EXPERIMENTAL LOGIC

and

THE AUTOMATIC ANALYSIS OF ALGORITHMS

Dr. Frank M. Brown

TR-83-16 August 1983

EXPERIMENTAL LOGIC

and

THE AUTOMATIC ANALYSIS OF ALGORITHMS

Dr. Frank M. Brown

Dept. Computer Sciences

The University of Texas at Austin

Austin, Texas 78712

(512)471-7316

CS.BROWN@UTEXAS-20

Table of Contents

1. INTRODUCTION

Experimental Logic can be viewed as a branch of logic dealing with the actual construction of useful deductive systems and their application to various scientific diciplines. In a sense it is a reversion of the study of logic back to its original purpose, before the study of logic became merely the metamathematical study of artificial language systems.

In this paper we descibe an experimental logic called Quantified Computational Logic(ie QCL) and an automatic theorem prover called the SYMbolic EVALuator(ie SYMEVAL) which automatically makes deductions in the QCL language. This logic is being applied to solving problems in several areas of computer science such as automatic complexity analysis of computer programs, automatic verification of the correctness of computer programs, automatic natural languge analysis, and as a model for advanced language design.

2. SYMEVAL AND QUANTIFIED COMPUTATIONAL LOGIC

Quantified Computational Logic is an experimental theorem proving and programming language interpreted by the SYMbolic EVALuator automatic theorem prover. This theorem prover is based on the following fundamental principle about deduction:

The Fundamental Deduction Principle Almost all steps in an automatic deduction should be viewed and usually take place by a method of replacement of expressions by equivalent expressions, and that the smaller such expressions are, the better. A few important steps may involve generalization of the theorem, but these are deliberate steps taken with due consideration - not the mindless application of basic generalizing inference rules.

We have come to believe this principal for two basic reasons:

1. The first reason is our overall impression of the many automatic theorem provers we have personally constructed in the last decade, that the closer we adhere to the above principle the better is the resulting automatic theorem prover because of the lack of redundancy in representing the same information throughout the proof process.

2. The second reason is that theorem provers have an important role to play in inductive reasoning, and that this role takes the form of using the theorem prover to simplify complex expressions containing unknown free variables into equivalent sentences which explicitly state the values of those variables. Note that if the resulting sentence were not equivalent because of a generalization step during the course of the deduction, then even if the resulting sentence were a solution to the unknown, there would still be no assurance that that solution were the only solution. Thus, a generalizing step would not produce all the the needed inductive information.

SOME SIMPLE CONSEQUENCES

The consequences of this principle are startling, and frankly, at first we were loathe to accept them because they seem to contradict much of the past and current research on automatic theorem proving(including our own). For example, almost all past research on automatic theorem proving including both resolution systems and sequent calculas(ie natural deduction) systems, has been based on the Prawitz-Robinson Unification algorithm[Prawitz,Robinson] which does not generally satisfy our principle. Consider the example where we have an axiom (F X 2)=(G X), and a theorem (IMPLIES(F 1 Y)(P Y). Then by unifying (F X 2) with (F 1 Y), binding X to 1 and Y to 2, and replacing (F 1 2) by (G 1) we can deduce the expression: (IMPLIES(G 1) (P 2)). But this expression is not generally equivalent in the theory to the original theorem because, although (F 1 2)=(G 1 2), (IMPLIES(F 1 2)(P 2)) is only an instance of (IMPLIES(F 1 Y)(P Y)).

A second consequence is that Robinson's general resolution[Robinson] inference rule(even ignoring the previously mentioned unification problem) does not generally satisfy this principle. For example, in the case of propositional logic, the resolution inference rule says that (C OR D) may be inferred from (L OR C) and ((NOT L) OR D). Since (C OR D) is not equivalent to either of the above expressions or their conjunction, it cannot generally satisfy this principle. (Note however, that there are certain cases of the resolution rule which do satisfy this principle. One such case is the in unit resolution of propositional logic where L or rather L=T is an axiom of our theory, and ((NOT L)OR D) is the expression being simplified. In this case, the resolvent is D, which is equivalent to the original expression being simplified. Unit resolution, sometimes called "forward chaining" or "backward chaining" in natural deduction theorem provers, has been found to be especially useful in certain theories[Bledsoe1,2,Pastre].)

From these example consequences, one can well imagin the difficulty of constructing a useful deductive system satisfying this principle. However, after much perserverence we have constructed a useful(but incomplete) deductive system for QCL which satisfies this principle. This deductive system is the SYMMETRIC LOGIC whose rules are described later.

This principle also has consequences with respect to the richness of the logical language used by a theorem prover. In particular, richer logical languages, such as those which include unrestricted quantifiers seem to have an advantage in being able to express axioms in accordance with this principle. One good example of this is the ELIM rule of the SYMMETRIC LOGIC which is stated schematically:

```
(EQUAL(h X Z1...Zn)
      (EX I1...Im (AND(EQUAL(const Z1...Zn I1...Im)X)
                      (c Z1...Zn I1...Im))) )
```

The closest one can write axioms of this form in a quantifier free language such as[Boyer1] seems to be:

```
(IMPLIES(h X Z1...Zn)
        (AND(EQUAL(const Z1...Zn(d1 X Z1...Zn)...(dm X Z1...Zn))X)
                  (c Z1...Zn(d1 X Z1...Zn)...(dm X Z1...Zn))) )
```

which can be obtained from ELIM by replacing the existential quantifiers (EX Ii(p Ii)) by the skolem function selectors (di Z1...Zn) and reducing the EQUAL to an IMPLY. The problem with the latter axiom scheme is that its use may cause a generalizing step to be made in the course of the proof. Another problem is that the introduction of the extra (di Z1...Zn) expressions necessitates the use of a later generalization to get rid of them, thus again causing another generalization. Finally, because the:

```
(c Z1...Zn(d1 X Z1...Zn)...(dm X Z1...Zn))
```

is logically implies: (c Z1...Zn I1...Im) it follows that it might be true without (c Z1...Zn I1...Im) being true. In such a case the needed generalization expression c would have to be explicitly given to the system.

CONSEQUENCES FOR ADVANCED PROGRAMMING LANGUAGE DESIGN We believe that our fundamental deduction principal applies also to deducing purely computational theorems such as those which are deduced when an automatic theorem prover is used as an interpreter of programs written in a logic such as QCL. Because linear Horn Clause Resolution theorem provers such as[Kowalski,Chester,Colmerauer], and SL Resolution theorem provers such as MOORE's Baroque[Moore] contradict this principal both with respect to unification and the resolution rule, it follows that such systems will not produce all the possible answers to a given problem.

2.1 LINGUISTIC CONVENTIONS OF SYMEVAL

The language of the SYMbolic EVALuator consists of constant symbols and variable symbols. There are two types of constant symbols: FUNSYMs and ATOMSYMs. There are also two types of variable symbols: variables and SCHEMATORs. Constant symbols are analogous to nouns and verbs in a natural language, whereas variable symbols are analogous to pronouns.

VARIABLES A variable symbol is any literal atom LITATOM which in not an ATOMSYM not occuring after a left parenthesis. For example X is not a variable, but Y is a variable in (X Y).

SCEMATOR A schemator is any list of the form: (schemator.symbol arg1 ... argN) whose schemator symbol is on the SCHEMATOR list. The initial schemator symbols are: SCH1 SCH2 SCH3 SCH4.

Every constant symbol is either a function symbol or an atom symbol. Some function symbols are quantifiers or modal symbols. The list of primitive, defined, and declared symbols of each type can be obtained by typing the following global variables:

FUNSYM List of all currently defined functional symbols. A FUNSYM symbol f applied to N arguments is written as (f arg1...argN). We call this a function value of f. The number of arguments may be zero in which case the function value is written (f). Some example functions in QCL are: IF,EQUAL,ALL,EX,LAMBDA,APPLY,QUOTE,QCL.

ATOMSYM List of all currently defined atom symbols. An atom symbol x is always written as x without parentheses. Some example ATOMSYMs in QCL are: NIL,T.

QUANTSYM List of all currently defined quantifier symbols. All quantifiers are of the form: (q v (f v) x1...xn) . Quantifiers must contain their associated bound variable in their first argument position. This variable is bound only in the first and second argument positions. Additional quantifiers may be defined, declared, or axiomatized. All such additional quantifiers should be added to the QUANTSYM list: (SETQ QUANTSYM(CONS new.quantifier QUANTSYM)). Some example primitive quantifiers in QCL are: ALL,EX,LAMBDA.

MODALSYM List of all currently defined modal symbols. An example modalsym in QCL is: QCL.

2.2 SYMEVAL

The SYMbolic EVALuator is a general interpreter of the Frege's quantificational logic[Frege] (ie. first order logic with schemators, but not higher order logic). Higher order logic is handled by axiomatizing it within first order logic. The actual symbols and inference rules of the logic are arbitrary as far as SYMEVAL is concerned. SYMEVAL evaluates a function value (f arg1...argN) in one of two ways. If the function f is not an FSUBRSYM then it applies the definitions, axioms, and rules about f to the result of evaluating each argument. However if f is an FSUBRSYM then only the first argument is evaluated before the f laws are applied. SYMEVAL's symbolic evaluation of expressions is thus somewhat analogous to LISP's[McCarthy] method of evaluating expression with the exception that SYMEVAL returns a "normalform" expression equal to the input expression whereas LISP returns the meaning of that "normalform" expression. Thus whereas (CONS(QUOTE A)(QUOTE B)) evaluates to its meaning: (A.B) in LISP, it SYMbolically EVALuates to the equal expression (QUOTE(A.B)). The evaluation to an equal expression allows SYMEVAL to handle quantified variables in a graceful manner. For example, whereas the evaluation of (CONS X Y) with unbound variables X and Y gives an error in LISP, its SYMbolic EVALuation results in the equal expression (CONS X Y).

FSUBRSYM List of functions to be evaluated as FSUBRSYMs. For example, QCL has the primitive FSUBRSYMs: IF and the defined FSUBRSYMs: AND,OR,LOR,IMPLIES,IMPLY.

Specifically SYMEVAL works as follows given as input an expression E to evaluate, an association list A of bindings to apply, an association list H of hypotheses, a list FL of recursive functions being tenatively unraveled, a list of universal variables which may be solved for, and a list QE of existential variables which may be solved for.

1. If E is a VARIABLE or a SCHEMATOR then if E is bound in the association list A then the result of replaceing E by its binding in A is returned, and if it is not bound in A then E itself is returned.

2. Otherwise if E is an explicitly quoted name (eg. (QUOTE FOO)) or an automatically quoted name such as a number or string then E itself is returned.

3. Otherwise if E is an ATOMic SYMbol then the result of APPLYing axioms to that atomic symbol is returned.

4. Otherwise if E is a QUANTifier SYMbol then it is of the form: (quantifier.symbol bound.var bound.arg unbound.arg1 ... unboundargN) In this case, the expressions in the unbound argument positions are recursively SYMbolically EVALuated using the input values of A H FL QA and QE. The expression in the bound argument position is recursively SYMbolically EVALuated with A and H changed to the result of deleting any bindings containing the bound variable, with the type of the bound variable also added to H, with FL unchanged, with QA set to NIL unless the quantifier symbol is ALL in which case the bound variable is added to QA, and with QE set to NIL unless the quantifier symbol is EX in which case the bound variable is added to QE. Finally the axioms about that quantifier are SYMbollically APPLYed to the expression resulting from evaluating the arguments as described.

5. Otherwise, if E is an FSUBR SYMbol then only the first argument of E is SYMbolically EVALuated with QA and QE changed to NIL. The other arguments are merely replaced by the result of substituting any free variables or schemators in E by their bindings in A. The axioms about that FSUBR SYMbol are then SYMbolically APPLYed to the expression resulting from evaluating and substituting the arguments as described.

6. Otherwise, E is of the form (function.symbol arg1 ... argN) and every argument of E is recursively SYMbolically EVALuated without changing A,H,FL,QA , or QE. The axioms about the symbol are then SYMbolically APPLYed to the expression resulting from evaluating the arguments as described.

The SYMbolic EVALuator function calls the SYMbolic EValuator APPLY function in order to apply the axioms about a given symbol to an expression starting with that symbol. Specifically SYMEVAPPLY works as follows given as input an expression E to apply, an association list H of hypotheses, a list FL of recursive functions being tenatively unraveled, a list of universal variables which may be solved for, and a list QE of existential variables which may be solved for.

1. If E is an explicitly quoted or automatically quoted name then E is returned.

2. Otherwise, if E has the form (function.symbol arg1 ... argN) and the function symbol is not on the FUNSYM list or if E has an atomic symbol not on the ATOMSYM list, then E itself is returned.

3. Otherwise, if E has the form (function.symbol arg1 ... argN), the function symbol is the name of a function defined in INTERLISP and the arguments are all explicitly or automatically quoted objects, then the result of KWOTEing the INTERLISP evaluation of E is returned.

4. Otherwise, if the type of E is NIL then NIL is returned, and if the type of E is T then T is returned.

5. Otherwise, E is an ATOMic SYMbol, or begins with a FUNtion SYMbol. If that symbol has an explicit definition, or a recursive definition such that some subset of the arguments in E form a known measured subset for that recursive definition, then the result of recursively SYMbolically EVALuating the result of using that definition is returned.

6. Otherwise, if any axioms or rules about this symbol can be used, then the result of recursively SYMbolically EVALuating the result of using the first such usable axiom or rule is applied.

7. Otherwise, the FNORMALFORM axiom is used if it can be used.

8. Otherwise, if the symbol has a recursive definition and that function symbol is not already on the list of function symbols FL being tenatively unraveled, then it is put on that list, and that the result of SYMbolically EVALuating the result of using that definition is compared with E itself, and the more useful of the two expressions is returned.

9. Otherwise, E itself is returned.

All axioms, including axioms which axiomatize the logical symbols such and AND, OR, NOT, ALL, and EX, are essentially of the form:

(IMPLIES c(EQUAL p q))

In accordance with the Fundamental Deduction Principal such an axiom is used to replace an expression p by an expression q which is logically equal to p given that c is true in the given theory and context. For example, because the c expression of a definition is the true symbol T, a definition is used to simply replace p by q generally.

Thus given an ATOMic SYMbol E or a list E consisting of a FUNction.SYMbol followed by zero or more arguments SYMEVAPPLY tries to use axioms whose p expression either is or begins with that symbol as follows:

1. If SYMEVAPPLY is trying to use a definition then the p expression of that definition is MATCHed to the expression E. The result returned is the result of using the bindings obtained from the MATCHing to substitute for the free occurances of variables and schemators in the q expression of the definition.

2. Otherwise if SYMEVAPPLY is trying to use an axiom then the p expression of that axiom is MATCHed to the expression E. If no match is possible then this axiom is not usable on E and that fact is returned and the next axiom is tried. If a MATCH is possible then the bindings obtained from this MATCH are used to substitute for the free variables and schemators in both the c and q expressions of this axiom. If the result of repeatedly recursively SYMbolically EVALuating the new c expression is not a non NIL value then again the axiom is not useable and the next one is tried. If the result is a non-NIL value the the axiom is usuable and the new q expression is returned.

Although all axioms are essentially of the form:

(IMPLIES c(EQUAL p q))

and even though the use of schemators allows a numerous kinds of axiom schemes to be represented in QCL, there are still a number of useful schemes which cannot be represented directly within the QCL language without assending to the meta level. For this reason, SYMEVAL allows such schemes to be represented as INTERLISP functions. SYMEVAL will attempt to use any such axiom scheme represented as an INTERLISP function in the same manner as it uses any other axiom. The interface to the INTERLISP function is this: The arguments to the function are the current E, H, FL, QA, and QE. The result returned by the function is either the instantiated q expression of the hypothetical usable axiom represented by the function, or E itself if the hypothetical axiom was unusuable. In any case, the prime requirement is that the expression returned from such a function must be logically equivalent in the given theory and in the context H to the input expression E. In accordance with INTERLISP's argument conventions the function need not have formals for H, FL, QA, and QE, in which case they are ignored. Such functions may obtain their result in any manner they wish, including calling the theorem prover itself.

2.3 FUNCTION OBJECTIFICATION

Function symbols of a theory can be theoretically divided into two classes. Namely, the symbols of the theory which appear essentially as arguments to the p expression in an axiom of the form: c ==> p=q and the symbols which do not. For example the CONS function is clearly an object because of the SHELL axiom: (CAR(CONS X Y)) = X. However, the recursive function APPEND is not an object, at least until we prove some theorems about it, because there is no axiom of the form: (f(APPEND X Y) = q. It is

important to determine which non-primitive symbols are objectified at a given point in an extensible theory [Brown11] in order to avoid their evaluation in order to get the effect of a call by need evaluation[Brown4,6].

2.4 PRIMITIVE SYMBOLS OF QCL

Quantified Computational Logic consists of both primitive symbols and non-primitive symbols. The primitive symbols of QCL listed below are axiomatized by the ADDRULE command described later.

NIL "falsity" "normal end of a list"

T "truth"

(IF p l r) "if p then l else r" FSUBERSYM

(EQUAL x y) "x equals y", or if x and y are sentences "x if and only if y"

(ALL v x) "for all v x" QUANTSYM

(EX v x) "for some v x" QUANTSYM

(QUOTE x) treats x as a name

(LAMBDA v x) "the function mappinp v to x" QUANTSYM

(AP f a) "the application of function f to a"

(LAMBDAP f) "f is a good function

(BADLAMBDAP f "f is a bad function"

(QCL x) "x is QCL-true" MODALSYM

2.5 TYPES

Every symbol in QCL has one or more types associated with its function value. In addition the bound variable of each quantifier is also of one or more types. The initial types are: NIL, T, LAMBDAP, BADLAMBDAP, OTHERN. The NIL type consists of the atom NIL. The T type consists of the atom T. The LAMBDAP type consists of those functions which can be arguments to an application. The BADLAMBDAP type are the other functions. The OTHERN type consists of those numbers which are not recursively constructed by the shell principle. An expression is of type NIL or T iff it is EQUAL to that value: (EQUAL X NIL) or (EQUAL X T) For any other type, an expression is recognized to be that type by a recognizer symbol in QCL whose name is that type. For example (LAMBDAP X) iff X is of type LAMBDAP, and (BADLAMBDAP X) iff X is of type BADLAMBDAP. Also types for lists, positive integers, negative integers, decimal numbers, strings, and literal atoms are intially created types produced by the SHELL principle described below. These types are called: LISTP, PNUMBERP, NNUMBERP, DECIMALP, STRINGP, LITATOMP. NIL and T are not LITATOMPs because they have their own type.

UNIVERSE The list of all currently defined types in the system. {LITATOMP STRINGP DECIMALP NNUMBERP PNUMBERP LISTP NIL T LAMBDAP BADLAMBDAP OTHERN OTHER}

(SETQ XXX(PRINTTYPES)) (PP XXX) Pretty prints the type of the function value of every symbol in the system.

The type of the function value of a primitive symbol is already know. The type of the bound variable of a primitive quantifier is already known. The type of the function value of a newly defined symbol is automatically created when that symbol is defined. However, the type of the bound variable of a newly defined quantifier must be explicitly declared by the following command.

(VASSUME quantifier.symbol types)

For example, a newly defined universal quantifier ALL.LISTS.AND.STRINGS ranging over only lists and strings would need the command:

(VASSUME 'ALL.LISTS.AND.STRINGS '(LISTP STRINGP)).

3. THE SYMMETRIC LOGIC DEDUCTION SYSTEM

3.1 INTRODUCTION

The SYMEVAL theorem prover now runs with a new logical system called SYMMETRIC LOGIC which treats universal and existential quantifiers in an analogous manner. For example as suggested by [Wang2] several years ago, SYMMETRIC LOGIC rewrites both of the following sentences to **(FOO A)**, after evaluating their subexpressions, when trying to prove them:

(ALL X (IMPLIES (EQUAL X A) (FOO X)))

(EX X (AND (EQUAL X A) (FOO X)))

Thus the essence of the SYMMETRIC LOGIC technique is to push quantifiers to the lowest scope possible in hopes of finding a way to eliminate them. Thus unlike the sequent calculus[Szabo,Brown1,3,4,6,12,25] and other logic systems[Bledsoe1,2,Bibel2] based on the Prawitz-Robinson Unification algorithm [Robinson], which essentially loses the scope of the existential quantifier during the skolemization process, SYMMETRIC LOGIC handles equalities very well indeed. The power of a logic which handles equalities like this is very convincing, in an application domain dealing essentially with equations such as real algebra, logic programming, and language analysis. SYMMETRIC LOGIC is the synthesis of several earlier logic systems including the initial symmetric logic used by the real algebra rule package[Brown24], and the bind logic used by the logic programming and natural language rule packages[Brown26]. Current research is aimed mainly at synthesizing the SYMETRIC LOGIC method of handling quantifiers with the method used by our sequent calculus system, into one general system. The purpose of synthesizing these different logical systems is to eventually develop one simple, systematic, yet general logic system capable of performing well in many application domains.

3.2 PROPOSITIONAL LOGIC

The syntax of SYMMETRIC LOGIC includes three propositional symbols:

NIL	meaning: false
T	meaning: true
(IF p l r)	meaning: if p then l else r FSUBR

IF treats all non-Boolean objects(eg. LISTPs NUMPERPs and OTHERS) as if they were T. An expression x is Boolean iff x is T or NIL when evaluated in some world. Thus any non-NIL object is assumed to be a true value. For example: (IF 44 1 2) is 1. The NIL and T atom symbols aer SUBRs, whereas the IF symbol is an FSUBR. Thus only the first argument of IF is SYMbolically EVALuated before the IF axioms are applied.

The SYMIF rule is used on an expression (IF p l r) in the following manner:

1. First the type of p determined. If it is NIL then r is returned and if it does not include NIL then l is returned. These rules can be written schematically as: 1. if the type of p does not contain NIL then (IF p l r) = l 2. if p=NIL then (IF p l r) = r

2. Otherwise, if l is of type NIL and r is not, r is of type NIL and l is not, the l-effects of p contribute significantly to binding the free variables in l ,or the r-effects of p contribute significantly to binding the free variables of r

then the IFNORMALFORM scheme is applied to (IF p l r). The effects of p consist of the equality statements in p of the form (EQUAL v t) or (EQUAL t v), hereafter called bindings: 3. if certain conditions hold then IFNORMALFORM

If neither of the above cases hold, then p is assumed to be true with the H list set accordingly, p is solved for a variable if possible and the A list is set accordingly(ie that solution is added as an additional binding and is also used to modify anyother bindings), and l is SYMbolically EVALuated. Next, p is assumed to be false and the H list is set accordingly and r is SYMbolically Evaluated.

SYMIF then tries to use the following rules in the given order:

4. if l=r then (IF p l r) = r
5. if l differs from r only in bindings contained in the context list A then (IF p l r) = r
6. if p => l=r then (IF p l r) = r
7. if p=l, r=NIL then (IF p l r) = p
8. if p is Bool, l=T, r=NIL then (IF x l r) = x
9. IFNORMALFORM

   This rule is not the scheme:

   (IF (IF p a b) l r) = (IF p (IF a l r) (IF b l r))

   which is far too inefficient for effective deduction. Instead the following more general scheme is used:

   (IF (IF p ...a...) L R) = (IF p ...(IF a l r)...)

   for every largest subexpression not containing an IF symbol. Furthermore, If a=NIL then (IF a l r) becomes r, and if the type of a does not include r then (IF a l r) becomes l.
10. Equality Substitution

    This rule applies the schema:

    (IF(EQUAL x y)(p x y)) = (IF(EQUAL x y)(p x1 y1))

    after solving the equation (equal x y) for an interesting subexpression u.

IF applies the IFNORMALFORM axiom before evaluating l and r whenever:

(1) l and r occur no more than once in the resulting expression
    (ie. if the truth values are known we get:
        (IF (IF p NIL NIL) l r) = by above axioms: (IF NIL l r) = r
        (IF (IF p NIL Tor) l r) = (IF p r l)
        (IF (IF p Tor NIL) l r) = (IF p l r)
        (IF (IF p Tor Tor) l r) = (IF p l l) = l
    Tor means T or any non boolean exp.)
(2) the bindings in x contribute significantly to the evaluation of

l or they contribute significantly to the evaluation of r.

(3) the type restrictions in x contribute significantly to l or to r

(not yet implemented).

The SYMMETRIC propositional logic also includes the functional normal form rule which eliminates embedded occurences the IF symbol. The FNORMALFORM axiom embodies:

```
(f x1...xn(IF p l r)y1...ym)  =  (IF p(f x1...xn l y1...ym)
                                    (f x1...xn r y1...ym))
```

3.3 EQUALITY LOGIC

The syntax of SYMMETRIC LOGIC includes four an equality symbol:

(EQUAL l r) meaning: l equals r

The EQUAL symbol is a SUBR. Thus all arguments to EQUAL are SYMbolically EVALuated before the EQUAL axioms are applied. Generally speaking, equality has four fundamental properties:

```
1.  Equality is reflexive:      x=x
2.  Equality is symmetric:      x=y implies y=z
3.  Equality is transitive:     x=y implies (y=z implies x=z)
4.  Equality is extentional:    x=y implies (Px implies Py)
                                x=y implies (y=z) = (x=z)
                                x=y implies fx = fy
```

The EQUAL axioms are given below.

The first two equality laws deal with the refexitivity of equality and the inequality of distinct data objects.

1. (EQUAL X X) = T
2. if l and r are different data objects then (EQUAL l r) = NIL

The next six axioms reduce EQUAL to IF whenever possible.

3. (EQUAL NIL r) = (IF r NIL T)
4. (EQUAL l NIL) = (IF l NIL T)
5. if l is boolean in a world then (EQUAL l T) = l
6. if r is boolean in a world then (EQUAL T r) = r
7. (EQUAL l(EQUAL x y)) = (IF(EQUAL x y)(EQUAL l T)(EQUAL l NIL))
8. (EQUAL (EQUAL x y)r) = (IF(EQUAL x y)(EQUAL r T)(EQUAL r NIL))

This law reduces the equality of functions (ie. Frege's Werthverlauf) to the equality of their function values on equal arguments.

9. (EQUAL (LAMBDA x(p x)) (LAMBDA y(p y)))
   = (ALL V(EQUAL (APPLY(LAMBDA x(p x))V) (APPLY(LAMBDA y(p y))V)))

3.4 QUANTIFICATIONAL LOGIC

SYMMETRIC LOGIC also has two quantifiers:

(ALL v p) meaning: for all v, p

(EX v p) meaning: for some v, p

The ALL axioms are:

```
0.ALLt       type p is not NIL ==> (ALL v p) = T
1.ALL~v      (ALL v p) = p
2.ALLvv      (ALL v v) = NIL
3.ALL=       (ALL v((not(EQUAL v t))...or...(p v))) = (p t)
4.ALLrecog   (ALL v(recog v)) = NIL
5.QUANTIF    (ALL v(IF p (l x)(r x))) = (IF p (ALL v(l v)) (ALL v(r v)))
6.ALLand     (ALL v(...or...(and (x v) (y v))...))
               = (and(ALL v(...or...(x v)...))(ALL v(...or...(y v)...)) )
7.ALLor
8.ALLnot    (ALL v(IF p NIL T)) = (IF(EX v p) NIL T)
9.ALLident  (ALL v(IF p T NIL)) = (IF(ALL v p) T NIL)
10.ELIM
```

The EX axioms are:

```
0.EXt        type p is not NIL ==> (EX v p) = T
1.EX~v       (EX v p) = p
2.EXvv       (EX v v) = T
3.EX=        (EX v((EQUAL v t)...and...(p v))) = (p t)
4.EXrecog    (EX v(recog v)) = T
5.QUANTIF    (EX v(IF p (l x)(r x))) = (IF p (EX v(l v)) (EX v(r v)))
6.EXor       (EX v(...and...(or (x v) (y v))...))
               = (or(EX v(...and...(x v)...))(EX v(...and...(y v)...)) )
7.EXand
8.EXnot     (EX v(IF p NIL T)) = (IF(ALL v p) NIL T)
9.EXident   (EX v(IF p T NIL)) = (IF(EX v p) T NIL)
10.ELIM
```

EX-OR does:

```
1.              (IF a (IF p l r) NIL) = (IF p (IF a l NIL) (IF a r NIL))
2. if l is Boolean then (IF p l r) = (IF (IF p l NIL) T (IF p NIL r))
3.              (EX v(IF(a v)T(b v))) = (IF (EX v(a x)) (EX v(b v)) T)
```

ALL-AND does:

```
1.              (IF a (IF p l r) T) = (IF p (IF a l T) (IF a r T))
2. if l is Boolean then (IF p l r) = (IF (IF p l T) (IF p T r) NIL)
```

```
3.              (ALL v(IF(a v)(b v)NIL)) = (IF(ALL v(a x))(ALL v(b v))NIL)
```

The Elimination axioms have the forms:

```
(EQUAL(h X Z1...Zn)
      (EX I1...Im (AND(EQUAL(const Z1...Zn I1...Im)X)
                      (c Z1...Zn I1...Im))) )
```

They have the effect of the rewrite schemmas:

```
(EQUAL(ALL X(s X))
      (AND(ALL Z1...(ALL Zn(ALL I1...(ALL In
                                (IMPLIES(c Z1...Zn I1...Im)
                                        (s(const Z1...Zn I1...Im)) ) ))))
          (ALL Z1...(ALL Zn(ALL X(IMPLIES(NOT(h X Z1...Zn))
                                        (s X)))))  ) )

(EQUAL(EX X(s X))
      (OR(EX Z1...(EX Zn(EX I1...(EX Im
                              (AND(c Z1...Zn I1...Im)
                                  (s(const Z1...Zn I1...Im)) ) ))))
          (EX Z1...(EX Zn(EX X(AND(NOT(h Z1...Zn))
                                  (s X))))) ))
```

3.5 ABSTRACTION LOGIC

The SYMMETRIC LOGIC has a primitive symbol: LAMBDA for functional abstraction,(ie. for forming the werthverlauf of a function value) and a primitive symbol: AP for function application: AP.

(LAMBDA v(p v))
(AP g a)

Five Axioms and theorems similar to the set theory abstraction axioms used in [Brown4,6] are assumed for lambda conversion:

```
T1.(BADLAMBDAP X)    ==>      (AP S X) = NIL
T2.(NOT(BADLAMBDAP X))  ==>   (AP(LAMBDA X(p X))X) = (p X)
A3.                           (AP(LAMBDA X(p X))X) = (IF(BADLAMBDAP X)NIL(p X))
T4.(AND(NOT(BADLAMBDAP X))
      (NOT(BADLAMBDAP S))
      (NOT(LAMBDAP S))) ==> (AP S X) = (EQUAL S X)
A5.(AND(NOT(BADLAMBDAP S))
      (NOT(LAMBDAP S)))  ==> (AP S X) = (IF(BADLAMBDAP X)NIL(EQUAL S X))
```

One axiom for determining the equality of functions is assumed:

```
A6.(EQUAL(LAMBDA v(p v))(LAMBDA v(q v))) =
              (ALL X(EQUAL(AP(LAMBDA v(p v))X) (AP(LAMBDA v(p v))X) ))
```

One theorem (similar to T1 above) is assumed for use by the type system:

```
T7.(AP F X) ==> (NOT(BADLAMBDAP X))
```

Set theoretic abstaction: SET and ELEmenthood are defined as follows:

```
(SET v(p v)) = (LAMBDA v(p v))
(ELE x s) = (APPLY s x)
```

Thus every function is a set and every set is a function. Functional equality is of course EQUAL, whereas set equality is:

```
(SETEQUAL x y) = (ALL V(IFF(ELE V x)(ELE V y)))
```

The set theory axioms are neutral with respect to a classical or Lesniewskian set theory. For example, at least the following axiom should be added for Lesniewskian set theory: (EQUAL (SET v(EQUAL v X)) X)

3.6 MODAL LOGIC

The SYMMETRIC LOGIC also includes two modal operators: (QCL p) meaning: p is QCL-logically true (POS p) meaning: p is QCL-logically possible

The QCL axioms are:

1. if x is NIL-or-CONTINGENT then (QCL x) = NIL
2. if x is not CONTINGENT then (QCL x) = x (ie x=T,NIL,or non-bool)

The POS axioms are:

1. if x=NIL then (POS x) = NIL
2. if x is not NIL then (POS x) = T

Other facilities for Modal Logic are described in Brown[17,18,21]

4. AXIOMATIZING THE THEORY

The QCL language may be augmented with new data structures, new function definitions, and other axioms. The function bodies of definitions may include quantifiers.

4.1 DATASTRUCTURES: THE SHELL PRINCIPLE

The shell principle allows QCL to create functions for dealing with new data stuctures after the user specifies a minimum of information. The Shell Principal produces axioms about datastructures which obey the Fundamental Deduction Principle discussed in section 2. Although many of the axioms are similar to those described in [Boyer1], some of the axioms are, in order to preserve the property of replacing expressions by equal expressions, necessarily quite different. Shells are added via:.

(SHELLCREATE const btm recog ac-list type-list default-list)

with

const	The name of the new function which constucts objects of the new type.
recog	The name of the new function which recognizes objects of the new type.
btm	Bottom object. Use T if there is no bottom object.
ac-list	List of the functions which are accessors of the data structure.
type-list	List of the type restrictions on the shell. Each restriction is an arbitrary formula in QCL consisting of symbols defined at that time.
default-list	List of the default values for the shell.

Shells may be tested for validity via:

(SHELLTEST const recog arity ac-list type-list default-list)

The function SHELL, combines the functions of SHELLTEST and SHELLCREATE. (To save computation time, it is recommended that SHELLCREATE be used instead, with the user calling SHELLTEST only when the shell is first introduced.)

Shells for the basic INTERLISP data structures of lists, positive integers negative integers, decimal numbers,strings, and atoms are initially automatically created when QCL is entered by the commands:

```
(SHELLCREATE 'CONS 'T 'LISTP '(CAR CDR) '(T T) '(NIL NIL))
(SHELLCREATE 'PADD1 '(ZERO) 'PNUMBERP '(PSUB1)
        '((PNUMBERP X1)) '((ZERO)))
(SHELLCREATE 'NMINUS 'T 'NNUMBERP '(POSPART)
        '((IF(EQUAL X1 0)NIL(PNUMBERP X1))) '(0))
(SHELLCREATE 'TENDIV 'T 'DECIMALP '(SIG MANTISSA)
        '((AND(NOT(EQUAL X1 0))(OR(PNUMBERP X1)(NNUMBERP X1)))
```

```
        '(AND(PNUMBERP X2)(NOT(EQUAL X2 0))))
        '(1 1))
(SHELLCREATE 'CONCATPNUM '(NULLSTRING) 'STRINGP
        '(FIRST REST) '((PNUMBERP X1)(STRINGP X2))
        '(0 (NULLSTRING)))
(SHELLCREATE 'MAKEATOM 'T 'LITATOMP '(MAKESTRING)
        '((AND(STRINGP X1)(NOT(MEMB X1 '("NIL","T",""))))) '("0"))
```

The shell axioms are qcl-true.

4.2 THEORY OF QUOTATION

Expressions concerning these six initial shells can be abreviated by a theory of quotation based on normal INTERLISP conventions. The following are examples of the special shorthand equivalents for lists, strings and numbers which can save the user time and effort.

```
(QUOTE(THIS IS A LIST))=(CONS(QUOTE THIS)
                             (CONS(QUOTE IS)
                                  (CONS(QUOTE A)
                                       (CONS(QUOTE LIST)NIL))))

4=(ADD1(ADD1(ADD1(ADD1(ZERO)))))


"ABC"=(CONCATPNUM(QUOTE 65)
                 (CONCATPNUM(QUOTE 66)
                            (CONCATPNUM(QUOTE 67)
                                       (QUOTE ""))))

(QUOTE ABC)=(MAKEATOM "ABC")
```

Note that numbers, strings, NIL(the normal end of a list), and T are automatically quoted. Sometimes C-LISP conventions will also work.

4.3 DEFINITIONS

Definitions are of the form:

(EQUAL (function-name-being-defined arg1...argN)definition-body)

The body of a definition may invlove both recursive calls to themselves and quantifiers such as ALL, EX, LAMBDA. The symbol being defined may itself be a quantifier. Schemators may also appear in both the argument list and body of the function being defined. For example a recursive definition for the summation function: SIGMA is:

```
(EQUAL(SIGMA K(SCH1 K)M N)
      (IF(PNUMBERP M)
         (IF(PNUMBERP N)
            (IF(EQUAL M N)
               (SCH1 N)
```

```
            (IF(LESSP M N)
               (PLUS(SIGMA K(SCH1 K)M(PSUB1 N))(SCH1 N))
               0))
         0)0) )
```

Recursive definitions for CLISP iteration operators can also be defined using schemators. Such definitions for COLLECT and JOIN are given below:

```
(EQUAL(COLLECT K(SCH1 K)M N)
      (IF(PNUMBERP M)
         (IF(PNUMBERP N)
            (IF(EQUAL M N)
               (LIST1(SCH1 N))
               (IF(PLESSP M N)
                  (APPEND(COLLECT K(SCH1 K)M(PSUB1 N))(LIST1(SCH1 N)))
                  NIL))
           NIL)NIL) )
(EQUAL(JOIN K(SCH1 K)M N)
      (IF(PNUMBERP M)
         (IF(PNUMBERP N)
            (IF(EQUAL M N)
               (SCH1 N)
               (IF(PLESSP M N)
                  (APPEND(JOIN K(SCH1 K)M(PSUB1 N))(SCH1 N))
                  NIL))
           NIL)NIL) )
```

Note that SCH1 is a schemator.

The commands which create definitions are:

(DEF definition) Adds a definition to those already in existence after ensuring that it is either an explicit or recursive definition. Induction templates are created and declared if the definition is recursive, and the type of the function value of the defined symbol is declared.

(DEFL list-of-definitions) Adds a list of definitions to those already existing using DEF.

(DEFLQ definition1...definitionN) FSUBR (FSUBR version of DEFL)

SYMEVALDEF Switch, when set to T causes the DEFinition commands to SYMbolically EVALuate the bodies of the definitions before the definitions are actually asserted. The Default is T.

The following declaration commands can be used to assert definitions that the system is currently unable to define automatically.

(DCL definition) Declares a definiton, and assumes that the definition is recursive on its first argument position. In fact it assumes that the COUNT measure is decreasing on the measured unit set consisting of the formal variable in that position. It does not first prove this fact and thus does not guard against infinite recursion. This command is useful for defining things the theorem prover can not yet handle. It is especially useful for asserting definitions which are recursive but which are to difficult for this theorem prover to prove. Psuedo functions(ie functions which are executed for their side effects) should be declared so as to force their type calculation to occur at run time instead of at definition time.

(DCLL list.of.definitions) Declares a list of definitions using DCL.

(DCLLQ definition1...definitionN) FSUBR (FSUBR version of DCLL)

SYMEVALDCL Switch, when set to T causes the DeCLaration commands to SYMbolically EVALuate the bodies of the definitions before the definitions are actually asserted. The Default is T.

Definition schemes are INTERLISP functions which when given an expression beginning with the symbol being defined returns another expression with that symbol replaced by its definition. These schemes are useful for defining new quantifiers including recursive quantifiers such as SIGMA, and infinite numbers of definitions. Definition schemes are implemented by use of the following command:

(DCLSCHEME symbol.being.defined interlisp.function.name)

The system does not automatically produce induction templates and type information for declared symbols. Such information can be given to the system by the following commands.

(DCLTEMPLATE symbol '(measure.expression machine axioms-justifying the template)). For example the template for SIGMA is: '((COUNT M) (((LESSP M N) ((M(PSUB1 N))))) PSUB1.LESSP)

(ADDTYPE symbol (list.of.types . list.of.argument.positions)). For example, the type of SIGMA is: '((PNUMBERP NNUMBERP DECIMALP OTHERN). NIL)

The following definitions are initially made when QCL is entered:

```
(DEFLQ
  (EQUAL (LET V(SCH1 V)X) (SCH1 X))
  (EQUAL (NOT P) (IF P NIL T))
  (EQUAL (AND P Q) (IF P Q NIL))
  (EQUAL (OR P Q) (IF P P Q))
  (EQUAL (LOR P Q) (IF P T Q))
  (EQUAL (IMPLIES P Q) (IF P Q T))
  (EQUAL (IMPLY P Q) (IF P Q T))
  (EQUAL (IFF P Q) (IF P(IF Q T NIL)(IF Q NIL T)))
  (EQUAL (OBJECT X) (IF(BADLAMBDAP X)NIL X))
  (EQUAL (LITATOM X) (LOR(LITATOMP X)(LOR(EQUAL X NIL)(EQUAL X T))))
  (EQUAL (INUMBERP X) (OR(PNUMBERP X)(NNUMBERP X)))
  (EQUAL (NUMBERP X) (OR(OTHERN X)(OR(DECIMALP X)(INUMBERP X))))
  (EQUAL (LISP.DATA.STRUCTURE X)
         (LOR(LISTP X)(LOR(STRINGP X)(LOR(NUMBERP X)(LITATOM X)))))
```

)

The (LET V(SCH1 V)X) function is used to assign a variable V to the result of SYMbolically EVALuating the value X only once and then substituting it into the (SCH1 v) expression as many times as v occurs there. It is somewhat like (APPLY(LAMBDA V(SCH1 V))X) but without any commitment to X not being a BADLAMBDAP. This construct is also analogous to the: "Let V=X and_return(SCH1 v))" construct found is some programming languages.

Note that AND and OR are the normal INTERLISP AND and OR functions where AND returns the last true value or NIL, and OR returns the first true value or NIL. LOR is a more logical OR function returning T or the third argument. AND, OR, and LOR have the following properties:

```
These laws hold:
   (AND(AND P Q)R)=(AND P(AND Q R))
   (OR (OR  P Q)R)=(OR  P(OR  Q R))
   (LOR(LOR P Q)R)=(LOR P(LOR Q R))
   (IF P X Y)=(OR(AND P X)(AND(NOT P)Y))

These laws do not hold:
   (IF P X Y)=(AND(OR P Y)(OR(NOT P)X))
   (IF P X Y)=(LOR(AND P X)(AND(NOT P)Y))
   (IF P X Y)=(AND(LOR(NOT P) X)(LOR P Y))
```

4.4 REDUCTION AXIOMS

Reduction axioms are of one of the following forms

p replaces p by T

(EQUAL p q) replaces p by q

(IMPLIES c(EQUAL p q)) replaces p by q, whenever c holds

(IF c (EQUAL p q) T) replaces p by q, whenever c holds

Unlike definitions, reduction axioms should not involve infinite looping. In particular, the q expression should be simpler than p. For example q should not contain an alphbetic varient of p.

(ADDAXIOM axiom name-of-axiom)

Adds axiom as a new rewrite rule to the system.

(ADDAXIOMS list-of-axioms) FSUBR

Calls ADDAXIOM on each axiom in the list. It uses the first symbol in the "p" sub-expression as the default name.

(ADDELIM elimination-axiom)

A special kind of reduction axiom schema called an Elimination axiom may be added by this command. Elimination axioms have the form:

```
(EQUAL(h X Z1...Zn)
     (EX I1...Im (AND(EQUAL(const Z1...Zn I1...Im)X)(c Z1...Zn I1...Im))) )
```

For example:

```
(EQUAL(h X Z1...ZN) (EX I1...Im (EQUAL(const Z1...Zn I1...Im)X)) )
```

They have the effect of the rewrite schemmas attached to the ALL and EX quantifiers:

```
(EQUAL(ALL X(s X))
      (AND(ALL Z1...(ALL Zn(ALL I1...(ALL In
                                    (IMPLIES(c Z1...Zn I1...Im)
                                          (s(const Z1...Zn I1...Im)) ) ))))
          (ALL Z1...(ALL Zn(ALL X(IMPLIES(NOT(h X Z1...Zn))(s X)))))  ) )
(EQUAL(EX X(s X))
      (OR(EX Z1...(EX Zn(EX I1...(EX Im(AND(c Z1...Zn I1...Im)
                                          (s(const Z1...Zn I1...Im)) ) ))))
          (EX Z1...(EX Zn(EX X(AND(NOT(h Z1...Zn))(s X))))) ))
```

The ELAXL list determines the use of such lemmas. It has the form:

```
( (elaxname ({selector          } ...
            {(selector position)} ...  )) ...)
```

(ADDRULE symbol lisp-function-name) The lisp-function-name is a lisp function which has the effect of one or more rewrite axioms associated with the formal symbol. This command is useful for axiomatizing quantifiers and other schematic symbols. It is also useful for asserting complex axiom schemes.

4.5 EQUATION SOLVING

SYMEVAL will try to solve equations at the appropriate time, and use those solutions at the appropriate time provided the user defines an INTERLISP function called RSOLVE which will solve equations for 0 or more solutions. The system will attempt to solve equations for variables whenever the quantifier binding that variable is being applied, or whenever that variable is free in the theorem being proven.(ie. whenever the variable is an unknown.) The system will also attempt to solve for an appropriate variable of lowest scope in an equation occurring as the first argument of an IF function value or for a latest defined function value in an equation occurrings as the first argument of an IF function value provided that the function value also occur in the second argument position of the IF statement.

4.6 INDUCTION AXIOMS

The Induction scheme was adopted from [Boyer1] and has since been extented to handling recursive definitions containing quantifiers, bound variables and schemators. Induction axioms have the form:

```
'(IMPLIES(test X1...Xn)
        (LESSP(measure(selector X1...Xn))(measure X1...Xn)) )
```

Many induction axioms are automatically created when the SHELL command is used to create a datastructure. For example:

```
'(IMPLIES(LISTP X)
        (LESSP(COUNT(CDR X))(COUNT X)) )
```

is created when the list datastructure is asserted. Additional induction lemmas may be asserted with the following command:

(MKINAXIOM induction.axiom.name induction.axiom)

4.7 LINK TO INTERLISP INTERPRETER

The theorem prover automatically links to the INTERLISP interpreter whenever all the evaluated arguments of a function are explicit values or quoted objects provided that the function name appear on the FUNSYM list. This is done because INTERLISP is much faster than SYMEVAL. The SHELL.DEF, and DCL commands automatically place the function names they deal with onto the FUNSYM list. Thus the user should be careful to define any logical functions having the same name as some INTERLISP function, to exactly correspond in effect to that INTERLISP function. However, it is not required that logical definitions exist for a link to interlisp to be made. For example, if the name of every INTERLISP subr function were CONSed onto FUNSYM then QCL would be a superset of INTERLISP subrs. INTERLISP psuedo-functions (ie. functions which are being executed for there side effects should be DeCLared (DCL) not DEFined, because DEF creates type info at definition time, which can cause the defined not to be executed at run time. For examply if the type of a DEFined print function were NIL it would never be executed at run time because the system would know that the function valued equaled NIL.

5. USING THE SYSTEM

Anyone who uses this system for programing as opposed to more general deductive tasks should bear in mind that this is a general theorem proving system which has not been engineered especially for computation. Thus, the system will appear to be somewhat slow in comparison to systems engineered specifically for computational tasks. This slowness is in no way a reflection on how well an interpreter for this language could be engineered for computational tasks.

5.1 INITIALIZING THE SYSTEM

getting started on the Research DEC20.
.LISP
*LOAD(AUX:<CS.BROWN>ATP.COM)

restarting the system:
*(SYSINIT)

The system version may be obtained by typing
ATP.VERSION
This paper corresponds to version 3.

5.2 EXECUTING PROGRAMS AND PROVING THEOREMS

(PV theorem) FSUBR (but if theorem is a variable it is evaluated like a SUBR) Attempts to prove a theorem using PV and collects statistics on the process. More useful than PROVE because theorem could be a variable containing the theorem.

(PROVE theorem) Attempts to prove a theorem using recursive SYMbolic EVALuation: SYMEVAL0, and Noetherian induction.

(SYMEVAL0 theorem) Attempts to prove a theorem using only recursive SYMbolic EVALuation.

5.3 DEBUGGING PROGRAMS

As **SYMEVAL0** recursively evaluates expressions tracing information is produced whenever a definition, axiom, or rule is applied. This information consists of three parts: 1. An input expression to which the axiom is being applied, called I. 2. The midterm expression produced by the application of the axiom, called M. 3. The output expression obtained by recursively evaluating the M expression, called O. Thus, a trace will generally be of the form:

```
I1:exp
M1:exp
   I2:exp
   M2:exp
      ...
   O2:exp
   I2:exp
   M2:exp
```

```
...
    02:exp
  01:exp
```

where the numbers immediately following I,M,or O are the level at which that application takes place. At a given level number i, Oi and Mi are always associated with the preceeding Ii.

DEBUGGING PRINCIPLE The basic method of debugging is this: if some Ii expression is not logically equal to the coresponding Oi in the given theory and in the specific context, then either Ii does not equal Mi or Mi does not equal Oi. If Ii does not equal Mi then the definition, axiom or rule used in that application is incorrect. If Mi does not equal Oi then one must recursively examine the i+1 level (ie Ii+1, Mi+1, Oi+1) to find the error.

TRACELIST List of currently defined symbols to be traced. Each time one of these symbols is used, tracing information will be printed in the trace file. (Note: if there is no trace file, the tracing information defaults to the screen.) When a trace of all function symbols is desired, TRACELIST can be set to FUNSYM.

OPENTR(trace-file) FSUBR Sets up a file for tracing information.

CLOSETR() Closes the current trace file.

TRACEM Switch, when set to T, causes the mid term (the term after an expression has been replaced by its definition, but before it has been evaluated) to be printed as well as the input and output expressions on the trace. The default is T.

TRACEN Switch, when set to T, causes the name of each axiom or rule to be printed when it is used. The default is T.

TRACEH Switch, when set to T, causes the evaluation of any hypothesis to an axiom, which the system is trying to use, to be traced. The default is NIL.

TRACESYS Switch, when set to T, causes the a-list and type set a-list to be printed along with each tracing line. The default is NIL.

5.4 EXAMPLE OF USING THE SYSTEM

Here is a simple example of using SYMEVAL-QCL as a programming language for natural language processing. We define functions for recognizing an adjective list and translating it into logic. For our purposes, an adjective list is either NIL, or an adjective followed by an adjective list.

```
@lisp

INTERLISP-10  27-NOV-79 ...


Hello, Handsome.
2_LOAD(ATP.COM)
compiled on  7-Jun-82 10:13:24
(ADDRULE redefined)

Transoring of<CS.BROWN>NATP..17
 done on 13-Feb-81 12:50:41
```

```
<CS.BROWN>ATP.COM.1
3_LOAD(ATP.DEFS)

<CS.BROWN>ATP.DEFS.1
```

After loading the theorem prover and the ATP.DEFS environment, we define SASSOCP for looking up words in the lexicon.

```
4_(DEFLQ
   (EQUAL (SASSOCP X L V)
          (EX U
              (IF (EQUAL U (SASSOC X L))
                  (IF U (EQUAL V U) NIL)
                  NIL))))
NIL
```

Next, we define the lexicon.

```
5_(DEFLQ
   (EQUAL (ADJW)
          (QUOTE((RED RED)
                 (BIG BIG)
                 (SMALL SMALL))) ))
NIL
```

Now, we define ADJ for processing a single adjective.

```
6_(DEFLQ
   (EQUAL (ADJ X Y A Z)
          (EX U
              (IF (SASSOCP (CAR X)(ADJW)U)
                  (IF(EQUAL Y (CDR X))
                     (EQUAL Z (LIST2 (CAR(CDR U)) A))
                     NIL)
                  NIL))))
NIL
```

Finally, we define ADJL to process an adjective list.

```
7_(DCLLQ
    (EQUAL(ADJL X0 X2 A Z)
          (IF(EX X1
               (EX Z1
                   (IF(ADJ X0 X1 A Z1)
                      (EX Z2
                          (IF(ADJL X1 X2 A Z2)
                             (EQUAL Z(LIST3 (QUOTE AND) Z1 Z2))
                             NIL))
                      NIL)))
             T
             (IF(EQUAL Z T) (EQUAL X2 X0) NIL))))
NIL
```

Having defined ADJL, we can now test it.

```
8_(PV (ADJL (QUOTE ()) NIL (QUOTE A) Z))
(EQUAL Z T)
```

```
9_(PV (ADJL (QUOTE (BIG RED)) NIL (QUOTE A) Z))
(EQUAL Z (QUOTE (AND (BIG A) (AND (RED A) T))))

10_(PV (ADJL (QUOTE (BIG BOY))(QUOTE (BOY)) (QUOTE A) Z))
(EQUAL Z (QUOTE (AND (BIG A) T)))
```

6. RULE PACKAGES

These libraries of rules may be acceses from account AUX:<CS.BROWN>. For example: AUX:<CS.BROWN>ATP.DEFS accesses the rule package ATP.DEFS described below.

ATP.COM This is the basic theorem prover code. It automatically sets up an initial environment for QCL programming.

6.1 REAL ALGEBRA

ATP.REAL This file contains a theory for simplifying expressions of real algebra. The primitive symbols are:

(PLUS n m)
(MINUS n)
(TIMES n m)
(EXP n i)
(LESSP n m)

This file contains rules axiomatizing each of these primitive symbols. A description of these rules?pis` given in [Brown24]. These rules include rewrite rules dealing with algebraic simplification and basis argument rule for eliminating ALL before an EQUAL expression. It also includes the equation solving system: RSOLVE linked to via SYMEVALS equation solving link. The defined symbols of the theory are:

(ADD1 n)
(SUB1 n)
(DIFFERENCE n m)
(QUOTIENT n m)
(MINUSP n)
(GREATERP n m)
(LEQ n m)
(GEQ n m)
(MIN n m)
(MAX n m)
(ABS n)
(SQRT n)
(FAC n) **factorial**
(CO x y) **combinations**
(SIGMA :k(f k)m n) **sigma**

6.2 MISCELLANEOUS DEFINITIONS

ATP.DEFS Defines some basic recursive functions for QCL programming along the lines described in [Brown20,Kowalski].

1. Defines more functions for manipulating lists:

   CADR CDDR CAAR CDAR CADDR CDDDR CADDDR LIST1 LIST2

   LIST3 NLISTP PLISTP APPEND SASSOC REVERSE MEMBER OCCUR

2. Defines functions for positive integers:

   PFIX PZEROP PPLUS PTIMES PDIFFERENCE PHALF PLESSP GCD

   INUMBERP

ATP.SORT Contains various sorting functions including a merge sort and two quicksorts.

7. SCHWIND'S THEORY OF LANGUAGE ANALYSIS

ATP.NATL A fragment of Schwind's theory of language analysis[Schwind,Brown13,14,15]. This fragment of Schwind's theory specifies how a small subset of English may be translated into the QCL Logic. The most important function in ATP.NATL is TEXT. TEXT takes a piece of text (one or more sentences) and returns its representation in logic. There are a number of lower level fuctions for translating nouns, verbs, prepostional phrases, relative clauses and so on. Consult ATP.NATL itself for these functions.

TEXT is invoked in the following manner.

(TEXT i-eng o-eng i-nlist o-nlist i-vlist o-vlist o-logic)

with

i-eng	The entire English input.
o-eng	The remainder of the English input after the first sentence or piece of text has been processed.
i-nlist	Input list of previously seen nouns for noun/pronoun substitution.
o-nlist	Output list of previously seen nouns including any found during processing.
i-vlist	Input list of object language variables.
o-vlist	Output list of unused object language variables.
o-logic	Output translation of the sentence or text into logic.

NATL A list of all the gramatical functions in NATL. Usually used to trace the parsing of a sentence, without tracing every FUNSYM.

We believe that other logically based natural language theories such as [Simmons1,2] would be naturally expressed in QCL and executed by SYMEVAL.

7.1 EXAMPLE DEFINITIONS

A few example definitions from Schwinds Natural Language Theory are given below. The first definition is a "backtracking" definition which involves a search through these 4 alternative non-exclusive cases. This definition states that a piece of text is a Noun Phrase iff it is a Noun Phrase of type 0, 2, 3, or 4.

```
(EQUAL (NP X0 X1 N1 N2 V1 V2 A STAR Z FR)
       (IF (NP0 X0 X1 N1 N2 V1 V2 A STAR Z FR) T
       (IF (NP2 X0 X1 N1 N2 V1 V2 A STAR Z FR) T
       (IF (NP3 X0 X1 N1 N2 V1 V2 A STAR Z FR) T
           (NP4 X0 X1 N1 N2 V1 V2 A STAR Z FR) ))) )
```

The next definition is a recursive definition that states that a Noun Phrase List is either a Noun Phrase followed by a Noun Phrase List or null.

```
(EQUAL(NPL X0 X2 N1 N3 V1 V3 AL STAR Z FRL)
      (IF(IF(LISTP X0)
             (EX X1(EX N2(EX V2(EX A(EX Z2(EX FR
               (IF(NP X0 X1 N1 N2 V1 V2 A Z2 Z FR)
```

```
                (EX AL2(EX FRL2(IF(NPL X1 X2 N2 N3 V2 V3 AL2 STAR Z2 FRL2)
                                  (IF(EQUAL AL(CONS A AL2))
                                     (EQUAL FRL(CONS FR FRL2))
                                     NIL)
                                  NIL) ))
                     NIL) ))))))
           NIL)
         T
         (IF(EQUAL N3 N1)
             (IF(EQUAL Z STAR)
                (IF(EQUAL AL NIL)
                   (IF(EQUAL X0 X2)
                      (IF(EQUAL V1 V3)
                         (EQUAL FRL NIL)
                         NIL)NIL)NIL)NIL)NIL) ))
```

This definition states that the only verb groups according to this grammar are verbs.

```
(EQUAL(VG X Y V Z FR)
      (VERB X Y V Z FR))
```

This last definition states that a piece of text is a DCLaritive SENTence TRANsitive iff it is a Noun Phrase followed by a Verb Group followed by a Noun Phrase List.

```
(EQUAL(DCLSENTTRAN X0 X3 N0 N2 V0 V2 Z)
      (EX X1(EX X2(EX N1(EX V1(EX A(EX AL(EX FRS(EX STAR(EX STAR2
         (IF (NP X0 X1 N0 N1 V0 V1 A STAR2 Z FRS)
             (EX FRV
               (IF(VG X1 X2 (CONS A AL) STAR FRV)
                  (EX FRL2
                    (NPL X2 X3 N1 N2 V1 V2 AL STAR STAR2 FRL2))
               NIL))NIL)))))))))) )
```

7.2 EXAMPLE EXECUTION

The English sentence "SOME BOY THROWS THE BIG RED BALL IN TEXAS" is proven to be a DeCLarative SENTence TRANsitive. In the course of this proof SYMEVAL deduces that there is exactly one possible translation of this sentence into QCL augumented with a special THE function. This translation is the expression equal to Z in the result of the evaluation given below.

It is worth while noting that this proof involves at least 201 existentially quantified variables, and that SYMEVAL systematically eliminates each one of these quantifiers. The final result contains no quantifiers or defined symbols, but is logically equivalent in Schwind's theory to the original input expression.

```
The expression to be recursively simplified is:
(DCLSENTTRAN (QUOTE (SOME BOY THROWS THE BIG RED BALL IN TEXAS))
             NIL NIL N2 (QUOTE (X2 X3))
             V3 Z)
 I1:(DCLSENTTRAN (QUOTE (SOME BOY THROWS THE BIG RED BALL IN TEXAS))
                 NIL NIL N2 (QUOTE (X2 X3))
                 V3 Z)
    by use of: DCLSENTTRAN
 M1:(EX
   *1
   (EX
     *2
     (EX
       *3
       (EX
```

```
      *4
      (EX
        *5
        (EX *6
            (EX *7
                (EX *8
                    (EX *9
                        (IF (NP (QUOTE (SOME BOY THROWS THE BIG RED BALL IN
                                                          TEXAS))
                                *1 NIL *3 (QUOTE (X2 X3))
                                *4 *5 *9 Z *7)
                            (EX *10
                                (IF (VG *1 *2 (CONS *5 *6)
                                        *8 *10)
                                    (EX *11
                                        (NPL *2 NIL *3 N2 *4 V3 *6 *8 *9
                                             *11))
                                    NIL))
                                NIL))))))))))
I2:(NP (QUOTE (SOME BOY THROWS THE BIG RED BALL IN TEXAS))
       *1 NIL *3 (QUOTE (X2 X3))
       *4 *5 *9 Z *7)
   by use of: NP
M2:(IF (NP0 (QUOTE (SOME BOY THROWS THE BIG RED BALL IN TEXAS))
            *1 NIL *3 (QUOTE (X2 X3))
            *4 *5 *9 Z *7)
       T
       (IF (NP2 (QUOTE (SOME BOY THROWS THE BIG RED BALL IN TEXAS))
                *1 NIL *3 (QUOTE (X2 X3))
                *4 *5 *9 Z *7)
           T
           (IF (NP3 (QUOTE (SOME BOY THROWS THE BIG RED BALL IN TEXAS))
                    *1 NIL *3 (QUOTE (X2 X3))
                    *4 *5 *9 Z *7)
               T
               (NP4 (QUOTE (SOME BOY THROWS THE BIG RED BALL IN TEXAS))
                    *1 NIL *3 (QUOTE (X2 X3))
                    *4 *5 *9 Z *7))))
O2:(IF
  (EQUAL *7 (QUOTE (M THIRD)))
  (IF
    (EQUAL *1 (QUOTE (THROWS THE BIG RED BALL IN TEXAS)))
    (IF
      (EQUAL *4 (QUOTE (X3)))
      (IF
        *3 NIL
        (IF (EQUAL *5 (QUOTE X2))
            (EQUAL Z
                   (CONS (QUOTE EX)
                         (CONS (QUOTE X2)
                               (CONS (CONS (QUOTE AND)
                                           (CONS (QUOTE (BOY X2))
                                                 (CONS *9 NIL)))
                                     NIL))))
            NIL))
      NIL)
    NIL)
  NIL)
I2:(VG (QUOTE (THROWS THE BIG RED BALL IN TEXAS))
       *2
       (CONS (QUOTE X2)
             *6)
```

```
       *8 *10)
   by use of: VG
M2:(VERB (QUOTE (THROWS THE BIG RED BALL IN TEXAS))
         *2
         (CONS (QUOTE X2)
               *6)
         *8 *10)
O2:(IF (EQUAL *2 (QUOTE (THE BIG RED BALL IN TEXAS)))
       (IF (EQUAL *8 (CONS (QUOTE THROW1)
                           (CONS (QUOTE X2)
                                 *6)))
           (EQUAL *10 (QUOTE (AGENT DO DEST)))
           NIL)
       NIL)
I2:(NPL (QUOTE (THE BIG RED BALL IN TEXAS))
        NIL NIL N2 (QUOTE (X3))
        V3 *6 (CONS (QUOTE THROW1)
                    (CONS (QUOTE X2)
                          *6))
        *9 *11)
   by use of: NPL
M2:(IF
  (IF
    (LISTP (QUOTE (THE BIG RED BALL IN TEXAS)))
    (EX
      *63
      (EX
        *64
        (EX
          *65
          (EX
            *66
            (EX
              *67
              (EX
                *68
                (IF (NP (QUOTE (THE BIG RED BALL IN TEXAS))
                        *63 NIL *64 (QUOTE (X3))
                        *65 *66 *67 *9 *68)
                    (EX *69
                        (EX *70
                            (IF (NPL *63 NIL *64 N2 *65 V3 *69
                                     (CONS (QUOTE THROW1)
                                           (CONS (QUOTE X2)
                                                 *6))
                                     *67 *70)
                                (IF (EQUAL *6 (CONS *66 *69))
                                    (EQUAL *11 (CONS *68 *70))
                                    NIL)
                                NIL)))
                    NIL)))))))
   NIL)
 T
 (IF (EQUAL N2 NIL)
     (IF (EQUAL *9 (CONS (QUOTE THROW1)
                         (CONS (QUOTE X2)
                               *6)))
         (IF (EQUAL *6 NIL)
             (IF (EQUAL (QUOTE (THE BIG RED BALL IN TEXAS))
                        NIL)
                 (IF (EQUAL (QUOTE (X3))
                            V3)
```

```
                        (EQUAL *11 NIL)
                        NIL)
                    NIL)
                NIL)
            NIL)
         NIL))
   02:(IF
     (EQUAL N2 (QUOTE (((N THIRD) . TEXAS))))
     (IF
       (EQUAL *9 (CONS (QUOTE THROW1)
                       (CONS (QUOTE X2)
                             *6)))
       (IF V3 NIL
           (IF (EQUAL *6
                      (QUOTE ((THE X3 ((SING-PLURAL)
                                       (AND (AND (BIG X3)
                                                 (AND (RED X3)
                                                      T))
                                            (AND (BALL X3)
                                                 (AND (IN X3 TEXAS)
                                                      T))))))))
               (EQUAL *11 (QUOTE ((N THIRD))))
               NIL))
       NIL)
     NIL)
 01:(IF
   (EQUAL
     Z
     (QUOTE (EX X2
                (AND (BOY X2)
                     (THROW1 X2
                             (THE X3 ((SING-PLURAL)
                                      (AND (AND (BIG X3)
                                                (AND (RED X3)
                                                     T))
                                           (AND (BALL X3)
                                                (AND (IN X3 TEXAS)
                                                     T)))))))))
   (IF (EQUAL N2 (QUOTE (((N THIRD) . TEXAS))))
       (IF V3 NIL T)
       NIL)
   NIL)

The result of recursive simplification is:
(IF
  (EQUAL
    Z
    (QUOTE (EX X2 (AND (BOY X2)
                       (THROW1 X2
                               (THE X3 ((SING-PLURAL)
                                        (AND (AND (BIG X3)
                                                  (AND (RED X3)
                                                       T))
                                             (AND (BALL X3)
                                                  (AND (IN X3 TEXAS)
                                                       T)))))))))
  (IF (EQUAL N2 (QUOTE (((N THIRD) . TEXAS))))
      (IF V3 NIL T)
      NIL)
  NIL)
end of deduction
```

EX1=110/EX2=0/EX3=71/EX4=0/EX5=6/EX6=0/EX7=0/EX8=0/EX9=0/EX10=0/
ALL1=0/ALL2=0/ALL3=0/ALL4=0/ALL5=0/ALL6=0/ALL7=0/ALL8=0/ALL9=0/ALL10=0/

8. SET THEORY

Axioms for LAMBDA abstraction are part of the SYMEVAL-QCL system and are described in section 3.5. Set Theory is developed in terms of LAMBDA abstraction by first defineing set theoretic abstraction and elementhood in terms of LAMBDA abstraction and APPLYcation, and then by asserting which sets are good sets in the sense that they may be members of other sets.

ATP.SETDEF This file contains definitions and theorems for the set theory described in Quine's book: SET THEORY AND ITS LOGIC.

ATP.SET Extra code for implementing set theory. Not currently used.

EXAMPLE

SYMEVAL can prove that the Weiner-Kurtowski set theoretic definition of an ordered pair is in fact an ordered pair. This proof is obtaind without the use of any lemmas whatsoever, and in fact in the course of the proof SYMEVAL proves a number of interesting lemmas about the equality of unordered pairs and unit sets. The only other automatic proof of this theorem that we are aware of in the literature is the sequent calculas based proof in [Brown6] which assumed several lemmas about unordered pairs and unitsets. (That sequent calculas theorem prover could prove the lemmas that were assumed if they were explicitly given to it.)

In order to state the ordered pair theorem, quantifiers whose bound variables range over anything but bad sets are declared

```
(SETQ QUANTSYM(APPEND '(QALL QEX SET) QUANTSYM))
(VASSUME 'SET (REMOVE 'BADLAMBDAP UNIVERSE))
(VASSUME 'QALL (REMOVE 'BADLAMBDAP UNIVERSE))
(VASSUME 'QEX  (REMOVE 'BADLAMBDAP UNIVERSE))
```

and then the following definitions are made:

```
(DCLLQ(EQUAL (ELE X Y) (AP Y X))
      (EQUAL (SET X(SCH1 X)) (LAMBDA X(SCH1 X)))
      (EQUAL (QALL X(SCH1 X)) (ALL X(IF (BADLAMBDAP X) T (SCH1 X))))
      (EQUAL (QEX X(SCH2 X)) (EX X(AND (NOT(BADLAMBDAP X))(SCH2 X))))
      (EQUAL (EQUALSETS A B) (QALL X(IFF(ELE X A)(ELE X B))) )
      (EQUAL (UNITSET A) (SET X(EQUAL X A)) )
      (EQUAL (PAIRSET A B) (SET X (LOR(EQUAL X A)(EQUAL X B))) )
      (EQUAL (ORDPAIRSET A B) (PAIRSET(UNITSET A)(PAIRSET A B)) )
```

Two axioms of set theory are assumed, namely that unit sets and unordered pairsets are not BADLAMBDAPs:

```
AX5:  (EQUAL(BADLAMBDAP(LAMBDA X(EQUAL X A))) NIL)
AX6:  (EQUAL(BADLAMBDAP(LAMBDA X(IF(EQUAL X A)T(EQUAL X B)))) NIL)
```

The ordered pair theorem: that two ordered pairs are equalsets iff their components are equal is now proven.

```
The expression to be recursively simplified is:
(QALL X (QALL Y (QALL U (QALL V (IFF (EQUALSETS (ORDPAIRSET X Y)
                                                (ORDPAIRSET U V))
                                     (AND (EQUAL X U)
                                          (EQUAL Y V)))))))

 I1:(ORDPAIRSET X Y)
    by use of: ORDPAIRSET
 M1:(PAIRSET (UNITSET X)
```

```
            (PAIRSET X Y))
   I2:(UNITSET X)
      by use of: UNITSET
   M2:(SET *1 (EQUAL *1 X))
   O2:(LAMBDA *1 (EQUAL *1 X))
   I2:(PAIRSET X Y)
      by use of: PAIRSET
   M2:(SET *2 (LOR (EQUAL *2 X)
                   (EQUAL *2 Y)))
   O2:(LAMBDA *2 (IF (EQUAL *2 X)
                     T
                     (EQUAL *2 Y)))
   I2:(PAIRSET (LAMBDA *1 (EQUAL *1 X))
               (LAMBDA *2 (IF (EQUAL *2 X)
                              T
                              (EQUAL *2 Y))))
      by use of: PAIRSET
   M2:(SET *3 (LOR (EQUAL *3 (LAMBDA *1 (EQUAL *1 X)))
                   (EQUAL *3 (LAMBDA *2 (IF (EQUAL *2 X)
                                            T
                                            (EQUAL *2 Y))))))
   O2:(LAMBDA *3 (IF (EQUAL *3 (LAMBDA *1 (EQUAL *1 X)))
                     T
                     (EQUAL *3 (LAMBDA *2 (IF (EQUAL *2 X)
                                              T
                                              (EQUAL *2 Y))))))
O1:(LAMBDA *3 (IF (EQUAL *3 (LAMBDA *1 (EQUAL *1 X)))
                  T
                  (EQUAL *3 (LAMBDA *2 (IF (EQUAL *2 X)
                                           T
                                           (EQUAL *2 Y))))))
I1:(ORDPAIRSET U V)
   by use of: ORDPAIRSET
M1:(PAIRSET (UNITSET U)
            (PAIRSET U V))
  I2:(UNITSET U)
     by use of: UNITSET
  M2:(SET *4 (EQUAL *4 U))
  O2:(LAMBDA *4 (EQUAL *4 U))
  I2:(PAIRSET U V)
     by use of: PAIRSET
  M2:(SET *5 (LOR (EQUAL *5 U)
                  (EQUAL *5 V)))
  O2:(LAMBDA *5 (IF (EQUAL *5 U)
                    T
                    (EQUAL *5 V)))
  I2:(PAIRSET (LAMBDA *4 (EQUAL *4 U))
              (LAMBDA *5 (IF (EQUAL *5 U)
                             T
                             (EQUAL *5 V))))
     by use of: PAIRSET
  M2:(SET *6 (LOR (EQUAL *6 (LAMBDA *4 (EQUAL *4 U)))
                  (EQUAL *6 (LAMBDA *5 (IF (EQUAL *5 U)
                                           T
                                           (EQUAL *5 V))))))
  O2:(LAMBDA *6 (IF (EQUAL *6 (LAMBDA *4 (EQUAL *4 U)))
                    T
                    (EQUAL *6 (LAMBDA *5 (IF (EQUAL *5 U)
                                             T
                                             (EQUAL *5 V))))))
O1:(LAMBDA *6 (IF (EQUAL *6 (LAMBDA *4 (EQUAL *4 U)))
                  T
```

```
                (EQUAL *6 (LAMBDA *5 (IF (EQUAL *5 U)
                                         T
                                         (EQUAL *5 V))))))
I1:(EQUALSETS (LAMBDA *3 (IF (EQUAL *3 (LAMBDA *1 (EQUAL *1 X)))
                             T
                             (EQUAL *3 (LAMBDA *2 (IF (EQUAL *2 X)
                                                      T
                                                      (EQUAL *2 Y))))))
              (LAMBDA *6 (IF (EQUAL *6 (LAMBDA *4 (EQUAL *4 U)))
                             T
                             (EQUAL *6 (LAMBDA *5 (IF (EQUAL *5 U)
                                                      T
                                                      (EQUAL *5 V)))))))
   by use of: EQUALSETS
M1:(QALL *7 (IFF (ELE *7 (LAMBDA *3 (IF (EQUAL *3 (LAMBDA *1
                                                  (EQUAL *1 X)))
                                        T
                                        (EQUAL *3 (LAMBDA
                                                  *2
                                                  (IF (EQUAL *2 X)
                                                      T
                                                      (EQUAL *2 Y)))))))
                 (ELE *7 (LAMBDA *6 (IF (EQUAL *6 (LAMBDA *4
                                                  (EQUAL *4 U)))
                                        T
                                        (EQUAL *6 (LAMBDA
                                                  *5
                                                  (IF (EQUAL *5 U)
                                                      T
                                                      (EQUAL *5 V)))))))))
  I2:(IFF (IF (EQUAL *7 (LAMBDA *1 (EQUAL *1 X)))
              T
              (EQUAL *7 (LAMBDA *2 (IF (EQUAL *2 X)
                                       T
                                       (EQUAL *2 Y)))))
          (IF (EQUAL *7 (LAMBDA *4 (EQUAL *4 U)))
              T
              (EQUAL *7 (LAMBDA *5 (IF (EQUAL *5 U)
                                       T
                                       (EQUAL *5 V))))))
     by use of: IFF
  M2:(IF (IF (EQUAL *7 (LAMBDA *1 (EQUAL *1 X)))
             T
             (EQUAL *7 (LAMBDA *2 (IF (EQUAL *2 X)
                                      T
                                      (EQUAL *2 Y)))))
         (IF (IF (EQUAL *7 (LAMBDA *4 (EQUAL *4 U)))
                 T
                 (EQUAL *7 (LAMBDA *5 (IF (EQUAL *5 U)
                                          T
                                          (EQUAL *5 V)))))
             T NIL)
         (IF (IF (EQUAL *7 (LAMBDA *4 (EQUAL *4 U)))
                 T
                 (EQUAL *7 (LAMBDA *5 (IF (EQUAL *5 U)
                                          T
                                          (EQUAL *5 V)))))
             NIL T))
    I3:(EQUAL (LAMBDA *1 (EQUAL *1 X))
              (LAMBDA *4 (EQUAL *4 U)))
       by use of: (LISPLINK SYMEQUAL)
    M3:(ALL *8 (EQUAL (AP (LAMBDA *1 (EQUAL *1 X))
```

```
                    *8)
                (AP (LAMBDA *4 (EQUAL *4 U))
                    *8)))
03:(EQUAL U X)
I3:(EQUAL (LAMBDA *1 (EQUAL *1 X))
          (LAMBDA *5 (IF (EQUAL *5 U)
                         T
                         (EQUAL *5 V))))
   by use of: (LISPLINK SYMEQUAL)
M3:(ALL *9 (EQUAL (AP (LAMBDA *1 (EQUAL *1 X))
                      *9)
                  (AP (LAMBDA *5 (IF (EQUAL *5 U)
                                     T
                                     (EQUAL *5 V)))
                      *9)))
03:NIL
I3:(EQUAL (LAMBDA *2 (IF (EQUAL *2 X)
                         T
                         (EQUAL *2 Y)))
          (LAMBDA *4 (EQUAL *4 U)))
   by use of: (LISPLINK SYMEQUAL)
M3:(ALL *10 (EQUAL (AP (LAMBDA *2 (IF (EQUAL *2 X)
                                      T
                                      (EQUAL *2 Y)))
                       *10)
                   (AP (LAMBDA *4 (EQUAL *4 U))
                       *10)))
03:(IF (EQUAL U X)
       (EQUAL Y X)
       NIL)
I3:(EQUAL (LAMBDA *2 (IF (EQUAL *2 X)
                         T
                         (EQUAL *2 Y)))
          (LAMBDA *5 (IF (EQUAL *5 U)
                         T
                         (EQUAL *5 V))))
   by use of: (LISPLINK SYMEQUAL)
M3:(ALL *11 (EQUAL (AP (LAMBDA *2 (IF (EQUAL *2 X)
                                      T
                                      (EQUAL *2 Y)))
                       *11)
                   (AP (LAMBDA *5 (IF (EQUAL *5 U)
                                      T
                                      (EQUAL *5 V)))
                       *11)))
03:(IF (EQUAL X U)
       (IF (EQUAL Y U)
           (EQUAL V U)
           (EQUAL Y V))
       (IF (EQUAL X V)
           (IF (EQUAL Y V)
               NIL
               (EQUAL Y U))
           NIL))

02:(IF (EQUAL *7 (LAMBDA *1 (EQUAL *1 X)))
       (EQUAL U X)
       (IF (EQUAL *7 (LAMBDA *2 (IF (EQUAL *2 X)
                                    T
                                    (EQUAL *2 Y))))
           (IF (EQUAL U X)
               (IF (EQUAL Y X)
```

```
                    T
                    (EQUAL Y V))
                (IF (EQUAL X V)
                    (IF (EQUAL Y V)
                        NIL
                        (EQUAL Y U))
                    NIL))
            (IF (EQUAL *7 (LAMBDA *4 (EQUAL *4 U)))
                NIL
                (IF (EQUAL *7 (LAMBDA *5 (IF (EQUAL *5 U)
                                             T
                                             (EQUAL *5 V))))
                    NIL T))))
I2:(QALL *7 (IF (EQUAL *7 (LAMBDA *1 (EQUAL *1 X)))
                (EQUAL U X)
                (IF (EQUAL *7 (LAMBDA *2 (IF (EQUAL *2 X)
                                             T
                                             (EQUAL *2 Y))))
                    (IF (EQUAL U X)
                        (IF (EQUAL Y X)
                            T
                            (EQUAL Y V))
                        (IF (EQUAL X V)
                            (IF (EQUAL Y V)
                                NIL
                                (EQUAL Y U))
                            NIL))
                    (IF (EQUAL *7 (LAMBDA *4 (EQUAL *4 U)))
                        NIL
                        (IF (EQUAL *7 (LAMBDA *5 (IF (EQUAL *5 U)
                                                     T
                                                     (EQUAL *5 V))))
                            NIL T)))))
   by use of: QALL
M2:(ALL *12
        (IF (BADLAMBDAP *12)
            T
            (IF (EQUAL *12 (LAMBDA *1 (EQUAL *1 X)))
                (EQUAL U X)
                (IF (EQUAL *12 (LAMBDA *2 (IF (EQUAL *2 X)
                                              T
                                              (EQUAL *2 Y))))
                    (IF (EQUAL U X)
                        (IF (EQUAL Y X)
                            T
                            (EQUAL Y V))
                        (IF (EQUAL X V)
                            (IF (EQUAL Y V)
                                NIL
                                (EQUAL Y U))
                            NIL))
                    (IF (EQUAL *12 (LAMBDA *4 (EQUAL *4 U)))
                        NIL
                        (IF (EQUAL *12 (LAMBDA *5 (IF (EQUAL *5 U)
                                                      T
                                                      (EQUAL *5 V))))
                            NIL T))))))
  I3:(BADLAMBDAP (LAMBDA *1 (EQUAL *1 X)))
     by use of: AX5
  M3:NIL
  03:NIL
  I3:(BADLAMBDAP (LAMBDA *2 (IF (EQUAL *2 X)
```

```
                                    T
                                    (EQUAL *2 Y))))
      by use of: AX6
    M3:NIL
    03:NIL
    I3:(BADLAMBDAP (LAMBDA *4 (EQUAL *4 X)))
      by use of: AX5
    M3:NIL
    03:NIL
    I3:(BADLAMBDAP (LAMBDA *5 (IF (EQUAL *5 X)
                                  T
                                  (EQUAL *5 V))))
      by use of: AX6
    M3:NIL
    03:NIL
    I3:(BADLAMBDAP (LAMBDA *4 (EQUAL *4 X)))
      by use of: AX5
    M3:NIL
    03:NIL
    I3:(BADLAMBDAP (LAMBDA *5 (IF (EQUAL *5 X)
                                  T
                                  (EQUAL *5 V))))
      by use of: AX6
    M3:NIL
    03:NIL
  02:(IF (EQUAL U X)
         (IF (EQUAL Y X)
             (EQUAL V X)
             (EQUAL Y V))
         NIL)
01:(IF (EQUAL U X)
       (IF (EQUAL Y X)
           (EQUAL V X)
           (EQUAL Y V))
       NIL)
I1:(IFF (IF (EQUAL U X)
            (IF (EQUAL Y X)
                (EQUAL V X)
                (EQUAL Y V))
            NIL)
        (IF (EQUAL X U)
            (EQUAL Y V)
            NIL))
   by use of: IFF
M1:(IF (IF (EQUAL U X)
           (IF (EQUAL Y X)
               (EQUAL V X)
               (EQUAL Y V))
           NIL)
       (IF (IF (EQUAL X U)
               (EQUAL Y V)
               NIL)
           T NIL)
       (IF (IF (EQUAL X U)
               (EQUAL Y V)
               NIL)
           NIL T))
01:T
I1:(QALL V T)
   by use of: QALL
M1:(ALL *21 (IF (BADLAMBDAP *21)
                T T))
```

```
  O1:T
  I1:(QALL U T)
     by use of: QALL
  M1:(ALL *22 (IF (BADLAMBDAP *22)
                  T T))
  O1:T
  I1:(QALL Y T)
     by use of: QALL
  M1:(ALL *23 (IF (BADLAMBDAP *23)
                  T T))
  O1:T
  I1:(QALL X T)
     by use of: QALL
  M1:(ALL *24 (IF (BADLAMBDAP *24)
                  T T))
  O1:T

The result of recursive simplification is:
T
which is true. QED.

EX1=0/EX2=0/EX3=0/EX4=0/EX5=0/EX6=0/EX7=0/EX8=0/EX9=0/EX10=0/
ALL1=0/ALL2=0/ALL3=22/ALL4=0/ALL5=0/ALL6=10/ALL7=0/ALL8=0/ALL9=0/ALL10=0/
```

9. ONTOLOGY

Lesniewski's Ontology[Luschei,Henry] is a set theory which grew out of the traditions of Medieval logic. Its "sets" closely correspond to noun phrases, including names, fictitious names(eg. Pegasus) and more general nouns. Its "elementhood" predicate corresponds to to the intransitive verb IS in English, and more closely to the Latin EST.

ATP.LES This file contains some definitions for Lesniewski's theory of Ontology.

EXAMPLE SYMEVAL can prove that '(Z X Y) is a permutation of '(X Y Z). In order to do this, we first define recursive ontological definitions of of the notion of a permutation:

```
(DCLLQ
(EQUAL(PERMSET L)(IF (EQUAL  L (NILSET))
                     NIL
                     (INSERTSET (CAR L)(PERMSET (CDR L)))))
(EQUAL(NILSET) (LAMBDA X (EQUAL X NIL)))
(EQUAL(CONSET A B)
      (LAMBDA X(EX Y(EX Z(AND(IS Y A)(AND(IS Z B)(EQUAL X(CONS Y Z))))))))
(EQUAL(INSERTSET X L)
      (IF(EQUAL L (NILSET))
         (CONS X NIL)
         (NOMINAL.OR(CONSET X L)
                    (LAMBDA Y(EX Z(AND(IS Z L)
                                      (IS Y(CONSET(CAR Z)
                                                  (INSERTSET X(CDR Z))))))))))
(EQUAL(NOMINAL.OR A B)
      (LAMBDA X (LOR(IS X A)(IS X B)))))


(SETQ TRACELIST '(NOMINAL.OR CONSET PERMSET INSERTSET))

A proof of the Ontological theorem:
      (IS (QUOTE(Z Y X)) (PERMSET(QUOTE(X Y Z))))
is given below.  The proof was edited by deleting most traces
less than level 2.

The expression to be recursively simplified is:
(IS (QUOTE (Z Y X))
    (PERMSET (QUOTE (X Y Z))))
  I1:(PERMSET (QUOTE (X Y Z)))
     by use of: PERMSET
  M1:(IF (EQUAL (QUOTE (X Y Z))
                (NILSET))
         NIL
         (INSERTSET (CAR (QUOTE (X Y Z)))
                    (PERMSET (CDR (QUOTE (X Y Z))))))
    I2:(PERMSET (QUOTE (Y Z)))
       by use of: PERMSET
    M2:(IF (EQUAL (QUOTE (Y Z))
                  (NILSET))
           NIL
           (INSERTSET (CAR (QUOTE (Y Z)))
                      (PERMSET (CDR (QUOTE (Y Z))))))
      I3:(PERMSET (QUOTE (Z)))
         by use of: PERMSET
      M3:(IF (EQUAL (QUOTE (Z))
                    (NILSET))
             NIL
```

```
          (INSERTSET (CAR (QUOTE (Z)))
                     (PERMSET (CDR (QUOTE (Z))))))
     I4:(PERMSET NIL)
        by use of: PERMSET
     M4:(IF (EQUAL NIL (NILSET))
            NIL
            (INSERTSET (CAR NIL)
                       (PERMSET (CDR NIL))))
     O4:NIL
   O3:(QUOTE (Z))
   I3:(INSERTSET (QUOTE Y)
                 (QUOTE (Z)))
      by use of: INSERTSET
   M3:(IF (EQUAL (QUOTE (Z))
                 (NILSET))
          (CONS (QUOTE Y)
                NIL)
          (NOMINAL.OR (CONSET (QUOTE Y)
                              (QUOTE (Z)))
                      (LAMBDA
                        *8
                        (EX *9 (AND (IS *9 (QUOTE (Z)))
                                    (IS *8 (CONSET (CAR *9)
                                                   (INSERTSET (QUOTE Y)
                                                              (CDR *9))))))))))
          )
   O3:(LAMBDA *20 (IF (EQUAL (QUOTE (Y Z))
                             *20)
                      T
                      (EQUAL (QUOTE (Z Y))
                             *20)))
O2:(LAMBDA *20 (IF (EQUAL (QUOTE (Y Z))
                          *20)
                   T
                   (EQUAL (QUOTE (Z Y))
                          *20)))
I2:(INSERTSET (QUOTE X)
              (LAMBDA *20 (IF (EQUAL (QUOTE (Y Z))
                                     *20)
                              T
                              (EQUAL (QUOTE (Z Y))
                                     *20))))
   by use of: INSERTSET
M2:(IF (EQUAL (LAMBDA *20 (IF (EQUAL (QUOTE (Y Z))
                                     *20)
                              T
                              (EQUAL (QUOTE (Z Y))
                                     *20)))
              (NILSET))
       (CONS (QUOTE X)
             NIL)
       (NOMINAL.OR
         (CONSET (QUOTE X)
                 (LAMBDA *20 (IF (EQUAL (QUOTE (Y Z))
                                        *20)
                                 T
                                 (EQUAL (QUOTE (Z Y))
                                        *20))))
         (LAMBDA *21
                 (EX *22 (AND (IS *22 (LAMBDA
                                        *20
                                        (IF (EQUAL (QUOTE (Y Z))
```

```
                                                    *20)
                                                T
                                                (EQUAL (QUOTE (Z Y))
                                                       *20))))
                                        (IS *21 (CONSET (CAR *22)
                                                        (INSERTSET (QUOTE X)
                                                                   (CDR *22)))))))))
  02:(LAMBDA *61
             (IF (EQUAL *61 (QUOTE (X Y Z)))
                 T
                 (IF (EQUAL *61 (QUOTE (X Z Y)))
                     T
                     (IF (EQUAL *61 (QUOTE (Y X Z)))
                         T
                         (IF (EQUAL *61 (QUOTE (Y Z X)))
                             T
                             (IF (EQUAL *61 (QUOTE (Z X Y)))
                                 T
                                 (EQUAL *61 (QUOTE (Z Y X)))))))))
01:(LAMBDA *61
           (IF (EQUAL *61 (QUOTE (X Y Z)))
               T
               (IF (EQUAL *61 (QUOTE (X Z Y)))
                   T
                   (IF (EQUAL *61 (QUOTE (Y X Z)))
                       T
                       (IF (EQUAL *61 (QUOTE (Y Z X)))
                           T
                           (IF (EQUAL *61 (QUOTE (Z X Y)))
                               T
                               (EQUAL *61 (QUOTE (Z Y X)))))))))
```

The result of recursive simplification is:
T
which is true. QED.

EX0=0/EX1=6/EX2=0/EX3=26/EX4=0/EX5=6/EX6=4/EX7=0/EX8=0/EX9=0/EX10=0/
ALL0=0/ALL1=0/ALL2=0/ALL3=1/ALL4=0/ALL5=0/ALL6=1/ALL7=0/ALL8=0/ALL9=0/ALL10=0/

10. COMPLEXITY PROJECT

ATP.COMPLEXITY contains four functions listed below:

(ANALYZE function.definition.being.analyzed basis.function.definition) ANALYZE tries to determine if the complexity of function definition being analyzed is linearly related to the given basis function. Later versions will handle multiple basis functions and non linear relationships. ANALYZE calls the routines COMFUN SIMFUN and the automatic theorem prover.

(COMFUN definition) Computes the complexity function definition of a given function definition.

(SIMFUN definition) Computes the simplified version of a function definition by deleting extraneous argument positions.

(BASISFUNS definition) Computes the immediate basis function definitions of a given complexity function definition.

The Complexity Analysis Project is concerned with the development of a reasoning system to automatically analyze and determine the complexity of computer programs. This research is important not only for theoretical computer science in providing a method for automating the process of analysing the complexity of algorithms, but also for the practical problem of verifying time dependent properties of computer programs used in such real time areas as flight control systems. The current complexity analysis system called ANALYZE is capable of for automatically analysing the complexity of simple recursive LISP functions ANALYZE calls on our automatic deduction system SYMEVAL in a number of places in order to achieve its results. ANALYZE is described in section 1, and the use it makes of SYMEVAL is exemplified in section 2.

10.1 COMPLEXITY ANALYSIS SYSTEM: ANALYZE

We have developed a prototype system called ANALYZE for analyzing the complexity of recursive LISP functions. The basic approach to automatic program analysis used by ANALYZE is this: The user specifies a recursive LISP function F of which he wishes to analyze the complexity. The user may also specify that the analysis is to be performed in terms of certain basis functions which essentially compute the size of the input data of the original function. The system then does the following:

(STEP 1) First, the COMplexity FUNction subsystem, called COMFUN automatically produces a new LISP function C.F which computes the complexity of F. This function is created by mimicking the recursive structure of F indicating the complexity of each branch.

(STEP 2) Second, the SIMplification FUNction subsystem, called SIMFUN tries to simplify the C.F function by deleting irrelevant argument positions and by SYMbolically EVALuating the function body.

(STEP 3) If they are not already specified, then the BASis FUNction subsystem, called BASFUN automatically produces the possible appropriate basis functions. A basis function is a function which measures the size of some data object such as, for example, a tree.

(STEP 4) Fourth, the system tries to guess a closed form solution to C.F in terms of the basis functions.

(STEP 5) Finally, using existential variables for coefficients, the system tries to prove that the recursive complexity function C.F equals the conjectured closed form solution. In the course of the proof, the system may automatically determine explicit values for those existential variables.

The result of steps (1)-(5) is an algebraic formula expressing the complexity of F in terms of (a) the size of the data object to which F is applied, and (b) the complexity of its subroutines. If the complexity of each subroutine and any subroutines called by such subroutines is determined by repeating steps (1)-(5), the complexity of F is then expressed as an algebraic formula containing only the complexity of primitive instructions and the size of the input data objects. The deductive parts of the system are based on the SYMEVAL theorem prover, the SYMMETRIC LOGIC, and the Real Algebra rule package. One novel aspect of this deduction system is that it integrates general structural inductive capabilities over arbitrary data objects along the lines of [Boyer1] with quantifier elimination techniques of the SYMMETRIC LOGIC and equation solving techniques of the real algebra theorem prover[Brown24]. Some inductive parts of the system have been studied earlier in collaboration with Prof. Sten-Ake Tarnlund of Upsalla University[Brown5,10].

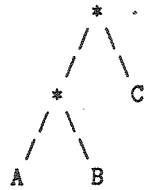
A SIMPLE EXAMPLE We suppose that the user of our proposed system wishes to analyze the complexity of the function FRINGE which computes the fringe of a binary tree T1 when L=NIL in terms of the number of nodes in the tree T1. The (FRINGE L NIL) of a binary tree is a list of its leaves. The definition of FRINGE is:

```
(EQUAL (FRINGE T1 L)
       (IF (LISTP T1)
           (FRINGE (CAR T1) (FRINGE (CDR T1) L))
           (CONS T1 L)))
```

The definition of the function NODEKNT which counts nodes in a tree is:

```
(EQUAL (NODEKNT T1)
       (IF (LISTP T1)
           (PLUS 1 (PLUS (NODEKNT (CAR T1)) (NODEKNT (CDR T1))))
           1))
```

The binary tree is assumed to be constructed from LISP CONSes. For example (CONS(CONS A B)C) represents the tree:



The LISP functions are all written in the SYMEVAL's logical language, which includes logical expressions, real numbers and recursive functions of pure LISP. We write (IF p x y) instead of the usual LISP conditional (COND(p x)(T y)).

We ask the system to try to analyze the complexity of FRINGE in terms of the function NODEKNT. The following is a trace of the complexity systems reasoning:

```
7_(ANALYZE FRINGE NODEKNT)

We are trying to determine whether the complexity of:
(EQUAL (FRINGE T1 L)
       (IF (LISTP T1)
           (FRINGE (CAR T1)
                   (FRINGE (CDR T1)
                           L))
           (CONS T1 L)))
is related to the basis function:
(EQUAL (NODEKNT T1)
       (IF (LISTP T1)
```

```
          (PLUS 1 (PLUS (NODEKNT (CAR T1))
                        (NODEKNT (CDR T1))))
          1))
```

(STEP 1) After the system states the problem, the subsystem COMFUN creates a recursive function which computes the abstract complexity of FRINGE:
The local constant (A0007) is the complexity of taking the true branch of the LISTP test except for the complexity of the recursive calls to FRINGE which are mentioned explicitly. The local constant (A0008) is the complexity of taking the false branch of the test.

The complexity function of FRINGE is:

```
(EQUAL (C.FRINGE T1 L)
       (IF (LISTP T1)
           (PLUS (A0007)
                 (PLUS (C.FRINGE (CAR T1)
                                 (FRINGE (CDR T1)
                                         L))
                       (C.FRINGE (CDR T1)
                                 L)))
           (A0008)))
```

where the local constants are defined as follows in terms of the complexities of the primitive operations of LISP. For example (C.LISTP) is the complexity of executing the LISTP function and (C-VAR) is the complexity of looking up the value of a variable in a shallow binding environment.

```
((EQUAL (A0007)
        (PLUS (C.IF.T)
              (PLUS (C.LISTP)
                    (PLUS (C-VAR)
                          (PLUS (TIMES 2 (C-BIND))
                                (PLUS (C.CAR)
                                      (PLUS (C-VAR)
                                            (PLUS (C.CDR)
                                                  (PLUS (C-VAR)
                                                        (C-VAR)))))))))))
 (EQUAL (A0008)
        (PLUS (C.IF.NIL)
              (PLUS (C.LISTP)
                    (PLUS (C-VAR)
                          (PLUS (TIMES 2 (C-BIND))
                                (PLUS (C.CONS)
                                      (PLUS (C-VAR)
                                            (C-VAR)))))))))
```

(STEP 2) The subsystem SIMFUN now tries to simplify the complexity function definition just produced:

Observing that the variable L is
not used in the body of the definition, it follows
that the complexity function simplifies to the new complexity function:

```
(EQUAL (C.FRINGE T1)
       (IF (LISTP T1)
           (PLUS (A0007)
                 (PLUS (C.FRINGE (CAR T1))
                       (C.FRINGE (CDR T1))))
           (A0008)))
```

(STEP 3) Step three is omitted in this example because we already suggested to the system that NODEKNT was an appropriate basis function.

(STEP 4) An appropriate complexity conjecture relating C.FRINGE to NODEKNT is now produced:

You hinted that the complexity of FRINGE

```
 was related to the basis function: NODEKNT.

We will now try to see if its linearly related to that basis
by first forming an expression stating that fact:

(ALL T1 (EQUAL (C.FRINGE T1)
               (PLUS (TIMES X (NODEKNT T1))
                     Y)))
and then simplifying this expression as much as possible using
our automatic theorem prover.  The result returned by our theorem
prover will be logically equivalent to this original expression.

(STEP 5) The conjectured relation between the complexity function and the
basis is now proven:

The first SYMbolic EVALuation

The expression to be recursively simplified is:
(ALL T1 (EQUAL (C.FRINGE T1)
               (PLUS (TIMES X (NODEKNT T1))
                     Y)))
The result of recursive simplification is:
(ALL T1 (EQUAL (PLUS (C.FRINGE T1)
                     (PLUS (MINUS (TIMES X (NODEKNT T1)))
                           (MINUS Y)))
               0))

Induction is now tried giving a new expression to simplify.

The expression to be recursively simplified is:
(AND
  (ALL T1 (IMPLIES (NOT (LISTP T1))
                   (EQUAL (PLUS (C.FRINGE T1)
                                (PLUS (MINUS (TIMES X (NODEKNT T1)))
                                      (MINUS Y)))
                          0)))
  (ALL
    T1
    (IMPLIES
      (AND (LISTP T1)
           (AND (EQUAL (PLUS (C.FRINGE (CAR T1))
                             (PLUS (MINUS (TIMES X (NODEKNT (CAR T1))))
                                   (MINUS Y)))
                       0)
                (EQUAL (PLUS (C.FRINGE (CDR T1))
                             (PLUS (MINUS (TIMES X (NODEKNT (CDR T1))))
                                   (MINUS Y)))
                       0)))
      (EQUAL (PLUS (C.FRINGE T1)
                   (PLUS (MINUS (TIMES X (NODEKNT T1)))
                         (MINUS Y)))
             0))))
The result of recursive simplification is:
(IF (EQUAL (PLUS (A0008)
                 (PLUS (MINUS X)
                       (MINUS Y)))
           0)
    (EQUAL (PLUS (A0007)
                 (PLUS Y (MINUS X)))
           0)
    NIL)
end of deduction
```

```
63798 conses
174.162 seconds
11.479 seconds, garbage collection time

We call the theorem prover again, this time letting
it solve for the unknowns
The expression to be recursively simplified is:
(IF (EQUAL (PLUS (A0008)
                 (PLUS (MINUS X)
                       (MINUS Y)))
           0)
    (EQUAL (PLUS (A0007)
                 (PLUS Y (MINUS X)))
           0)
    NIL)

The result of recursive simplification is:
(IF (EQUAL (PLUS (A0008)
                 (PLUS (MINUS X)
                       (MINUS Y)))
           0)
    (EQUAL (PLUS (A0007)
                 (PLUS (TIMES -2 X)
                       (A0008)))
           0)
    NIL)
end of deduction
1412 conses
2.685 seconds
1.836 seconds, garbage collection time
```

Observing that (IF p x NIL) means (AND p x) we see that the Automatic theorem prover has simplified the original closed form expression to an equivalent expression which is essentialy a conjuction of linear equations which when solved give explicit values for the unknowns X and Y. By solving these two linear equations we see that:

```
X =  ((A0007) + (A0008)) /2
Y =  ((A0008) - (A0007)) /2
```

where (A0007) and (A0008) are defined by the local definitions which in turn are defined in terms of the complexities of the primitive LISP operations.

Thus not only has the system proven the theorem:

```
(EX X(EX Y
(ALL T1(EQUAL(C.FRINGE T1)
             (PLUS (TIMES X (NODEKNT T1)) Y)))
))
```

where the unknowns X and Y are interpreted as being existentially quantified but in the course of proving this theorem, it has computed the only possible values for X and Y which make the expression true. Thus, in fact it proves the stronger theorem:

```
(ALL T1(EQUAL(C.FRINGE T1)
             (PLUS(TIMES ((A0007)+(A0008))/2 (NODEKNT T1))
                  ((A0008)-(A0007))/2 (NODEKNT T1) )))
```

Since the deductive system itself can handle existential variables, this greatly eases the burden on the inductive step (4) of the proposed system since that step will not have to worry about guessing the exact coefficients of a conjecture of a closed form solution.

10.2 USING THE SYMBOLIC EVALUATOR: SYMEVAL

Although SYMEVAL is used by ANALYZE in a number of places, for example to simplify the bodies of function definitions, its major use is in step 5 where it is used to prove the equivlence between the closed form solution and the original complexity function. It is therefore worthwhile looking at this reasoning step in more detail in order to describe SYMEVAL's current abilities. The outline of the proof qiven below is presented in the manner as one might trace the execution of a LISP program. Essentially an input expression labeled In: is given to SYMEVAL which by application of an axiom produces a middle expression labeled Mn: which is then recursively simplified producing an output expression labeled On:. The key point, is that In: is logically equivalent in the given theory to the immediately following Mn: and also to the immediately following On:. The n refers to the current level of tracing. By specifying what symbols to trace, SYMEVAL can be asked to present its reasoning at different levels of detail. In the following proof only a few key symbols have been traced, and a number of less important steps have been eliminated by hand. Nothing however has been added except English text. This proof involves a number basic deduction facilities including the nine listed below. The first use in this proof of each of these nine facilities is marked by the same number.

```
(1).Methods for deciding when to repace definitions including recursive
    definitions by their body.
(2).Rules for the algebraic simplification of expressions about real numbers.
(3).A rule for Noetherian Induction over arbitrary recursively construced
    data structures and recursive definitions.
(4).Propositional Logic based on an IF_THEN_ELSE_ construct.
(5).Rules of a Quantificational Logic based on the SYMMETRIC LOGIC
    of reducing the scope of quantifiers.
(6).The ability to return useful information as answers to subgoals rather
    than having to return True or False.
(7).The ability to solve equations for interesting expressions which
    can be substituted into other expressions so as to help solve the
    problem.
(8).Axioms about recursive data structures
(9).Instantiation Rules for Quantificational Logic.  Note that each
    induction hypothesis is eliminated by noting that it is equivalent
    to true assuming the linear equation produced by the base case.

The proof is now given:

The expression to be recursively simplified is:
(ALL T1 (EQUAL (C.FRINGE T1)
               (PLUS (TIMES X (NODEKNT T1))
                     Y)))

(1)SYMEVAL expands the definition of C.FRINGE and then changes its "mind".

  I1:(C.FRINGE T1)
     by use of: C.FRINGE
  M1:(IF (LISTP T1)
         (PLUS (A0007)
               (PLUS (C.FRINGE (CAR T1))
                     (C.FRINGE (CDR T1))))
         (A0008))

  O1:(C.FRINGE T1)

(2)The Real algebra equality rule is applied.

  I1:(EQUAL (C.FRINGE T1)
            (PLUS (TIMES X (NODEKNT T1))
                  Y))
```

```
    by use of: (LISPLINK REQUAL)

  M1:(EQUAL (PLUS (C.FRINGE T1)
                  (PLUS (MINUS (TIMES X (NODEKNT T1)))
                        (MINUS Y)))
            0)

  01:(EQUAL (PLUS (C.FRINGE T1)
                  (PLUS (MINUS (TIMES X (NODEKNT T1)))
                        (MINUS Y)))
            0)

The result of recursive simplification is:

(ALL T1 (EQUAL (PLUS (C.FRINGE T1)
                     (PLUS (MINUS (TIMES X (NODEKNT T1)))
                           (MINUS Y)))
               0))

(3)Induction is now tried giving a new expression to simplify:

(AND
  (ALL T1 (IMPLIES (NOT (LISTP T1))
                   (EQUAL (PLUS (C.FRINGE T1)
                                (PLUS (MINUS (TIMES X (NODEKNT T1)))
                                      (MINUS Y)))
                          0)))
  (ALL T1
    (IMPLIES
      (AND (LISTP T1)
           (AND (EQUAL (PLUS (C.FRINGE (CAR T1))
                             (PLUS (MINUS (TIMES X (NODEKNT (CAR T1))))
                                   (MINUS Y)))
                       0)
                (EQUAL (PLUS (C.FRINGE (CDR T1))
                             (PLUS (MINUS (TIMES X (NODEKNT (CDR T1))))
                                   (MINUS Y)))
                       0)))
      (EQUAL (PLUS (C.FRINGE T1)
                   (PLUS (MINUS (TIMES X (NODEKNT T1)))
                         (MINUS Y)))
             0))))

The Base Case of the Induction is Evaluated

  I1:(IMPLIES (IF (LISTP T1)
                  NIL T)
              (EQUAL (PLUS (C.FRINGE T1)
                           (PLUS (MINUS (TIMES X (NODEKNT T1)))
                                 (MINUS Y)))
                     0))
    by use of: IMPLIES
  M1:(IF (IF (LISTP T1)
             NIL T)
         (EQUAL (PLUS (C.FRINGE T1)
                      (PLUS (MINUS (TIMES X (NODEKNT T1)))
                            (MINUS Y)))
                0)
         T)

(4)C.FRINGE becomes (A0008) in the Base Case
```

```
      I3:(C.FRINGE T1)
         by use of: C.FRINGE
      M3:(IF (LISTP T1)
             (PLUS (A0007)
                   (PLUS (C.FRINGE (CAR T1))
                         (C.FRINGE (CDR T1))))
             (A0008))
      03:(A0008)

      I3:(NODEKNT T1)
         by use of: NODEKNT
      M3:(IF (LISTP T1)
             (PLUS 1 (PLUS (NODEKNT (CAR T1))
                           (NODEKNT (CDR T1))))
             1)
      03:1

The Base Case evaluated

  01:(IF (LISTP T1)
         T
         (EQUAL (PLUS (A0008)
                      (PLUS (MINUS X)
                            (MINUS Y)))
                0))

(5)The quantifier is eliminated on the Base Case

  I1:(ALL T1 (IF (LISTP T1)
                 T
                 (EQUAL (PLUS (A0008)
                              (PLUS (MINUS X)
                                    (MINUS Y)))
                        0)))
     by use of: (LISPLINK SYMALL)
  M1:(IF (EQUAL (PLUS (A0008)
                      (PLUS (MINUS X)
                            (MINUS Y)))
                0)
         T
         (ALL T1 (IF (LISTP T1)
                     T NIL)))
  01:(EQUAL (PLUS (A0008)
                  (PLUS (MINUS X)
                        (MINUS Y)))
            0)

(6)The Remaining problem after evaluating the Base

  I1:(AND
    (EQUAL (PLUS (A0008)
                 (PLUS (MINUS X)
                       (MINUS Y)))
           0)
    (ALL
      T1
      (IMPLIES
        (AND
          (LISTP T1)
          (AND (EQUAL (PLUS (C.FRINGE (CAR T1))
                            (PLUS (MINUS (TIMES X (NODEKNT (CAR T1))))
                                  (MINUS Y)))
```

```
                    0)
              (EQUAL (PLUS (C.FRINGE (CDR T1))
                           (PLUS (MINUS (TIMES X (NODEKNT (CDR T1))))
                                 (MINUS Y)))
                     0)))
        (EQUAL (PLUS (C.FRINGE T1)
                     (PLUS (MINUS (TIMES X (NODEKNT T1)))
                           (MINUS Y)))
               0))))

Evaluating the Induction Step
    I2:(IMPLIES
      (IF (LISTP T1)
          (IF (EQUAL (PLUS (C.FRINGE (CAR T1))
                           (PLUS (MINUS (TIMES X (NODEKNT (CAR T1))))
                                 (MINUS Y)))
                     0)
              (EQUAL (PLUS (C.FRINGE (CDR T1))
                           (PLUS (MINUS (TIMES X (NODEKNT (CDR T1))))
                                 (MINUS Y)))
                     0)
              NIL)
          NIL)
       (EQUAL (PLUS (C.FRINGE T1)
                    (PLUS (MINUS (TIMES X (NODEKNT T1)))
                          (MINUS Y)))
              0))
        by use of: IMPLIES

C.FRINGE includes (A0007) on the Induction Step
        I4:(C.FRINGE T1)
           by use of: C.FRINGE
        M4:(IF (LISTP T1)
               (PLUS (A0007)
                     (PLUS (C.FRINGE (CAR T1))
                           (C.FRINGE (CDR T1))))
               (A0008))
        04:(PLUS (A0007)
                 (PLUS (C.FRINGE (CAR T1))
                       (C.FRINGE (CDR T1))))

        I4:(NODEKNT T1)
           by use of: NODEKNT
        M4:(IF (LISTP T1)
               (PLUS 1 (PLUS (NODEKNT (CAR T1))
                             (NODEKNT (CDR T1))))
               1)
        04:(PLUS 1 (PLUS (NODEKNT (CAR T1))
                         (NODEKNT (CDR T1))))

(7)The hypothesis is solved for (C.FRINGE(CDR T1))
and substituted into the conclusion.

        M4:(IF
          (EQUAL (PLUS (C.FRINGE (CDR T1))
                       (PLUS (MINUS (TIMES X (NODEKNT (CDR T1))))
                             (MINUS Y)))
                 0)
          (EQUAL
            (PLUS
              (A0007)
              (PLUS
```

```
                (C.FRINGE (CAR T1))
                (PLUS
                  (PLUS Y (TIMES X (NODEKNT (CDR T1))))
                  (PLUS (MINUS X)
                        (PLUS (MINUS (TIMES X (NODEKNT (CAR T1))))
                              (PLUS (MINUS (TIMES X (NODEKNT (CDR T1))))
                                    (MINUS Y)))))))
              0)
            T)
```

which is then simplified

```
        04:(IF
          (EQUAL (PLUS (C.FRINGE (CDR T1))
                       (PLUS (MINUS (TIMES X (NODEKNT (CDR T1))))
                             (MINUS Y)))
                 0)
          (EQUAL (PLUS (A0007)
                       (PLUS (C.FRINGE (CAR T1))
                             (PLUS (MINUS X)
                                   (MINUS (TIMES X (NODEKNT (CAR T1)))))))
                 0)
          T)
```

The Hypothesis is solved for (C.FRINGE(CAR T1))
and then substituted into the conclusion

```
        M4:(IF
          (EQUAL (PLUS (C.FRINGE (CAR T1))
                       (PLUS (MINUS (TIMES X (NODEKNT (CAR T1))))
                             (MINUS Y)))
                 0)
          (IF
            (EQUAL (PLUS (C.FRINGE (CDR T1))
                         (PLUS (MINUS (TIMES X (NODEKNT (CDR T1))))
                               (MINUS Y)))
                   0)
            (EQUAL (PLUS (A0007)
                         (PLUS (PLUS Y (TIMES X (NODEKNT (CAR T1))))
                               (PLUS (MINUS X)
                                     (MINUS (TIMES X (NODEKNT
                                                      (CAR T1)))))))
                   0)
            T)
          T)
```

which is then simplified

```
        04:(IF
          (EQUAL (PLUS (C.FRINGE (CAR T1))
                       (PLUS (MINUS (TIMES X (NODEKNT (CAR T1))))
                             (MINUS Y)))
                 0)
          (IF (EQUAL (PLUS (C.FRINGE (CDR T1))
                           (PLUS (MINUS (TIMES X (NODEKNT (CDR T1))))
                                 (MINUS Y)))
                     0)
              (EQUAL (PLUS (A0007)
                           (PLUS Y (MINUS X)))
                     0)
              T)
          T)
```

The Result of Evaluating the Induction Step

```
    02:(IF
      (LISTP T1)
      (IF (EQUAL (PLUS (C.FRINGE (CAR T1))
                       (PLUS (MINUS (TIMES X (NODEKNT (CAR T1))))
                             (MINUS Y)))
                 0)
          (IF (EQUAL (PLUS (C.FRINGE (CDR T1))
                           (PLUS (MINUS (TIMES X (NODEKNT (CDR T1))))
                                 (MINUS Y)))
                     0)
              (EQUAL (PLUS (A0007)
                           (PLUS Y (MINUS X)))
                     0)
              T)
          T)
      T)
```

(8)The (ALL T1) quantifier is reduced in scope

```
    I2:(ALL
      T1
      (IF
        (LISTP T1)
        (IF (EQUAL (PLUS (C.FRINGE (CAR T1))
                         (PLUS (MINUS (TIMES X (NODEKNT (CAR T1))))
                               (MINUS Y)))
                   0)
            (IF (EQUAL (PLUS (C.FRINGE (CDR T1))
                             (PLUS (MINUS (TIMES X (NODEKNT (CDR T1))))
                                   (MINUS Y)))
                       0)
                (EQUAL (PLUS (A0007)
                             (PLUS Y (MINUS X)))
                       0)
                T)
            T)
        T))
       by use of: (LISPLINK SYMALL)
```

resulting in

```
    M2:(IF
      (EQUAL (PLUS (A0007)
                   (PLUS Y (MINUS X)))
             0)
      T
      (ALL
        T1
        (IF
          (LISTP T1)
          (IF
            (EQUAL (PLUS (C.FRINGE (CAR T1))
                         (PLUS (MINUS (TIMES X (NODEKNT (CAR T1))))
                               (MINUS Y)))
                   0)
            (IF (EQUAL (PLUS (C.FRINGE (CDR T1))
                             (PLUS (MINUS (TIMES X (NODEKNT (CDR T1))))
                                   (MINUS Y)))
                       0)
                NIL T)
            T)
          T)))
```

```
The Quantified sub expression is examined
      I3:(ALL
        T1
        (IF
          (LISTP T1)
          (IF
            (EQUAL (PLUS (C.FRINGE (CAR T1))
                         (PLUS (MINUS (TIMES X (NODEKNT (CAR T1))))
                               (MINUS Y)))
                   0)
            (IF (EQUAL (PLUS (C.FRINGE (CDR T1))
                             (PLUS (MINUS (TIMES X (NODEKNT (CDR T1))))
                                   (MINUS Y)))
                       0)
                NIL T)
            T)
          T))
         by use of: (LISPLINK SYMALL)

T1 is replaced by (CONS *1 *2)
      M3:
        (ALL
          *1
          (ALL
            *2
            (IF
              (LISTP (CONS *1 *2))
              (IF
                (EQUAL
                  (PLUS (C.FRINGE (CAR (CONS *1 *2)))
                        (PLUS (MINUS (TIMES X
                                            (NODEKNT (CAR (CONS *1 *2)))))
                              (MINUS Y)))
                  0)
                (IF
                  (EQUAL
                    (PLUS (C.FRINGE (CDR (CONS *1 *2)))
                          (PLUS (MINUS (TIMES X
                                              (NODEKNT
                                                (CDR (CONS *1 *2)))))
                                (MINUS Y)))
                    0)
                  NIL T)
                T)
              T)))

(9)Resulting in 03 below because
         I5:(ALL *2
                 (IF (EQUAL (PLUS (C.FRINGE *2)
                                  (PLUS (MINUS (TIMES X (NODEKNT *2)))
                                        (MINUS Y)))
                            0)
                     NIL T))
              by use of: EX
         05:NIL

       I4:(ALL *1 (IF (EQUAL (PLUS (C.FRINGE *1)
                                   (PLUS (MINUS (TIMES X (NODEKNT *1)))
                                         (MINUS Y)))
                             0)
                      NIL T))
          by use of: EX
```

```
        04:NIL
    03:NIL

The Result of the Induction Step
    02:(EQUAL (PLUS (A0007)
                    (PLUS Y (MINUS X)))
              0)

The result of recursive simplification is:
(IF (EQUAL (PLUS (A0008)
                 (PLUS (MINUS X)
                       (MINUS Y)))
           0)
    (EQUAL (PLUS (A0007)
                 (PLUS Y (MINUS X)))
           0)
    NIL)
end of deduction
63798 conses
174.162 seconds
11.479 seconds, garbage collection time

EX1=0/EX2=0/EX3=0/EX4=0/EX5=0/EX6=0/EX7=0/EX8=0/EX9=0/EX10=0/
ALL1=0/ALL2=0/ALL3=0/ALL4=1/ALL5=1/ALL6=0/ALL7=2/ALL8=2/ALL9=1/ALL10=1/
```

11. CONCLUSION

We have constructed an entire automatic deduction and induction system based on a single principle which we call The Fundamental Deduction Principle. Unlike most other automatic deduction systems, this system does no unification whatsoever. Instead, it is based on the SYMMETRIC LOGIC technique of reducing the scope of quantifiers. We have used this sytem both as a programming language interpreter to make deductions in natural language theory and Ontology and as a more general deduction system to prove theorems and analysize the complexity of LISP functions. This research has not been done in a vacume. Indeed, Bledsoe[Bledsoe2] argued over a decade ago that automatic deduction systems must INCLUDE ideas akin to our Fundamental Deduction Principle as opposed to the then prevailing Resolution viewpoint of deduction as being a problem of exploring a search space. However, the thesis of this research is incredibly stronger, for we are argueing that for many interesting theories in mathematics and computer science, that the Fundamental Deduction Principle is the ONLY idea that needs to be included. We were lead to this principle partly by trial and error in constructing deduction systems and suffering the effects of redundand expessions whenever we departed from this principle, and partly by Meltzer's contention that induction must in some way be related to deduction[Meltzer]. The research of Boyer and Moore[Boyer] has also influenced us, in fact we have applied their techniques for Noetherian Induction in quantifier free logic to our Quantified logic. This research has also been influenced by the idea that computation is a very special case of deduction, and that a deductive system must be capable of computation. This is related to Kowalski's[Kowalski] thesis that deducive systems are capable of computation. For logical languages based on quantifier free functional repersentation, such as pure LISP it is easy to achieve computational ability while obeying the fundamental deduction principal. However, for logical languages which include quantifiers and relational notation, achieving computational ability while obeying this principal would seem to be rather difficult since neither resolution nor unification generally obey this principal. Never-the-less we have succeeded in constructing a new deductive method called the SYMMETRIC LOGIC which has significant computational ability and which satisfies the fundamental deduction principal. This deductive method was first degugged in collaboration with Schwind[Brown14,15,17] by hand simulation of part of her theory[Schwind] for translating natural language into logic.

I wish to thank my students who have worked on this project, particularly Nelson Bishop who tested much of Schwind's grammer and Song Park who worked on the set theory examples.

12. REFERENCES

Bibel2 W. and Schreiber J. "Proof Search in a Gentzen like system of first order logic", INTERNATIONAL COMPUTING SYMPOSIUM 1975, North Holland, Amsterdam.

Bledsoe1 W.W. "Splitting and Reduction Heuristics in Automatic Theorem Proving" ARTIFICIAL INTELLIGENCE vol. 2, no.1 Spring 1971.

Bledsoe2 W.W. "Non Resolution Theorem Proving" ARTIFICIAL INTELLIGENCE Vol. 9, 1977.

Boyer1,Robert S. and Moore J Strother, A COMPUTATIONAL LOGIC, Academic Press 1979.

Brown1,F.M., "A Deductive System for Elementary Arithmetic", 2nd AISB CONFERENCE PROCEEDINGS, Edinburgh, July 1976.

Brown3,F.M., "Doing Arithmetic Without Diagrams", ARTIFICIAL INTELLIGENCE, Vol. 8, Spring 1977.

Brown4,F.M., "A Theorem Prover for Elementary Set Theory", 5th INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, MIT, August 1977. Also the abstract is in the WORKSHOP ON AUTOMATIC DEDUCTION COLLECTED ABSTRACTS, MIT, August 1977.

Brown5,F.M., and Sten-Ake Tarnlund, "Inductive Reasoning in Mathematics", 5th INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, MIT, August 1977.

Brown6,F.M., "Towards the Automation of Set Theory and its Logic", ARTIFICIAL INTELLIGENCE, Vol. 10, 1978.

Brown10,F. M., and Sten-Ake Tarnlund, "Inductive Reasoning on Recursive Equations" ARTIFICIAL INTELLIGENCE, Vol. 12 #3, November 1979.

Brown11,F. M., "An investigation into the goals of research in Automatic Theorem Proving as related to Mathematical Reasoning" Artificial Intelligence, 1980.

Brown12,F.M., "A Sequent Calculus for Modal Quantificational Logic", 3rd AISB/GI CONFERENCE PROCEEDINGS, Hamburg, July 1978.

Brown13,F.M., and C.B. Schwind, "Analyzing and Representing Natural Language in Logic," 3rd AISB/GI CONFERENCE PROCEEDINGS, Hamburg, July 1978.

Brown14,F.M., and C.B. Schwind, "Towards an Integrated Theory of Natural Language Understanding," To appear on microfilch in Linguistics Journal. Also extended abstract in 4th COLING CONFERENCE PROCEEDINGS, Bergen, 1978.

Brown15,F.M., and C.B. Schwind, "Outline of an Integrated Theory of Natural Language Understanding", REPRESENTATION AND PROCESSING OF NATURAL LANGUAGE, ed. Leonard Bolc, Carl Hanser Verlag 1980.

Brown17,F.M., "An Automatic Proof of the Completeness of Quantificational Logic," Department of Artificial Intelligence Research Report 52, 1978.

Brown18,F.M., "A Theorem Prover for Metatheory", 4th CONFERENCE ON AUTOMATIC THEOREM PROVING, Austin Texas,1979.

Brown20,F.M. "Logic Algorithms for Natural Numbers" TR 108,1979.

Brown21,F.M. "A Deductive System for Meta Theoretic Reasoning", TR 132, Jan 1980.

Brown24,F.M. "A Deductive System for Real Algebra" TR 141, March 1980.

Brown25,F.M. "A Sequent Calculus for Intensional Logic" TR 143 April 1980.

Brown26,F.M. "Computation with Automatic Theorem Provers", to appear in the proceedings of the NSF workshop on Logic Programming, Los Angeles, 1981.

Chester1,Daniel "HCPRVR: An Intrepreter for Logic Programs" procedings of THE FIRST ANNUAL NATIONAL CONFERENCE on ARTIFICIAL INTELLIGENCE The American Association for Artificial Intellengence. 1980.

Colmerauer A.,Kanoui H.,Van Caneghem M. "Last Steps Towards an Ultimate Prolog" PROCEEDINGS OF THE SEVENTH JOINT INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, pages947-948, University of British Columbia ,1981.

Frege Gottlob, "Begfiffschrift, a formula language, modeled upon that of arithmetic, for pure thought" 1879, in FROM FREGE TO GODEL, Harvard Univ. Press 1967.

Henry,D.P., MEDIEVAL LOGIC AND METAPHYSICS, Hutchinson & CO LTD, 1972.

Kowalski, Robert "Predicate Logic as programming language", PROCEEDINGS IFIP, 1974.

Luschei,E. C., THE LOGICAL SYSTEMS OF LESNIEWSKI, 1962.

McCarthy, J. et. al. LISP 1.5 Programmer's Manual, MIT Press, 1966.

Meltzer, B. "The Programming of Deduction and Induction" DCL memo 45. Department of Computational Logic, The University of Edinburgh, 1971.

Moore,J "Computational Logic: Structure Sharing and Proof of Program Properties" Phd thesis, University of Edinburgh, 1973.

Pastre D. DEMONSTRATION AUTOMATIQUE DE THEOREMS EN THEORIE DES ENSEMBLES These de 3eme cycle, Paris VI, 1976.

Prawitz D. "An Improved Proof Procedure" THEORIA 26, 1960.

Robinson,J.A. "A machine oriented logic based on the resolution principle" J.ACM 12, 1965.

Schwind,C.B. EIN FORMALISMUS ZUR BESCHREIBUNG DER SYNTAX UND BEDEUTUNG VON FRAGE-ANTWORT-SYSTEMEN, Ph.D. thesis, Technischen Universitat, Munchen. 1977.

Simmons1 R.F. and Chester D.L. "Relating Sentences and Semantic Networks with Procedural Logic", CACM (to appear) 1979.

Simmons2 R.F. "A Narrative Schema in Procedural Logic" in Clark K. and Tarnlund S. (eds.) LOGIC PROGRAMMING, (in press) Academic Press, New York, 1982.

Szabo,M.E. ed. THE COLLECTED PAPERS OF GERHARD GENTZEN, North Holland, Amsterdam, 1969.

Wang2 Hao "On the long-range prospects of automatic theorem-proving", LECTURE NOTES IN MATHEMATICS: SYMPOSIUM ON AUTOMATIC DEMONSTRATION, held at Versailles/France 1968, Springer-Verlag 1970.