# AUTOMATIC ANALYSIS OF THE COMPLEXITY OF RECURSIVE FUNCTIONS

Dr. Frank M. Brown

TR-83-15     August 1983

# AUTOMATIC ANALYSIS OF THE COMPLEXITY OF RECURSIVE FUNCTIONS

Dr. Frank M. Brown
Dept. Computer Sciences
The University of Texas at Austin
Austin, Texas 78712
CS.BROWN@UTEXAS-20

## 1 INTRODUCTION

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[Brown50] 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.

## 2 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.

```
          (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

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:

```
                       *
                      / \
                     /   \
                    *     C
                   / \
                  /   \
                 A     B
```

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)
```

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.

## 3 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)

 O1:(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))
O3:(A0008)
```

```
I3:(NODEKNT T1)
   by use of: NODEKNT
M3:(IF (LISTP T1)
       (PLUS 1 (PLUS (NODEKNT (CAR T1))
                     (NODEKNT (CDR T1))))
       1)
O3:1
```

The Base Case evaluated

```
O1:(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)))
O1:(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))
        O4:(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)
        O4:(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/
```

## 4 CONCLUSION

A reasoning system: ANALYZE capable of analysing the complexity of some recursive LISP functions has been developed. This system uses a sophisticated automatic deduction system: SYMEVAL as its basic reasoning mechanism. A description of SYMEVAL is given in[Brown50].

## 5 REFERENCES

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

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

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

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

Brown50,F.M.. "Experimental Logic and the Automatic Analysis of Algorithms" THE ARMY CONFERENCE ON APPLICATION OF ARTIFICIAL INTELLIGENCE TO BATTLEFIELD INFORMATION MANAGEMENT, April 20,1983 - April 23,1983. To be published in Spring 1984.