduction heuristics in automatic theorem-proving *Artif. Intel 2* (1971), 55–77
3. BLEDSOE, W W., BOYER, R. S , AND HENNEMAN, W H Computer proofs of limit theorems. *Artif. Intel 3* (1972), 27–60
4. BLEDSOE, W W., AND BRUELL, P. A man-machine theorem-proving system. Third International Joint Conf on Artificial Intelligence, 1973, pp. 36–65.
5 BOYER, R. S., AND MOORE, J S. The sharing of structure in theorem-proving programs. In *Machine Intelligence 7*, B. Meltzer and D Michie, Eds., Edinburgh U. Press, Edinburgh, Scotland, 1972, pp. 101–116
6 COLMERAUER, A. Les systemes-*Q* ou un formalisme pour analyser et synthetiser des phrases sur ordinateur Pub. interne No 43, Dep d'Informatique, Université de Montréal, Montréal, Canada, 1970.
7 COLMERAUER, A Cancellation systems Personal communication, 1972.
8. COLMERAUER, A., KANOUI, H., PASERO, R , AND ROUSSEL, P. Un systeme de communication homme-machine en Francais. Rapport preliminaire, Groupe de Researche en Intelligence Artificielle, Université d'Aix Marseille, Luminy, 1972.
9 EMDEN, M H van. First-order predicate logic as a high-level program language. Memo No. MIP-R-106, U. of Edinburgh, Edinburgh, Scotland, 1974.
10. GELPERIN, D Deletion-directed search in resolution-based proof procedures Advance Papers of the Third International Joint Conf on Artificial Intelligence, Stanford U., Stanford, Calif., 1973, pp 47–50.
11. GEOFFRION, A M., AND MARSTEN, R. E Integer programming algorithms· A framework and state-of-the-art survey. *Manag. Sci. 18* (1972), 465–491.
12 HAYES, P. J Computation and deduction. Proc. 2nd MFCS Symposium, Czechoslovak Academy of Sciences, 1973, pp. 105–118.

13 HEWITT, C. PLANNER· A language for proving theorems in robots. Proc of the International Joint Conf. on Artificial Intelligence, Washington D. C , 1969, pp 295–301.
14. KOWALSKI, R , AND HAYES, P. J. Semantic trees in automatic theorem-proving. In *Machine Intelligence 4*, B. Meltzer and D. Michie, Eds , Edinburgh U. Press, Edinburgh, Scotland, 1968, pp. 87–101.
15. KOWALSKI, R. The case for using equality axioms in automatic demonstration. Proc. IRIA Symposium on Automatic Demonstration (Lecture Notes in Mathematics, No. 125), Springer-Verlag, Berlin, Heidelberg, New York, 1968, pp. 112–127.
16. KOWALSKI, R. Studies in the completeness and efficiency of theorem-proving by resolution. Ph.D Th., U. of Edinburgh, Edinburgh, Scotland, 1970.
17. KOWALSKI, R. And-or graphs, theorem-proving graphs and bi-directional search In *Machine Intelligence 7*, B Meltzer and D Michie, Eds., Edinburgh U Press, Edinburgh, Scotland, 1972, pp 167–194.
18. KOWALSKI, R An improved theorem-proving system for first-order logic D C.L. Memo No. 65, U of Edinburgh, Edinburgh, Scotland, 1973.
19. KOWALSKI, R Predicate logic as programming language. Proc IFIP Cong. 1974, Stockholm, North-Holland Pub. Co , Amsterdam, pp. 569–574.
20. KOWALSKI, R Logic for problem-solving D.C L. Memo No. 75, U. of Edinburgh, Edinburgh, Scotland, 1974
21 KOWALSKI, R., AND KUEHNER, D. Linear resolution with selection function. *Artif. Intel 2* (1971), 227–260.
22. LOVELAND, D. A linear format for resolution Proc. IRIA Symp on Automatic Demonstration, Versailles, France, Springer-Verlag, Berlin, Heidelberg, New York, 1970, pp. 147–162.
23 LUCKHAM, D. Some tree-paring strategies for theorem-proving. In *Machine Intelligence 3*, D Michie, Ed , Edinburgh U Press, Edinburgh, Scotland, 1968, pp. 95–112.
24 LUCKHAM, D Refinement theorems in resolution theory. Proc. IRIA Symp on Automatic Demonstration, Versailles, France, Springer-Verlag, Berlin, Heidelberg, New York, 1970, pp. 162–190
25. MELTZER, B. Some notes on resolution strategies. In *Machine Intelligence 3*, D. Michie, Ed , Edinburgh U. Press, Edinburgh, Scotland, 1968, pp 71–75
26 MINKER, J , AND VAN DER BRUG, G. J Representations of the language recognition problem for a theorem-prover. Tech. Rep. TR-199, Computer Science Center, U of Maryland, College Park, Md., 1972.
27. NILSSON, N. J. Problem solving methods in artificial intelligence. McGraw-Hill, New York, 1971
28 POHL, I. Bi-directional search. In *Machine Intelligence 7*, B. Meltzer and D. Michie, Eds., Edinburgh U. Press, Edinburgh, Scotland, 1972, pp 127–140.
29. REITER, R. The predicate elimination strategy in theorem-proving Proc. 2nd ACM Symp on the Theory of Computing, Northampton, Mass , 1970, pp 180–183.
30. ROBINSON, J. A. A machine-oriented logic based on the resolution principle *J. ACM 12*, 1 (Jan 1965), 23–41
31 ROBINSON, J. A. Automatic deduction with hyper-resolution. *Int. J. of Computer Math. 1* (1965), 227–234.
32 ROBINSON, J A A review of automatic theorem-proving Proc. of Symposia in Applied Mathematics, Vol 19, Amer. Math. Soc , Providence, R. I., 1967, pp. 1–18
33. STILLMAN, R. B The concept of weak substitution in theorem-proving. *J ACM 20*, 4 (Oct. 1973), 648–667.
34. WILKINS, D E. QUEST. A non-clausal theorem-proving system. M.Sc. Th , U of Essex, Colchester, Essex, England, 1973.
35 WINSTON, P. H The M I T Robot. In *Machine Intelligence 7*, B. Meltzer and D. Michie, Eds., Edinburgh U. Press, Edinburgh, Scotland, 1972, pp 431–463
36. WOS, L., CARSON, D. F , AND ROBINSON, G A The unit preference strategy in theorem-proving Proc. AFIPS 1964 FJCC, Vol 26, Spartan Books, New York, pp 616–621.
37. WOS, L., ROBINSON, G A , AND CARSON, D. F Efficiency and completeness of the set of support strategy in theorem proving *J ACM 12*, 4 (Oct 1965), 536–541.
38. ZAMOV, N K , AND SHARONOV, V I. On a class of strategies which can be used to establish decidability by the resolution principle (In Russian). *Issled po konstruktivnoye matematikye i matematicheskoie logikye III 16* (1969), 54–64 (National Lending Library, Russian Translating Program 5857, Boston Spa, Yorkshire), England)

RECEIVED APRIL 1974; REVISED OCTOBER 1974