# AUTOLOGIC

by
Charles G. MORGAN

## *0. Introduction*

We will assume throughout that the reader is familiar with the terminology, algorithms, and theoretical results concerning resolution based theorem provers. For details, the reader may profitably consult [10] and the more recent [9].

As usually implemented and employed, resolution based theorem provers suffer from a number of severe limitations, among which are the following: (1) Resolution assumes the logic to be classical propositional or first-order logic. Logics weaker than classical logic cannot generally be handled. (2) Resolution techniques assume that the language contains conjunction, disjunction, and negation. So generally, one cannot use resolution techniques even for a classical logic which is restricted to a different set of connectives. (3) Resolution assumes that all expressions can be put into clause form. Thus one cannot use resolution directly for various logics, e.g. modal logics, which are extensions of classical logic but for which the usual normal form theorems fail. (4) Resolution based theorem provers generally make use of only one inference rule (or some slight variations of it). Thus such routines cannot be used to search for alternate proofs using different inference rules.

In previous work, we have attempted to overcome some of these shortcomings. In [6] we developed a resolution principle suitable for any finite-valued logic whose syntax includes (i) either primitive or defined auto-descriptivity operators, and (ii) sufficient resources to define types of "conjunction" and "disjunction" permitting the formation of conjunctive and disjunctive normal forms. In [7] we developed techniques for non-classical logics which depend on embedding either the proof theory or the formal semantics in a first-order meta-language, and then treating the meta-language using normal resolution schemes. Our recent investigations are closer in spirit to those reported in [7]. It should be emphasized, however, that the

techniques described here depend only on proof theory and not at all on any formal semantics for the systems under investigation.

The program AUTOLOGIC was designed to find object language proofs of specified theorems in arbitrary axiomatic systems. We began by investigating simple propositional logics whose only inference rule is modus ponens. Such logics are seldom, if ever, classical. Thus, the usual resolution schemes are not applicable. Early in the research, we realized that the system could be easily extended to work with arbitrary axiomatic systems with arbitrary sentential operators and inference rules.

## *1. Backward proof tree generation*

For the most part, our discussion will concentrate on propositional languages. We will use x, y, and z, with and without subscripts, as sentential variables. We will use p, q, and r, with and without subscripts, as sentential constants. We will use $\supset$ as a two-place sentential operator representing an arbitrary (not necessarily classical) conditional. Other sentential operators will be introduced as required. We will use parentheses as formula punctuation, and we assume the usual formation rules. For convenience, we will frequently drop the outermost set of parentheses when writing a formula.

To be completely correct, we should talk about two distinct languages. The object language consists of those formulas which are built up solely of constants, the connectives, and parentheses; it is the language under study and in which proofs are being sought. Students given the task of deriving a theorem *in* some system of logic should come up with a sequence of object language formulas constructed according to certain rules. The meta-language is the language we use to talk about the object language. We use the meta-language when we wish to prove things *about* the system of logic under study. Certainly when talking about people, no one would confuse the name of an individual with the individual itself. But when talking about words, formulas, and other more abstract objects, many people are easily confused. In English it would not be correct to say: John is a word with four letters. Instead, English uses quotation marks as a grammatical technique to turn any word into a name for itself. So it would be

correct to say: "John" is a word with four letters. Similarly, we should not say: $p \supset (q \supset p)$ is an axiom. It would be more correct to state the claim in the following manner: "$p \supset (q \supset p)$" is an axiom. Over the years, logicians have stopped using quotation marks. Strictly speaking then, the meta-language should contain a name for each of the symbols in the object language. But for reasons of efficiency, logicians are loathe to use English words where simple symbols would do. Consequently, we have come to use the object language symbols as names for themselves in the meta-language. This usage greatly blurs the distinction between the object language and the meta-language. In most applications, no serious consequences result from our failure to maintain a strict hierarchy of languages, and we can rely on our common sense to sort out the details where required. In our case, formulas with sentential variables are formulas of the meta-language. Formulas which contain no sentential variables may be regarded as being part of either the object language or the meta-language, depending on the context.

A proof theory for a logic is generally specified by listing axioms and rules of inference. An infinite number of axioms may be specified by using sentential variables instead of sentential constants; in such a case, we use the phrase "axiom scheme" instead of "axiom". It is frequently much more convenient from a meta-theoretical point of view to use axiom schemes so that a rule of substitution need not be included with the inference rules. A rule of inference can generally be specified by indicating a set of "antecedent" formulas (or formula schemes) and the allowed "consequent" formula (or formula scheme). Just as with axiom schemes, rule schemes represent an infinite number of allowed inferences; most logics are formulated with rule schemes, and thus should be properly thought of as having an infinite number of inference patterns. So, by a logic **L**, we mean a set of axioms (or axiom schemes) and inference rules (or rule schemes), each inference rule (rule scheme) being specified by a set of antecedent formulas (formula schemes) and an allowed consequent formula (formula scheme). A proof in **L** is just a finite sequence of object language formulas, each of which is either an axiom (instance of an axiom scheme) or follows from previous members of the sequence by an inference rule. We may give the following simple definition of "theorem" of **L**:

T1. Every axiom (instance of an axiom scheme) of **L** is a theorem of **L**.

T2. Suppose that (i) $A_1,\ldots,$ and $A_n$ are all theorems if **L**; and (ii) C is the consequent formula (an instance of the consequent formula scheme) of an inference rule of **L**, the antecedent formulas (instances of the antecedent formula schemes) of which are $A_1,\ldots,$ and $A_n$. Then C is a theorem of **L**.

Trivially, a theorem of **L** is just the last line of a proof in **L**.

For an arbitrary logic **L**, the most simple-minded theorem prover would be a "backward" proof tree search. To be completely specific, we will sketch the algorithm for the generation of such a tree. Technically, our backward proof tree generation technique manipulates formulas of the meta-language. Our tree generation technique sometimes attempts to match formulas by using the unification algorithm. Readers unfamiliar with the unification algorithm may consult [9].

The root node of the backward proof tree for **L** is the proposed theorem, which is to be regarded as a single-membered "sequence". Note that since we are usually searching for a theorem *in* the logic rather than a theorem *about* the logic, the root node will usually contain no sentential variables, although we will specifically discuss exceptions below. (If it contains no variables, then the root node is a formula of the meta-language that serves as the name of the same formula in the object language.) Subsequent nodes consist of other ordered sequences of formulas. A node is expanced by considering as a «target» formula the left-most formula in the sequence. (The choice of left-most is arbitrary.) New nodes are generated in two ways:

(N.1) The target formula is checked to see if it can be unified with any of the axioms. (Any variables in the axiom are first changed so that the old node and the axiom have no common variables.) For each match found, a new node is generated by deleting the target formula from the old node sequence and making the unification substitutions in the remaining formulas of the sequence.

(N.2) The target formula is checked to see if it can be unified with the consequent of any rule of inference. (Any variables in the rule are first changed so that the old node

and the rule have no common variables.) For each match found, a new node is generated by replacing the target formula in the old node sequence by the sequence of antecedent formulas from the rule in question and making the unification substitutions in the resulting sequence.

When an empty node (here designated by $\phi$) is generated, a proof of the original theorem has been found.

Note that free variables may occur in the statement of the axioms and inference rules, and that free variables may also occur on the nodes of the tree. Free variables occurring in the *axioms* and in the *inference rules* are implicitly universally quantified. The implicit meta-theoretical claim with regard to each of the axioms is that for all values of the variables, the corresponding formula is provable. The implicit meta-theoretical claim with regard to each of the inference rules is that for all values of the variables, if the antecedent formules are provable, then the corresponding consequent formula is provable. The characterization of the free variables on the *nodes* of the tree is a bit more complicated. As long as the root node contains variables, then intuitively the claim made about each node is that for all values of the variables, if all of the formulas on the node are provable, then the formula on the root node is provable. Since the root node contains the desired theorem, the implicit claim being made is that for all values of the variables, if all of the formulas on the node are provable, then the original theorem is provable. As an example, suppose the desired theorem is T, and suppose T contains no variables, which will usually be the case for reasons explained above. Suppose the logic includes an inference rule scheme which sanctions the conclusion y from the two antecedents x and $x \supset y$. Consider a node generated by N.2; it would contain only the two formulas $x \supset T$ and x. The node actually represents the meta-theoretical claim "For all x, if $x \supset T$ and x are both provable, then T is provable." If T has no free variables, then this meta-theoretical claim is equivalent to the claim "If there is an x such that both $x \supset T$ and x are provable, then T is provable." Thus the free variables on the nodes may in some cases be regarded as in a sense being existentially quantified. But we may use the unification algorithm to match target formulas on the nodes with axioms because in another sense the free variables on the nodes are universally

quantified. We may state these considerations more formally in the following theorem.

*Theorem 1*: Suppose E is the root node of a backward proof tree for **L**, and E has free variables $x_1, \ldots, x_k$. Let $F_1, \ldots, F_m$ be all of the formulas on any node of the tree, and let $x_r, \ldots, x_s$ be all of the free variables occurring in the $F_i$. Then there is a formula E′ which is a substitution instance of E such that:

> For all $x_1, \ldots$, all $x_k$, all $x_r, \ldots$, and all $x_s$, if $F_1, \ldots$, and $F_m$ are all theorems of **L**, then E′ is a theorem of **L**.

*Proof*: The proof is by a simple induction on the levels of the tree. For the basis step, consider level 0. Trivially:

> (1.1) For all $x_1, \ldots$, and all $x_k$, if E is a theorem of **L**, then E is a theorem of **L**.

For the induction step, suppose the theorem is true of all nodes at level $n-1$; we must show that it holds at level n. So, consider an arbitrary node at level n, which has been generated from a node at level $n-1$. The inductive assumption tells us that:

> (1.2) For all $x_1, \ldots$, all $x_k$, all $y_1, \ldots$, and all $y_h$ if $G_1, \ldots$, and $G_j$ are all theorems of **L** then E″ is a theorem of **L**.

Of course E″ is a substitution instance of E, the $G_i$ are the formulas on the parent node on level $n-1$, and the $y_i$ are the free variables in those formulas. We may suppose that the node at level n contains the formulas $F_1, \ldots$, and $F_m$ with free variables $x_r, \ldots, x_s$. There are only two cases to consider: the new node was either generated by N.1 or by N.2. First consider the case of node generation by N.1. Then there must be an axiom (scheme), call it AX, such that $G_1$ and AX are unifiable. But clearly where the $z_i$ are the free variables in AX:

> (1.3) For all $z_1$, and ..., and all $z_r$, AX is a theorem of **L**.

The unification algorithm simply finds "minimal" substitutions for the variables in (1.2) and in (1.3) so that after the substitutions, AX and $G_1$ are identical. After the substituions, $G_2$ becomes $F_1, \ldots, G_j$ becomes $F_k$, and E″ becomes some substitution instance E′ of E. It

thus trivially follows from (1.2) and (1.3) that:

(1.4) For all $x_1$, ..., all $x_k$, all $x_r$, ..., and all $x_s$, if $F_1$, ..., and $F_m$ are all theorems of **L**, then E′ is a theorem of **L**.

For the second case, consider node generation by N.2. It must be the case that there is some inference rule (scheme) whose consequent is unifiable with $G_1$. Let the antecedents of the rule be $A_1$, ..., $A_s$, let the consequent of the rule be C, and let the free variables of the rule be $z_1$, ..., $z_r$. Then we know:

(1.5) For all $z_1$, and ..., and all $z_r$, if $A_1$, ..., and $A_s$ are all theorems of **L**, then C is a theorem of **L**.

Then just as with the first case, the required (1.4) follows trivially from (1.2) and (1.5).

Since every formula in $\phi$ is a theorem of **L** (there are no members of the empty sequence), the "correctness" of the backward proof tree generation technique trivially follows from Theorem 1. To make it a bit easier to state this and subsequent results, we adopt the notation "$S \in BPT(\mathbf{L}, E, n)$" to mean that the ordered sequence of formulas S is a node of the backward proof tree of logic **L**, with root node E, at level n. Unless explicitly stated otherwise, we do not assume that E has no free variables.

*Theorem 2*: Let E be an arbitrary formula, and suppose that $\phi \in BPT(\mathbf{L}, E, n)$, for some n. Then there is some substitution instance E′ of E, which contains no free variables, such that E′ is a theorem of **L**.

Note that if E has no free variables, then the only substitution instance of E is E itself. Thus, if the root node of the tree has no free variables, and the empty node occurs on the tree, then Theorem 2 guarantees that E is a theorem of **L**. Note also that if the empty node is found on a tree whose root node contains free variables, then the free variables are appropriately regarded as being existentially quantified. For example, if we place $x \supset x$ on the root node of a tree and $\phi$ is subsequently found, we know only that there is *some* formula of the form $x \supset x$ which is provable. Universal questions may be asked by formulating them using constants which do not occur explicitly in the meta-theoretical statements of the axioms and inference rules. That

is, suppose p does not occur explicitly in the statements of the axioms and inference rules. Let E, E′, and E″ be any object language formulas such that E′ is the result of uniformly substituting E″ for p in E. Then for logics of the sort discussed here (those that can be formulated using axiom schemes and inference rule schemes as discussed above), it is easy to show that if E is provable, then so is E′. Simply make the substitution in each line of the proof of E, and note that axiomhood is preserved as are applications of the inference rules.

We will now prove a sequence of theorems leading to the completeness of the backward proof tree technique. In all theorems and proofs, we treat trivial variable variants as the same formula. For convenience, we will assume that ordered sequences of formulas are written from left to right, each formula followed by ";".

*Theorem 3*: Let $E_1$ and $E_2$ be formulas, and let $S_1$ and $S_2$ be ordered sequences of formulas. If $S_1 \in \mathrm{BPT}(\mathbf{L}, E_1, k)$ and $E_1; S_2 \in \mathrm{BPT}(\mathbf{L}, E_2, m)$, then for some $n \leq k + m$ and some substitution instance $S_{2'}$ of $S_2$, it must be the case that $S_1 S_{2'} \in \mathrm{BPT}(\mathbf{L}, E_2, n)$.

*Proof*: The proof of the desired conclusion is by a trivial induction on the level k. For the basis step, suppose $k = 0$. Then $S_1$ is just the sequence $E_1;$, and the desired conclusion follows trivially from the hypothesis of the theorem. For the induction, assume the theorem holds at level $k - 1$; we must show that it holds at level k. Suppose the hypothesis of the theorem is true. Let S be the node on level $k - 1$ which is the immediate predecessor of $S_1$ on level k. The induction assumption assures us that there is some substitution instance $S_{2''}$ of $S_2$ such that $SS_{2''} \in \mathrm{BPT}(\mathbf{L}, E_2, n - 1)$. Node $S_1$ was obtained from node S by either N.1 or N.2. In either case, exactly the same step of the algorithm will obtain $S_1 S_{2'}$ from $SS_{2''}$, where of course $S_{2'}$ is a substitution instance of $S_{2''}$ and hence of $S_2$. It trivially follows that $S_1 S_{2'} \in \mathrm{BPT}(\mathbf{L}, E_2, n)$, as required.

For the proof of the next theorem we will have to be more explicit about substitutions. A single substitution may be thought of as a pair, frequently written "E/x", in which "E" is a formula and "x" is a variable. To apply such a substitution to a formula, one simultaneously and uniformly replaces all occurrences of x by the formula E. We use $\delta$, $\varrho$, $\sigma$, and $\pi$ to represent ordered sequences of substitutions.

We permit the empty sequence to be considered as an ordered sequence of substitutions. Trivially, concatenation of two ordered sequences of substitutions is an ordered sequence of substitutions. We use the notation "E$\sigma$" to mean the formula which results from applying the ordered sequence of substitutions in $\sigma$ to the formula E. We will use similar notation when applying an ordered sequence of substitutions to an ordered sequence of formulas; note that in such a case, each substitution in the substitution sequence is to be applied to every formula in the formula sequence. Where no confusion will arise, we will use "substitution" instead of "ordered sequence of substitutions".

For the next theorem, we will also need the notion of one logic being more general than another. In the definition, we regard an inference rule (scheme) as an ordered sequence of formulas (formula schemes), such that the first member is the consequent and the subsequent members are the antecedents of the rule. We will say that logic **L** is more general than logic **L'** just in case: (i) for every axiom (scheme) AX' of **L'**, there is an axiom (scheme) AX of **L**, and some substitution $\sigma$ such that $AX' = AX\sigma$; and (ii) for every inference rule (scheme) IR' of **L'**, there is an inference rule (scheme) IR of **L** and some substitution $\sigma$ such that $IR' = IR\sigma$.

*Theorem 4*: Suppose logic **L** is more general than logic **L'** and that for some n, $S' \in BPT(\mathbf{L}', E, n)$. Then there is a sequence of formulas S such that S' is a substitution instance of S and $S \in BPT(\mathbf{L}, E, n)$.

*Proof*: The proof is by induction on the level n. For the basis step, suppose $n = 0$. Then S' has E as its only member, and trivially S will also have E as its only member. For the induction step, suppose the theorem holds for level $n - 1$; we must show that it holds for level n. Let $C'; S_{p'}$ be the predecessor node on level $n - 1$ to S' on level n, where C' is some formula. Thus we know that $C'; S_{p'} \in BPT(\mathbf{L}', E, n - 1)$. By the induction hypothesis, we know that $C; S_p \in BPT(\mathbf{L}, E, n - 1)$, where for some substitution $\pi$, $C'; S_{p'} = (C; S_p)\pi$. Now, S' had to be generated by either N.1 or N.2. First suppose S' was generated by N.1. Then there is some axiom (scheme) AX' of **L'** such that after the variables of AX' are changed so that they differ from those of C', AX' and C' are unifiable. But AX' is a substitution instance of axiom

(scheme) AX of logic **L**, so there is some substitution $\varrho$ such that $AX' = AX\varrho$. Since the variables in the axioms are just treated as dummy place holders (they are always changed by the tree generation technique to be different from the variables in the node under consideration), we may without loss of generality assume that the variables of AX and AX′ are distinct from those in both C′ and C. Hence the substitutions $\pi$ and $\varrho$ are completely independent, and furthermore $C' = C\pi\varrho$ and $AX' = AX\pi\varrho$. Since C′ and AX′ are unifiable, there is a substitution $\delta$ such that $C'\delta = AX'\delta$ and $S' = S_{p'}\delta$. It follows trivially that the substitution $\pi\varrho\delta$ unifies C and AX; that is, C and AX are unifiable. Thus N.1 will generate at level n in the backward proof tree of E in logic **L** a node $S = S_p\sigma$, where $\sigma$ is a substitution found by the unification algorithm. Now, by the induction hypothesis, $S_{p'} = S_p\pi$, so $S_{p'} = S_p\pi\varrho$ since $\pi$ and $\varrho$ are completely independent and $\varrho$ has no variables in common with $S_p$. Thus it follows that $S' = S_{p'}\delta = S_p\pi\varrho\delta$. Since the unification algorithm finds a *minimal* unifying substitution for C and AX, call it $\sigma$, and we know that $\pi\varrho\delta$ is also a unifying substitution for C and AX, it follows that $S_p\pi\varrho\delta$ is a substitution instance of $S_p\sigma$; that is, S′ is a substitution instance of S. The case for N.2 is very similar. The only difference is that the sequences of rule antecedents are concatenated onto the left of $S_{p'}$ and $S_p$ before the substitutions are made to obtain S′ and S, respectively. But the rule antecedents for **L**′ are just substitution instances of the corresponding rule antecedents for **L**. The desired result will then follow by the same argument used for the N.1 case.

*Theorem 5*: Let E and E′ be any formulas such that E′ has no free variables and for some substitution $\sigma$, $E' = E\sigma$. If E′ is a theorem of **L**, then $\phi \in BPT(\mathbf{L}, E, n)$ for some n.

*Proof*: Suppose E, E′, and $\sigma$ are related as described, and suppose that E′ is a theorem of **L**. Consider the logic **L**′, which consists of the actual object language axioms cited in the proof of E′ and the instances of the rules actually used in the proof of E′. Then the axioms and rules of **L**′ contain no variables, and the proof of E′ in **L** constitutes a proof of E′ in **L**′. Further, **L** is more general than **L**′. The remainder of the proof has three main steps. (1) First we claim that $\phi \in BPT(\mathbf{L}', E', n)$ for some n. The proof of this claim is by induction

on the **L**′ proof of E′. For the basis step, suppose E′ is an axiom. Then trivially by N.1 we know that $\phi \in \text{BPT}(\mathbf{L}', E', 1)$. For the induction step, suppose that E′ follows by an inference rule. The consequent of the rule must be E′, and the antecedents of the rule must be **L**′ theorems, say $A_1, \ldots, A_k$. As hypothesis of the induction, we may assume that for each i there is an n such that $\phi \in \text{BPT}(\mathbf{L}', A_i, n)$. Further, N.2 of the generation technique assures that $A_1; \ldots; A_k \in \text{BPT}(\mathbf{L}', E', 1)$. Then using the induction hypotheses, k applications of Theorem 3 will yield $\phi \in \text{BPT}(\mathbf{L}', E', n)$ for some n. That ends the induction. (2) We claim on the basis of the previous result that $\phi \in \text{BPT}(\mathbf{L}', E, n)$ for some n. The argument for this second claim is quite simple. Note that $\phi$ cannot occur on level 0, since there is no "empty" formula. Comparing the trees for E and E′, it is easy to see that they are identical at level 1. If a node on level 1 of the E′ tree is generated by N.1, then that node must be $\phi$ and E′ must be one of the axioms. Since E′ is a substitution instance of E, it follows that E and E′ (one of the axioms of **L**′) are unifiable, and hence $\phi$ will also be on level 1 of the E tree. On the other hand, if a node on level 1 of the E′ tree is generated by N.2, then that node must be $A_1; \ldots; A_k$ and **L**′ must have a rule with E′ as consequent and the $A_i$ as antecedents. But again, since E′ is a substitution instance of E, E and the consequent of the same rule are unifiable. Since the $A_i$ have no free variables, the node on the E tree will just be $A_1; \ldots; A_k$, the same as the corresponding node on the E′ tree. So the two trees are identical at level 1. Since they are identical at level 1, they are identical at all lower levels. Hence $\phi \in \text{BPT}(\mathbf{L}', E, n)$ for some n, as claimed. (3) Finally, we obtain the desired result that $\phi \in \text{BPT}(\mathbf{L}, E, n)$ by applying Theorem 4 to the previous claim and noting that $\phi$ is the only substitution instance of $\phi$.

Theorem 5 is the desired general completeness result for backward proof tree generation. As a special case of the theorem, note that if E is a formula with no free variables that is a theorem of **L**, then the empty node will be generated at some level of the tree with root node E. Thus backward proof tree generation can be used to determine whether or not a particular formula is a theorem. As noted above, it also follows that the technique can be used to check "universal" claims about theoremhood, since for the type of logics we are considering, substitution of a formula for a "non-special" sentential

constant in an **L** theorem is always an **L** theorem. In the general case, note that if E has free variables and some substitution instance is an **L** theorem, then the empty node will be generated at some level of the tree with root node E. So Theorem 5 justifies our earlier claim that free variables on the root node may be regarded as existentially quantified. In short, simple meta-theoretical claims may be checked using the tree method by applying a "Skolemization" process to the meta-theoretical claim, replacing universally quantified variables by constants and existentially quantified variables by free variables.

## 2. *Modus ponens as the only inference rule*

The inference rule known as modus ponens ("detachment of the conditional", "implication elimination") sanctions the inference from a conditional, "if A, then B", and its antecedent, "A", to its consequent, "B". A great many logics are normally presented as a set of axioms (or axiom "schemes") with modus ponens as the only inference rule. Examples include classical logic [5], various "partial" systems [2], quite a few many-valued logics [8], intuitionistic logic [3], several modal logics [4], and various systems of "strict" implication [1]. Sometimes these systems are formulated with modus ponens *and* a rule of substitution. In such cases, the rule of substitution can be eliminated by replacing axioms with axiom "schemes", i.e., by allowing an infinite number of axioms represented by formulas using sentential variables instead of sentential constants. For the reasons mentioned above, resolution based theorem provers are not really suitable for the investigation of such logics. While investigating logics with only modus ponens, we incorporated an additional node generation method which sometimes greatly improved the general technique. We will discuss the new method in this section.

If modus ponens is the only inference rule of **L**, then we may give the following simple definition of "theorem" of **L**:

MPT1. Every axiom (instance of an axiom scheme) of **L** is a theorem of **L**.

MPT2. If x is a theorem of **L** and $x \supset y$ is a theorem of **L**, then y is a theorem of **L**.

We will use this characterization in the proof of the theorem below. The theorem was used as a basis for an extended version of modus ponens which is incorporated in AUTOLOGIC. For reasons which will become obvious, we call the extended version "axiom chain" modus ponens.

*Theorem 6*: If B is a theorem of logic **L**, which has modus ponens as its only rule of inference, then either (a) B is an axiom of **L**, or (b) there is an axiom of **L** which has the following form:

$$E_1 \supset (\ldots \supset (E_n \supset B)\ldots)$$

where each of the $E_i$ is a theorem of **L**.

*Proof*: The proof is by an easy induction on the set of theorems of **L**, as defined in MPT1 and MPT2, above. The basis part of the proof is trivial, since every axiom of **L** is characterized by (a) above. For the induction, we assume that (i) B follows by modus ponens from the two **L** theorems $A \supset B$ and A, (ii) either (a) or (b) is true of A, and (iii) either (a) or (b) is true of $A \supset B$. Consider inductive assumption (iii). If (a) is true of $A \supset B$, then (b) is true of B, since the inductive assumption (i) assures that A is a theorem of **L**. On the other hand, suppose (b) is true of $A \supset B$. Then there is an axiom of **L** which has the following form:

$$E_1 \supset (\ldots \supset (E_n \supset (A \supset B))\ldots)$$

where each of the $E_i$ is a theorem. But inductive assumption (i) guarantees that A is a theorem, so (b) is true of B.

Briefly put, Theorem 6 states that for any logic **L** whose only rule of inference is modus ponens, every theorem of **L** which is not itself an axiom of **L** must occur as the consequent of a conditional chain which is an axiom of **L** and each of whose antecedent formulas is an axiom of **L**.

While generating a backward proof tree, if modus ponens is among the inference rules of the logic and N.2 is applied to a given node n times in a row, then one of the formulas on the resulting node will have the form:

$$x_1 \supset (\ldots \supset (x_n \supset A)\ldots)$$

Applying step N.1 to such a formula as target assures that the general backward proof tree search examines instances of the sort anticipated by Theorem 6. However, the examination of such instances requires the expansion of the tree to some considerable depth. In certain cases, some efficiency may be gained by adding the following node generation technique to the tree generation procedure.

N.3 Apply the following algorithm to each axiom:

a. Set F to be the present axiom under consideration. (Change the variables in F to be distinct from those in the node being considered.) Set S to be the sequence of formulas in the node being considered for expansion, minus the target formula. Go to step b.

b. If F is not a conditional, then exit. But if F is a conditional, break it into its antecedent and consequent, designated by ANT and CON, respectively. Add ANT to the sequence of formulas in S. Go to step c.

c. Check to see if the target formula and CON are unifiable. If so, then form a new node consisting of the formulas in S with the required unifying substitution. In either case, go on to step d.

d. Set F to be CON. Go to step b.

This routine simply checks to see if the target formula can be regarded as the right-most formula of a conditional chain which is an instance of an axiom. If so, then a new node is generated with the target formula replaced by the appropriate set of antecedent formulas. Note that the maximum "length" (in the sense of "number of antecedents") of a conditional chain that is checked by N.3 is the maximum number of occurrences of $\supset$ in any axiom. However, in conjunction with N.2, "new" target formulas will be generated at subsequent levels with an arbitrary number of "vacuous" antecedents. These "new" targets will also be tried as consequents in conditional chain axioms. So, N.3 will not generate nodes that would not eventually be generated anyway. That is, as long as the inference rules of the logic *include* modus ponens, then Theorem 2 (correctness) and Theorem 5 (completeness) both hold even when the tree generation technique includes N.3.

The value of N.3 is that it anticipates nodes which would otherwise be generated only at greater depths in the search. For a simple example, we will consider a standard problem in partial propositional logics. Suppose we wish to derive $p \supset p$ from the following axiom schemes, using only the rule of modus ponens.

AX.1 $x \supset (y \supset x)$
AX.2 $(x \supset (y \supset z)) \supset ((x \supset y) \supset (x \supset z))$

Using only N.1 and N.2, the interesting branch on the backward proof tree would look like the following:

| | | |
|---|---|---|
| level 0 | $p \supset p$ | root |
| level 1 | $x_1 \supset (p \supset p); x_1$ | N.2 |
| level 2 | $x_2 \supset (x_1 \supset (p \supset p)); x_2; x_1$ | N.2 |
| level 3 | $p \supset (y \supset p); p \supset y$ | N.1, axiom 2 |
| level 4 | $p \supset y$ | N.1, axiom 1 |
| level 5 | $\phi$ | N.1, axiom 1 |

However, if N.3 is included in the tree generation algorithm, then the following branch occurs:

| | | |
|---|---|---|
| level 0 | $p \supset p$ | root |
| level 1 | $p \supset y; p \supset (y \supset p)$ | N.3, axiom 2 |
| level 2 | $p \supset ((z \supset p) \supset p)$ | N.1, axiom 1 |
| level 3 | $\phi$ | N.1, axiom 1 |

Thus, including N.3 would result in a saving in terms of the number of levels in the tree that must be generated. We refer to this phenomenon as level saving.

Our example is not at all misleading with regard to the general case. Given Theorem 6, it is easy to see that if modus ponens is the only inference rule, then the inclusion of N.3 will *always* result in some efficiency in terms of the number of levels required to be generated. Suppose A is the desired **L** theorem. Then by Theorem 6, there is an instance of an axiom of **L** which has the following form:

$$E_1 \supset (\ldots \supset (E_n \supset A) \ldots)$$

where each of the $E_i$ is an **L** theorem. Step N.3 will search for such axioms prior to the level at which this general form would be generated as a target formula by N.2 alone. The precise number of

levels "saved" will depend on the number of occurences of $\supset$ in the corresponding axiom (scheme). In addition to the level saving in the search for A, there may also be level savings in the search for the other theorems required in the proof of A, namely the $E_i$. For any $E_i$ which is not an (instance of an) axiom (scheme), there will be level saving in the search for its proof. It should be clear that level saving is additively cummulative.

It should be noted that when the logic under investigation includes rules other than modus ponens, then Theorem 6 may not hold. As a simple example, consider the following logic:

Axioms: $\sim(p \supset r) \supset \sim q$
$q$
$p$
Rules: Infer x from $\sim x \supset \sim y$ and y.
Infer x from $y \supset x$ and y

It is easy to see that r is a theorem of this logic but it does not occur on the right of a conditional chain axiom. So when rules other than modus ponens are included, there is no guarantee that N.3 will result in any level saving at all. In the example above, N.3 would not add any new nodes. But if we include an axiom scheme like $x \supset (y \supset x)$, then more nodes will be generated by N.3. So there are many pathological examples to show that inclusion of N.3 sometimes leads to an inefficiency when the logic includes rules other than modus ponens.

## 3. *The efficiency of level saving*

Of course there is a price to be paid for the level saving due to N.3. The price is the increase in the number of nodes generated at each level. We need to consider the general question of whether or not an increase in the number of nodes generated at each level is worth a decrease in the number of levels that must be generated. We will briefly discuss both time-like and space-like efficiency considerations.

If time is the major concern, then the total number of nodes generated is an appropriate statistic to consider. For an arbitrary problem, let N be the average number of new nodes generated per old

node considered, and let L be the number of levels searched, both using some arbitrary tree generation scheme, call it "TGS.1". Let M be the average number of *additional* nodes generated per old node when some other tree generation scheme, call it "TGS.2", is used; i.e., when TGS.2 is used, on average M more new nodes per old node will be generated than when TGS.1 is used, resulting in an average of N+M new nodes per old node. Let S be the total level saving from using TGS.2 as opposed to TGS.1; i.e., using TGS.2, only L-S levels are generated. In a real example, the process would be terminated sometime during the generation of the last level, but since there is no way to tell how much of the last level will actually be required, we will assume the entire level is generated. We will use the notation "n exp m" to represent n raised to the m power. The total number of nodes generated using only TGS.1 will be:

$$\text{TotNodes(TGS.1)} = (1 - N \exp (L + 1))/(1 - N)$$

The total number of nodes generated using TGS.2 will be:

$$\text{TotNodes(TGS.2)} = \frac{(1 - (N + M) \exp (L - S + 1))}{(1 - (N + M))}$$

Consequently, if TGS.2 is to result in an efficiency in terms of total number of nodes generated, then we must have:

$$\text{TotNodes(TGS.2)} < \text{TotNodes(TGS.1)}$$

In the situation of interest to us at the present time, we are comparing tree generation using N.1 and N.2 alone with tree generation using N.1, N.2, and N.3. In the worst case the only level saving will be from a single anticipated conditional chain, and that saving may be only one level. However the increase in the number of nodes per level will be on the order of the sum of the number of occurrences of $\supset$ over all the axioms. So in the worst case, the inclusion of N.3 will not result in any efficiency. Happily, experience has shown that the worst case is seldom the real case.

For the logics we have studied, N is generally no greater than about 5, while M is of the order of the number of axioms, and often smaller. The best way to get a feel for the possible efficiencies is to examine Table 1. We have tabulated values of N and L, along with the total number of nodes that would be generated by TGS.1. Then for values

of level saving (designated by S) from 1 up to 4, we have calculated a maximum increase in the average number of new nodes generated per old node (i.e., maximum values of M) if TGS.2 is to be more efficient than TGS.1.

TABLE 1: Time-like Considerations

| N | L | totnodes | maximum values of M for values of S S = 1 | S = 2 | S = 3 | S = 4 |
|---|---|---|---|---|---|---|
| 2 | 5 | 63 | 0.48 | 1.57 | 5.39 | 60.00 |
| | 7 | 255 | 0.29 | 0.77 | 1.69 | 3.97 |
| | 10 | 2,047 | 0.18 | 0.43 | 0.79 | 1.36 |
| | 15 | 65,535 | 0.11 | 0.24 | 0.41 | 0.62 |
| | 20 | 2.10E + 06 | 0.08 | 0.17 | 0.28 | 0.40 |
| 3 | 5 | 364 | 1.07 | 3.77 | 15.56 | 360.00 |
| | 7 | 3,280 | 0.65 | 1.82 | 4.29 | 11.51 |
| | 10 | 88,573 | 0.41 | 1.01 | 1.93 | 3.49 |
| | 15 | 2.15E + 07 | 0.25 | 0.57 | 0.99 | 1.54 |
| | 20 | 5.23E + 09 | 0.18 | 0.40 | 0.66 | 0.98 |
| 4 | 5 | 1,365 | 1.80 | 6.74 | 32.44 | 1360.00 |
| | 7 | 21,845 | 1.10 | 3.16 | 7.89 | 23.61 |
| | 10 | 1.40E + 06 | 0.69 | 1.72 | 3.39 | 6.40 |
| | 15 | 1.43E + 09 | 0.43 | 0.98 | 1.70 | 2.70 |
| | 20 | 1.47E + 12 | 0.31 | 0.68 | 1.13 | 1.69 |
| 5 | 5 | 3,906 | 2.63 | 10.40 | 56.99 | 3900.00 |
| | 7 | 97,656 | 1.60 | 4.74 | 12.42 | 40.71 |
| | 10 | 1.22E + 07 | 1.01 | 2.55 | 5.14 | 10.00 |
| | 15 | 3.81E + 10 | 0.62 | 1.43 | 2.53 | 4.06 |
| | 20 | 1.19E + 14 | 0.45 | 0.99 | 1.67 | 2.51 |

From the table, three trends should be quite clear: (1) The greater the level generated, the smaller the value of M for which a given level saving actually results in an efficiency. (2) For any given level of generation, the greater the level saving, the larger the value of M can be and still result in an efficiency. (3) The greater the value of N, the greater M can be and still result in an efficiency, all other things being equal.

With regard to trends (1) and (2), one should bear in mind that generally speaking, the longer a proof (i.e., the greater the value of L), the more likely it is that there will be several "opportunities" for level saving. Thus the longer the proof, the more likely it is that the level saving will be great. So it would *not* be appropriate to conclude that adding N.3 to N.1 and N.2 would be unwise if the expected length of proof is large. However, with regard to trend (3), one should be very careful in using N.3 if the value of N is quite small.

If considerations of space are of primary importance, then the total number of nodes generated may not be the appropriate statistic to consider. In order to generate the nodes at level L, it is not necessary to have all previously generated nodes available; rather, one requires only the nodes from level $L-1$. So the number of nodes that must be stored in order to generate the nodes at level L is the number of nodes generated at level $L-1$. A measure of the space requirements for TGS.1 is then given by the following:

$$\text{StrdNodes}(\text{TGS.1}) = N \exp (L-1)$$

A measure of the space requirements for TGS.2 is given by the following:

$$\text{StrdNodes}(\text{TGS.2}) = (N+M) \exp (L-S-1)$$

In order for the use of TGS.2 rather than TGS.1 to result in an efficiency, we must have the following:

$$\text{StrdNodes}(\text{TGS.2}) < \text{StrdNodes}(\text{TGS.1})$$

In Table 2, we have tabulated values of N and L, along with corresponding values for the number of nodes at level $L-1$. Then for values of level saving, S, from 1 through 4 we have tabulated the maximum values of M for which the level saving strategy results in an efficiency. Table 2 is *very* similar to Table 1, exhibiting the same trends noted before. In fact, the maximum allowed values of M do not in general differ by very much. So we can draw the same conclusions from Table 2 as we did from Table 1.

TABLE 2: Space-like Considerations

| N | L | strnodes | maximum values of M for values of S S = 1 | S = 2 | S = 3 | S = 4 |
|---|---|---|---|---|---|---|
| 2 | 5 | 16 | 0.52 | 2.00 | 14.00 | ——— |
| | 7 | 64 | 0.30 | 0.83 | 2.00 | 6.00 |
| | 10 | 512 | 0.18 | 0.44 | 0.83 | 1.48 |
| | 15 | 16,384 | 0.11 | 0.24 | 0.42 | 0.64 |
| | 20 | 524,288 | 0.08 | 0.17 | 0.28 | 0.41 |
| 3 | 5 | 81 | 1.33 | 6.00 | 78.00 | ——— |
| | 7 | 729 | 0.74 | 2.20 | 6.00 | 24.00 |
| | 10 | 19,683 | 0.44 | 1.11 | 2.20 | 4.22 |
| | 15 | 4.78E + 06 | 0.26 | 0.60 | 1.05 | 1.66 |
| | 20 | 1.16E + 09 | 0.19 | 0.41 | 0.69 | 1.02 |
| 4 | 5 | 256 | 2.35 | 12.00 | 252.00 | ——— |
| | 7 | 4096 | 1.28 | 4.00 | 12.00 | 60.00 |
| | 10 | 262,144 | 0.76 | 1.94 | 4.00 | 8.13 |
| | 15 | 2.68E + 08 | 0.45 | 1.04 | 1.84 | 2.96 |
| | 20 | 2.75E + 11 | 0.32 | 0.71 | 1.19 | 1.79 |
| 5 | 5 | 625 | 3.55 | 20.00 | 620.00 | ——— |
| | 7 | 15,625 | 1.90 | 6.18 | 20.00 | 120.00 |
| | 10 | 1.95E + 06 | 1.11 | 2.92 | 6.18 | 13.12 |
| | 15 | 6.10E + 09 | 0.66 | 1.54 | 2.76 | 4.52 |
| | 20 | 1.91E + 13 | 0.47 | 1.04 | 1.76 | 2.68 |

*Other speed-up techniques*

Space limitations prohibit the detailed discussion of the wide variety of speed-up techniques which might prove useful in various situations. We will briefly mention a few of those with widest applicability.

In many situations, one is only concerned to determine whether or not a given expression is an **L** theorem, without caring about the details of the proof. In such situations, the user may be familiar with other theorems and derived inference rules which might be relevant or useful for the problem at hand. The backward proof tree technique is

quite amenable to the inclusion of such information. In fact, in our implementation, AUTOLOGIC, no distinction is made between primitive and derived inference rules, nor between axioms and other known theorems ("derived" axioms). For just one example, suppose we know that the deduction theorem holds for our logic and we wish to check a formula of the form $A \supset B$ for theoremhood. Then we may add A to our list of "axioms" and look for a proof of B.

There is a simple technique for reducing the required level of tree generation by one, at no increase in the number of new nodes per old node generated. Whenever a node with a single formula is generated, it may be checked against the axioms (step N.1) immediately, rather than waiting till the next level. If the node and some axiom are unifiable, then a proof has been found and tree generation may cease. On the other hand, if the node is not unifiable with any axiom, then no additional node has been generated. Of course implementation of such a procedure means that single formula nodes need not be checked against the axioms at the next level. Now, it is easy to see that as we have described the tree generation algorithm, the empty node can be generated only from a single formula node and only using N.1. Hence, the procedure described here will always result in a level reduction of one, regardless of the logic under study. For obvious reasons, we call this technique the "unit node" technique.

A reduction in the number of nodes which must be generated on the last level can be obtained by ordering the nodes at the previous level before beginning to generate the nodes at the next level. By considering the nodes from the previous level with the fewest number of formulas first, the empty node will in general be produced sooner on the last level. We call this procedure the "level ordering" technique.

In some cases it is possible to eliminate one (or more) formula(s) from a node. If that node is on a branch leading to the empty node, elimination of the excess formula will reduce by one the level to which the tree must be expanded. The simplest example of a redundant formula on a node is one which is exactly the same as some other formula on the node. The more general characterization is as follows: A formula E occurring on a node is "redundant" just in case there is a substitution for the variables occurring in E but not occurring in any other formula on the node, such that the resulting formula $E'$ is identical to some other formula on the node. The redundancy elimi-

nation technique simply eliminates any redundant formulas from each node as it is generated.

Theoretically, a vast improvement in speed can be obtained by judicious pruning of the tree. One of the standard tree pruning techniques in resolution routines is some version of "subsumption" (see [9]). The same techniques work with backward proof trees. Node 1 is said to subsume node 2 just in case there is some substitution that can be applied to node 1 such that the resulting sequence of formulas contains every formula in node 2. If a new node is subsumed by an old node then it can be shown to be redundant and may be pruned from the tree.

There are several problems with subsumption pruning. First of all, in its most general form, subsumption pruning requires the storage of all nodes generated, which may be quite prohibitive in terms of the space required. Secondly, the time required to do the subsumption check goes up with the number of nodes generated. In our experience, the time required becomes prohibitive after only a few levels. There are three special cases of subsumption pruning which avoid these difficulties to varying degrees. Previous level subsumption pruning checks each new node only against the nodes from the previous level. Parent node subsumption pruning checks each new node against only its parent node. Original node subsumption pruning checks each new node against the originally proposed theorem. In our experience, parent node and original node subsumption pruning are much more "cost effective" than either general or previous level subsumption pruning.

Another pruning technique which fits easily and naturally with the backward tree generation scheme is non-theorem pruning. Very often an investigator knows a number of non-theorems of the system under study. For example, we may know that $p \supset q$ is not a theorem of the logic being investigated. Or we may know that no formula of the form $x \supset x$ is a theorem. Any node that contains a non-theorem, or a substitution instance of a scheme of non-theorems, may be pruned from the tree.

Completeness and correctness are easy to show for all of the speed-up techniques discussed above. Space limitations prohibit giving the details here.

## 5. *Implementation*

We could have represented formulas in the usual way in AUTOLOGIC. However, such representation does not allow for every efficient processing, so an alternative notation was adopted. In AUTOLOGIC, the conditional is the only sentence connective which is intrinsically recognized. The symbol ">" is used to represent the conditional. The program recognizes 20 two-character sequences as variables: "V0"-"V9" and "VA"-"VJ". It also recognizes 20 two-character sequences as constants: "K0"-"K9" and "KA"-"KJ". Left and right parentheses are used as object language punctuation for ease in parsing formulas, but the formation rules for formulas are slightly non-standard, as described below. The semicolon ";" and the slash "/" are used as special delimiters for the representation of strings of formulas and inference rules, as described below. All other characters (and sequences of them) are available for use as sentential connectives or as predicate and function expressions. These additional symbols will be said to be "non-special".

The formation rules for *formulas* are as follows:

F1. Any variable or constant is a formula.

F2. If $E_1$ and $E_2$ are formulas, then the following is a formula: $(E_1 > E_2)$

F3. If c is any non-special character or sequence of non-special characters and $E_1, \ldots, E_n$ are any formulas, then the following is a formula: $(cE_1 \ldots E_n)$

Thus, additional sentence operators (or predicates and functions in a first-order language) are treated as in Polish notation, with the addition of parentheses around the entire expression. The addition of the parentheses permits the program to correctly parse an arbitrary string without specific information concerning the polyadicity of the operator (or predicate or function).

Obviously there was no real need to include a "special" conditional symbol and to treat the formation of conditionals in a way different from the formation of other compounds. Our only reason for doing so was the fact that the original routine was specifically designed to investigate conditional logics. Axiom chain modus ponens allows for

efficient chaining of modus ponens steps (as described above), so we decided to incorporate this efficiency in the general routine. The use of this intrinsic rule may be avoided by simply not using the symbol ">".

In AUTOLOGIC, nodes on the proof tree are represented as *strings* of formulas. The simplest string is the empty string, whose internal representation is just an alphanumeric string of length 0, although for convenience, we have used $\phi$ to represent the empty string in this paper. The formation rules for *strings* are the following:

S1. $\phi$ is a string.
S2. If S is any string and E is any formula, then the following is a string: SE;

So a string is just a sequence of formulas, each formula followed by a semicolon.

Inference rules are represented by the notation "C/S", where C is a formula and S is a string of formulas. The string S represents the premises of the rule, while C represents the allowed conclusion. For example, the simplest version of the rule modus ponens could be represented as "VB/(VA > VB); VA;".

An axiom "A" could be represented as the rule "A/$\phi$", since an axiom is really just an inference rule requiring no premises. However, since humans usually do not think of axioms as being special inference rules, we decided to allow the user to specify rules and axioms in two separate categories.

At the beginning of each run, the user is asked to input the axioms, the inference rules, any non-theorems, and the proposed theorem for which a proof is sought. The user is then allowed to select from among several options for speed-up techniques, of the sort discussed above. AUTOLOGIC then begins to generate nodes of the backward proof tree, searching for the empty node.

AUTOLOGIC has been implemented in several versions in different hardware environments, including a version for the IBM PC! Our experience indicates that implementations on microcomputers are not very useful unless the hardware includes devices for substantial mass storage.

We have made a few elementary comparisons between AUTOLOGIC and the techniques in [7]. (We would like to thank Larry Wos and his associates at Agonne National Laboratories for making some long runs using their very efficient resolution based system.) On the few sample problems we tried, AUTOLOGIC PERFORMED AS WELL AS THE RESOLUTION SYSTEM WHICH WAS RUNNING IN A MUCH MORE POWERFUL HARDWARE ENVIRONMENT. However, it must be emphasized that the test base was very small indeed.

Just as with human investigators, the more information that is supplied to AUTOLOGIC in the way of non-theorems, theorems, and derived rules the more likely the routine is to find desired proofs. On the other hand, AUTOLOGIC almost never finds "disproofs". If no new nodes are produced at a given level, then there is no proof of the formula on the root node. However, the resources available in most logics are such that it is almost never the case that no new nodes are generated. For example, with the rule modus ponens available, a new node will always be generated from an old node, the new node containing additional formulas of the form $x \supset A$ and x, where A is the target formula. Unless such a node is pruned by one of the speed-up techniques, AUTOLOGIC would theoretically go on generating new nodes forever if the empty node never turns up. However, because of time and space restrictions, we have never been able to generate a tree for any reasonable logic beyond about level 7 or 8 – see Table 2.

We are presently investigating the extension of these techniques to predicate logics with general quantifiers. Even in its present formulation, variables can range over predicate and function expressions. However, the desired extension will require incorporation of a special "substitution" function intrinsic to the routine so that quantifier rules may be input by the user.

*University of Victoria*
Department of Philosophy
Victoria, British Columbia
Canada V8W 2Y2

Charles G. MORGAN

## REFERENCES

[1] Anderson A.R., and Belnap N.D., Jr. *Entailment: The Logic of Relevance and Necessity*. Princeton University Press, Princeton (1975).

[2] Church A. *Introduction to Mathematical Logic*. Princeton University Press, Princeton (1956).

[3] Fitting M.C. *Intuitionistic Logic, Model Theory, and Forcing*. North-Holland Publishing Co., Amsterdam (1969).

[4] Hughes G.E., and Cresswell M.J. *An Introduction to Modal Logic*. Methuen and Co., Ltd., London (1968).

[5] Mendelson E. *Introduction to Mathematical Logic*. D. Van Nostrand Company, Inc., Princeton (1966).

[6] Morgan C.G. "A resolution principle for a class of many-valued logics," *Logique et Analyse*, no. 74-75-76 (1976), pp. 311-339.

[7] Morgan C.G. "Methods for automated theorem proving in nonclassical logics," *IEEE Transactions on Computers*, vol. C-25 (1976), pp. 852-862.

[8] Rescher N. *Many-Valued Logic*. McGraw-Hill Book Company, New York (1969).

[9] Wos L., Ross Overbeek, Ewing Lusk, and Jim Boyle. *Automated Reasoning: Introduction and Applications*. Prentice-Hall, Inc., Englewood Cliffs (1984).

[10] Yasuhara, Ann. *Recursive Function Theory & Logic*. Academic Press, New York (1971).