Artificial Intelligence Project---M.I.T. and M.I.T. Computation Center Symbol Manipulating Language--Memo 14--The Wang Algorithm for the Propositional Calculus Programmed in LISP

by

John McCarthy

This memorandum describes a LISP program for deciding whether an expression in the propositional calculus is a tautology according to Wang's algorithm. Wang's algorithm is an excellent example of the kind of algorithm which is conveniently programmed in LISP, and the main purpose of this memorandum is to help would-be users of LISP see how to use it.

1. The Wang algorithm. We quote from pages 5 and 6 of Wang's paper.

"The propositional calculus (System P)

Since we are concerned with practical feasibility, it is preferable to use more logical connectives to begin with when we wish actually to apply the procedure to concrete cases. For this purpose we use the five usual logical constants ~ (not), & (conjunction), ∨ (disjunction), ⊃ (implication), ≡ (bi-conditionals), with their usual interpretations.

A propositional letter P, Q, R, M or N, et cetera, is a formula (and an "atomic formula"). If φ, ψ are formulae, then ~φ, φ & ψ, φ ∨ ψ, φ ⊃ ψ, φ ≡ ψ are formulae. If π, ρ are strings of formulae (each, in particular, might be an empty string or a single formula) and φ is a formula, then π, φ, ρ is a string and π → φ is a sequent which, intuitively speaking, is true if and only if either some formula in the string π (the "antecedent") is false or some formula in the string ρ (the "consequent") is true, i.e., the conjunction of all formulae in the antecedent implies the disjunction of all formulae in the consequent.

There are eleven rules of derivation. An initial rule states that a sequent with only atomic formulae (proposition letters) is a theorem if and only if a same formula occurs on
both sides of the arrow. There are two rules for each of the 
five truth functions—one introducing it into the antecedent, 
one introducing it into the consequent. One need only reflect 
on the intuitive meaning of the truth functions and the arrow 
sign to be convinced that these rules are indeed correct. Lat- 
ter on, a proof will be given of their completeness, i.e., all 
intuitively valid sequents are provable, and of their consist- 
ency, i.e., all provable sequents are intuitively valid.

Pl. Initial rule: if $\lambda, \zeta$ are strings of atomic formulae, then 
$\lambda \rightarrow \zeta$ is a theorem if some atomic formula occurs on both 
sides of the arrow.

In the ten rules listed below, $\lambda$ and $\zeta$ are always strings 
(possibly empty) of atomic formulae. As a proof procedure in 
the usual sense, each proof begins with a finite set of cases 
of Pl and continues with successive consequences obtained by 
the other rules. As will be explained below, a proof looks 
like a tree structure growing in the wrong direction. We shall, 
however, be chiefly interested in doing the step backwards, 
thereby incorporating the process of searching for a proof.

The rules are so designed that given any sequent, we can 
find the first logical connective, i.e., the leftmost symbol 
in the whole sequent that is a connective, and apply the ap- 
propriate rule to eliminate it, thereby resulting in one or 
two premises which, taken together, are equivalent to the con- 
clusion. This process can be repeated until we reach a finite 
set of sequents with atomic formulae only. Each connective-
free sequent can then be tested for being a theorem or not, by 
the initial rule. If all of them are theorems, then the or- 
iginal sequent is a theorem and we obtain a proof; otherwise 
we get a counterexample and a disproof. Some simple samples 
will make this clear.
For example, given any theorem of "Principia", we can automatically prefix an arrow to it and apply the rules to look for a proof. When the main connective is \( \vdash \), it is simpler, though not necessary, to replace the main connective by an arrow and proceed. For example:

*2.45. \( \vdash : \neg (P \lor Q) \rightarrow \neg \neg P \).

*5.21. \( \vdash : \neg P \land \neg Q \rightarrow P \equiv Q \)

can be rewritten and proved as follows.

*2.45  \( \neg (P \lor Q) \rightarrow \neg P \)  
1. \( \neg P, P \lor Q \)   
2. \( P \rightarrow P \lor Q \)   
3. \( P \rightarrow P, Q \)   
VALID  

*5.21  \( \rightarrow \neg P \land \neg Q \rightarrow P \equiv Q \)  
1. \( \neg P \land \neg Q ightarrow P \equiv Q \)   
2. \( \neg P, \neg Q \rightarrow P \equiv Q \)   
3. \( \neg Q \rightarrow P \equiv Q, P \)   
4. \( \neg P \rightarrow \neg Q, P \)   
5. \( P \rightarrow Q, P, Q \)   
VALID  

(5) \( Q \rightarrow P, P, Q \)  
VALID

---

P2a. Rule \( \rightarrow \neg \): If \( \phi, \xi \rightarrow \lambda, \psi, \rho \), then \( \xi \rightarrow \lambda, \neg \phi, \rho \).

P2b. Rule \( \rightarrow \neg \): If \( \neg \lambda, \rho \rightarrow \tau, \phi \), then \( \neg \lambda, \neg \phi, \rho \rightarrow \tau \).

P3a. Rule \( \rightarrow \& \): If \( \xi \rightarrow \lambda, \phi, \psi, \rho \) and \( \xi \rightarrow \lambda, \psi, \rho \), then \( \xi \rightarrow \lambda, \phi \land \psi, \rho \).

P3b. Rule \( \rightarrow \& \): If \( \lambda, \phi, \psi, \rho \rightarrow \tau \) and \( \lambda, \psi, \rho \rightarrow \tau \), then \( \lambda, \phi \land \psi, \rho \rightarrow \tau \).

P4a. Rule \( \rightarrow \lor \): If \( \xi \rightarrow \lambda, \phi, \psi, \rho \) and \( \xi \rightarrow \lambda, \phi \lor \psi, \rho \).

P4b. Rule \( \rightarrow \lor \): If \( \lambda, \phi, \psi, \rho \rightarrow \tau \) and \( \lambda, \psi, \rho \rightarrow \tau \), then \( \lambda, \phi \lor \psi, \rho \rightarrow \tau \).

P5a. Rule \( \rightarrow \lor \): If \( \xi, \phi \rightarrow \lambda, \psi, \rho \) and \( \xi \rightarrow \lambda, \phi \lor \psi, \rho \).

P5b. Rule \( \rightarrow \lor \): If \( \lambda, \psi, \rho \rightarrow \tau \) and \( \lambda, \rho \rightarrow \tau \), then \( \lambda, \phi \lor \psi, \rho \rightarrow \tau \).

P6a. Rule \( \equiv \): If \( \phi, \xi \rightarrow \lambda, \psi, \rho \) and \( \xi \rightarrow \lambda, \phi \equiv \psi, \rho \).

P6b. Rule \( \equiv \): If \( \phi, \psi, \lambda, \rho \rightarrow \tau \) and \( \lambda, \rho \rightarrow \tau \), then \( \lambda, \phi \equiv \psi, \rho \rightarrow \tau \).
2. The LISP program. We define a function theorem \([s]\)
whose value is truth or falsity according to whether the sequent
\(s\) is theorem.

The sequent
\[s: \varphi_1, \ldots, \varphi_n \rightarrow \psi_1, \ldots, \psi_m\]
is represented by the S-expression
\[s^1: \text{(ARROW,} (\varphi_1^1, \ldots, \varphi_n^1), (\psi_1^1, \ldots, \psi_m^1))\]
where in each case the ellipsis \(\ldots\) denotes missing terms.

Propositional formulae are represented as follows:

1. For "atomic formulae" (Wang's terminology) we use
   "atomic symbols" (LISP terminology).

2. The following table gives our "Cambridge Polish" way
   of representing propositional formulae with given main connectives:

   - 1. \(\sim \varphi\) becomes \((\text{NOT}, \varphi^1)\)
   - 2. \(\varphi \land \psi\) becomes \((\text{AND}, \varphi^1, \psi^1)\)
   - 3. \(\varphi \lor \psi\) becomes \((\text{OR}, \varphi^1, \psi^1)\)
   - 4. \(\varphi \Rightarrow \psi\) becomes \((\text{IMPLIES}, \varphi^1, \psi^1)\)
   - 5. \(\varphi \equiv \psi\) becomes \((\text{EQUIV}, \varphi^1, \psi^1)\)

Thus the sequent
\[\sim P \land \sim Q \rightarrow P \equiv Q, R \lor S\]
is represented by
\[(\text{ARROW,} ((\text{AND,} (\text{NOT}, P), (\text{NOT}, Q))), ((\text{EQUIV}, P, Q), (\text{OR}, R, S))))\]

The S-function theorem \([s]\) is given in terms of auxiliary functions as follows:

\[\text{theorem } [s] = \text{th1[NIL;NIL;\text{cadtr}[s];\text{cadadr}[s]]}\]
\[\text{th1[a1;a2;a;c] = [\text{null[a]} \rightarrow \text{th2[a1;a2;NIL;NIL;c]}; T \rightarrow}\]
\[\text{member[car[a;c]]; \text{atom[car[a]]}} \rightarrow \text{th1[member[car[a];a1] \rightarrow a1; T \rightarrow cons[car[a];a1]; a2 cdt[a;c]; T \rightarrow th1[a1; member[car[a];a2] \rightarrow a2;}}
\[\text{T \rightarrow cons[car[a];a2]; cdr[a;c]]}]\]
\text{th2}[^a_1; a_2; c_i; c_2; c] = [\text{null}[^e] \rightarrow \text{th}[^a_1; a_2; c_i; c_2];
\text{atom}[^{\text{car}}[c]] \rightarrow \text{th2}[^a_1; a_2; \text{member}[^{\text{car}}[c]; c_i] \rightarrow c_1; T \rightarrow \\
\text{cons}[^{\text{car}}[c]; c_i]; c_2; \text{cdr}[^{\text{a}_1}; c_2]; T \rightarrow \text{th2}[^a_1; a_2; c_i; c_2]]
\text{member}[^{\text{car}}[c]; c_2]; c_2; T \rightarrow \\
\text{cons}[^{\text{car}}[c]; c_2]; \text{cdr}[c_i]]
\text{th}[^a_1; a_2; c_i; c_2] = [\text{null}[^a_2] \rightarrow \sim \text{null}[^c_2] \land \text{th}[^c_2; a_1; a_2; c_i; \text{cdr}[c_2]]; T \rightarrow \text{th}[^\text{car}[a_2]; a_1; \\
\text{cdr}[a_2]; c_1; c_2]]

\text{th} \text{ is the main predicate through which all the recursions take place. theorem, th1 and th2 break up and sort the information in the sequent for the benefit of th. The four arguments of th are:}
\begin{itemize}
  \item \text{a1: atomic formulae on left side of arrow}
  \item \text{a2: other formulae on left side of arrow}
  \item \text{c1: atomic formulae on right side of arrow}
  \item \text{c2: other formulae on right side of arrow}
\end{itemize}

The atomic formulae are kept separate from the others in order to make faster the detection of the occurrence of formula on both sides of the arrow and the finding of the next formula to reduce. Each use of \text{th} represents one reduction according to one of the 10 rules. The formula to be reduced is chosen from the left side of the arrow if possible. According to whether the formula to be reduced is on the left or right we use \text{th1} or \text{th2}. We have
\[
\text{th1}[^u; a_1; a_2; c_i; c_2] = [
  \begin{array}{l}
    \text{car}[^u] = \text{NOT} \rightarrow \text{th1}[^{\text{cadr}}[u]; a_1; a_2; c_i; c_2] \\
    \text{car}[^u] = \text{AND} \rightarrow \text{th2}[^{\text{cdr}}[u]; a_1; a_2; c_i; c_2] \\
    \text{car}[^u] = \text{OR} \rightarrow \text{th1}[^{\text{cadr}}[u]; a_1; a_2; c_i; c_2] \land \text{th1}[^{\text{cadr}}[u]; a_1; a_2; c_i; c_2] \\
    \text{car}[^u] = \text{IMPLIES} \rightarrow \text{th1}[^{\text{cadr}}[u]; a_1; a_2; c_i; c_2] \land \text{th1}[^{\text{cadr}}[u]; a_1; a_2; c_i; c_2] \\
    \text{car}[^u] = \text{EQUIV} \rightarrow \text{th2}[^{\text{cdr}}[u]; a_1; a_2; c_i; c_2] \land \text{th2}[^{\text{cdr}}[u]; a_1; a_2; c_i; c_2] \\
  \end{array}
\]
\[
\text{thr}[u; a_1; a_2; c_1; c_2] = [\]
\text{car}[u] \rightarrow \text{th1l}[\text{cadr}[u]; a_1; a_2; c_1; c_2] \\
\text{car}[u] \rightarrow \text{thir}[\text{cadr}[u]; a_1; a_2; c_1; c_2] \\
\text{caddr}[u]; a_1; a_2; c_1; c_2 \wedge \text{th1l}[\text{cadr}[u]; a_1; a_2; c_1; c_2] \\
\text{caddr}[u]; caddr[u]; a_1; a_2; c_1; c_2] \\
\text{caddr}[u]; a_1; a_2; c_1; c_2] \\
\text{OR} \rightarrow \text{th2r}[\text{cadr}[u]; a_1; a_2; c_1; c_2] \\
\text{IMPLIES} \rightarrow \text{th1l}[\text{cadr}[u]; caddr[u]; a_1; a_2; c_1; c_2] \\
\text{EQUIV} \rightarrow \text{th1l}[\text{cadr}[u]; caddr[u]; a_1; a_2; c_1; c_2] \wedge \text{th1l}[\text{cadr}[u]; caddr[u]; a_1; a_2; c_1; c_2]
\]

The functions \text{th1l}, \text{thir}, \text{th2r}, \text{th2l} distribute the parts of the reduced formula to the appropriate places in the reduced sequent.

These functions are

\[
\text{th1l}[v; a_1; a_2; c_1; c_2] = [\text{atom}[v] \rightarrow \text{member}[v; c_1] \vee \\
\text{th}[\text{cons}[v; a_1]; a_2; c_1; c_2]; T \rightarrow \text{member}[v; c_2] \vee \\
\text{th}[a_1; \text{cons}[v; a_2]; c_1; c_2]]
\]

\[
\text{thir}[v; a_1; a_2; c_1; c_2] = [\text{atom}[v] \rightarrow \text{member}[v; a_1] \vee \\
\text{th}[a_1; a_2; \text{cons}[v; c_1]; c_2]; T \rightarrow \text{member}[v; a_2] \vee \\
\text{th}[a_1; a_2; c_1; \text{cons}[v; a_2]]]
\]

\[
\text{th2l}[v; a_1; a_2; c_1; c_2] = [\text{atom}[\text{car}[v]] \rightarrow \text{member}[\text{car}[v]; c_1] \vee \\
\text{th1l}[\text{cadr}[v]; \text{cons}[\text{car}[v]; a_1]; a_2; c_1; c_2]; T \rightarrow \text{member}[\text{car}[v]; c_2] \vee \\
\text{th1l}[\text{cadr}[v]; a_1; \text{cons}[\text{car}[v]; a_2]; c_1; c_2]]
\]

\[
\text{th2r}[v; a_1; a_2; c_1; c_2] = [\text{atom}[\text{car}[v]] \rightarrow \text{member}[\text{car}[v]; a_1] \vee \\
\text{thir}[\text{cadr}[v]; a_1; a_2; \text{cons}[\text{car}[v]; c_1]; c_2]; T \rightarrow \text{member}[\text{car}[v]; a_2] \vee \\
\text{th}[\text{cadr}[v]; a_1; a_2; c_1; \text{cons}[\text{car}[v]; c_2]]]
\]
th11(v1; v2; a1; a2; c1; c2) = [atom(v1) → member(v1; c1) V
thir(v2; cons(v1; a1; a2; c1; c2); T → member(v1; c2) V
thir(v2; a1; cons(v1; a2; c1; c2); ]

Finally the function \textit{member} is defined by

\texttt{member(x; u) = ~null(u) \land [equal(x; car(u)) V member(x; cdr(u))]}\]

3. The LISP Program as Written in S-expressions. The present section is redundant for those who fully understand LISP. In it we give the translation of the functions of the preceding section into S-expressions.

We have

\texttt{DEFINE ((
(THEOREM (LAMBDA (S) (TH1 NIL NIL, (CADR S) (CADDR S))))

\texttt{(TH (LAMBDA (A1 A2 C1 C2) (COND ((NULL A2) (AND (NOT (NULL C2))
(THR (CAR C2) A1 A2 C1 (CDR C2))) ) (T (THL (CAR A2) A1 (CDR A2)
C1 C2)))))

\texttt{(TH1 (LAMBDA (A1 A2 A C) (COND ((NULL A)
(TH2 A1 A2 NIL NIL C)) (T
(OR (MEMBOB (CAR A) C) (COND ((ATOM (CAR A))
(TH1 (COND ((MEMBOB (CAR A) A1 ) A1)
(T (CONS (CAR A) A1))) A2 (CDR A) C))
(T (TH1 A1 (COND ((MEMBOB (CAR A) A2) A2)
(T (CONS (CAR A) A2))) (CDR A) C))))))

\texttt{(TH2 (LAMBDA (A1 A2 C1 C2 C) (COND
((NULL C) (TH A1 A2 C1 C2))
((ATOM (CAR C)) (TH2 A1 A2 (COND
((MEMBOB (CAR C) C1) C1) (T
(CONS (CAR C) C1))) C2 (CDR C)))
(T (TH2 A1 A2 C1 (COND ((MEMBOB
(CAR C) C2) C2) (T (CONS (CAR C) C2))
(CDR C))))))))
(THL (LAMBDA (U A1 A2 C1 C2) (COND
  ((EQ (CAR U) (QUOTE NOT)) (TH1R (CADR U) A1 A2 C1 C2))
  ((EQ (CAR U) (QUOTE AND)) (TH2L (CDR U) A1 A2 C1 C2))
  ((EQ (CAR U) (QUOTE OR)) (AND (TH1L (CADR U) A1 A2 C1 C2)
    (TH1R (CADR U) A1 A2 C1 C2)))
  ((EQ (CAR U) (QUOTE IMPLIES)) (AND (TH1L (CADR U) A1 A2 C1 C2)
    (TH2L (CDR U) A1 A2 C1 C2)))
  (T (ERROR (LIST (QUOTE THL) U A1 A2 C1 C2))))
))

(THR (LAMBDA (U A1 A2 C1 C2) (COND
  ((EQ (CAR U) (QUOTE NOT)) (TH1L (CADR U) A1 A2 C1 C2))
  ((EQ (CAR U) (QUOTE AND)) (AND (TH1R (CADR U) A1 A2 C1 C2)
    (TH2R (CDR U) A1 A2 C1 C2)))
  ((EQ (CAR U) (QUOTE OR)) (TH2L (CDR U) A1 A2 C1 C2))
  ((EQ (CAR U) (QUOTE IMPLIES)) (TH1L (CADR U) (CADDR U)
    A1 A2 C1 C2))
  ((EQ (CAR U) (QUOTE EQUIV)) (AND (TH1R (CADR U) (CADDR U)
    A1 A2 C1 C2) (TH1L (CADR U) (CADDR U) A1 A2 C1 C2)))
  (T (ERROR (LIST (QUOTE THR) U A1 A2 C1 C2))))
))

(TH1L (LAMBDA (V A1 A2 C1 C2) (COND
  ((ATOM V) (OR (MEMBER V C1)
    (TH (CONS V A1) A2 C1 C2)))
  (T (OR (MEMBER V C2) (TH A1 (CONS V A2) C1 C2))))
))

(TH1R (LAMBDA (V A1 A2 C1 C2) (COND
  ((ATOM V) (OR (MEMBER V A1)
    (TH A1 A2 (CONS V C1) C2)))
  (T (OR (MEMBER V A2) (TH A1 A2 C1 (CONS V C2)))))
))
(TH2L (LAMBDA (V A1 A2 C1 C2) (COND
   ((ATOM (CAR V)) (OR (MEMBER (CAR V) C1)
   (TH1L (CADR V) (CONS (CAR V) A1 ) A2 C1 C2)))
   (T (OR (MEMBER (CAR V) C2) (TH1L (CADR V) A1 (CONS (CAR V)
A2) C1 C2)))))
   )))

(TH2R (LAMBDA (V A1 A2 C1 C2) (COND
   ((ATOM (CAR V)) (OR (MEMBER (CAR V) A1)
   (TH1R (CADR V) A1 A2 (CONS (CAR V) C1) C2 )))
   (T (OR (MEMBER (CAR V) A2) (TH1R (CADR V) A1 A2 C1
   (CONS (CAR V) C2)))))
   )))

(TH1L (LAMBDA (V1 V2 A1 A2 C1 C2) (COND
   ((ATOM V1) (OR (MEMBER V1 C1) (TH1R V2 (CONS V1 A1) A2 C1
C2)))
   (T (OR (MEMBER V1 C2) (TH1R V2 A1 (CONS V1 A2) C1 C2)))
   )))

(MEMBER MEMBOB)
)) ()

This causes the functions mentioned to be defined.
In our test run we next gave

TRACLIS ((TH)) ()

which caused the arguments and values of the function th to be printed each time the function came up in the recursion. Accidentally, it turns out that these arguments essentially constitute a proof in Wang's style of the theorem.
4. Wang's Algorithm and Ambiguous Functions. This section contains a remark on the theory of computation suggested by the experience of programming Wang's algorithm. The functions given in section 2 owe some of their complexity to an effort to gain efficiency and could be made still more efficient at the cost of greater complexity. Namely, in the verbal description of the algorithm in section 1 it is not specified how the term in the sequent to be acted on first is chosen. This choice does not affect the ultimate result, i.e., whether a sequent is found to be a theorem, but does affect the time required.

This suggests that for the purposes of proving theorems about computable functions we introduce the concept of ambiguous function and certain of its properties.

An ambiguous function $f$ assigns to certain $n$-tuplets $x_1, \ldots, x_n$ a value $f(x_1, \ldots, x_n)$. However, this value is not completely determined by the function $f$ and the $n$-tuplet $x_1, \ldots, x_n$ but may be any member of a set $R(f, x_1, \ldots, x_n)$. No mechanism, probabilistic or otherwise, is provided for deciding which element will be chosen. Ambiguous functions may be combined to get new functions by composition, conditional expressions, and recursion.

In order to consider the class $\mathcal{C}(\{\mathcal{F}\})$ of ambiguous functions computable in terms of a class of basic functions $\mathcal{F}$, (This corresponds to $\mathcal{C}(\{\mathcal{F}\}$ in [4]) we introduce the ambiguity operator $\text{amb.}$. $\text{amb.}(x, y)$ is an ambiguous function of $x$ and $y$ whose value may be either $x$ or $y$. Other ambiguities may be
introduced by using amb. For example we can define

$$
\begin{align*}
\text{amblist} & : \{ \text{null}[\ell] \& \text{null}[\text{cdr}[\ell]] \rightarrow \text{car}[\ell]; \\
\text{null}[\ell] & \rightarrow \text{amb}[\text{car}[\ell]; \text{amblist}[\text{cdr}[\ell]]] \\
\end{align*}
$$

However, unless we have a list of all atomic symbols available
we cannot define a function whose value is ambiguous among all
atomic symbols.

We can introduce a transitive relation of descent between
ambiguous functions. We say that $f$ is a descendant of $g$, written

$$f \prec g$$

if for every $x$ every possible value of $f(x)$ is also a possi-
bile value of $g(x)$. The operations used to define new func-
tions in terms of old ones, i.e. composition, conditional ex-
pressions and recursion preserve descent in the following sense:
If a function $h_1$ is defined by a recursive definition in terms
of a function $f_1$ and other functions, and a function $h_2$ is
defined in the very same way but with $f_1$ replaced by $f_2$;
then from $f_1 \prec f_2$ it follows that $h_1 \prec h_2$.

A property $P$ of functions such that if $f$ satisfies $P$ and
g$\prec f$ implies that $g$ satisfies $P$ is called hereditary.

We apply these concepts to the example of Wang's algorithm
as follows:

1. We define a predicate reduced[s] which is true if and
   only if the sequent has only atomic terms.

2. We define dup[s] which is true if the reduced sequent $s$
   has a common term on the right and left.

3. We define reduced[s; term] and reduce2[s; term] which gives
   the two sequents into which the sequent $s$ reduces when it is
   attacked at term (one of these may be trivial).

4. We define terms[s], the list of terms of the sequent.

Wang's algorithm for determining whether a sequent is a theorem
is given by

\[
\text{wang}(s) = [\text{reduced}(s) \rightarrow \text{dup}(s); T \rightarrow \\
\text{wang}([\text{reduced1}(s; \text{amblis}(\text{terms}(s)))] \wedge \\
\text{wang}([\text{reduced2}(s; \text{amblis}(\text{terms}(s)))]])
\]

This is an apparently ambiguous function because of the presence of \text{amblis}. However, it is not hard to show that actually it converges for each sequent \text{s} and that the result is \text{T} or \text{F} according to whether \text{s} is a theorem. This property is hereditary and hence applies to any descendant of the function \text{wang}.

Particular functions realizing Wang's algorithm choose the term to be reduced in a sequent in a sophisticated way. However, we need only show that they are descendants of \text{wang} in order to show that they give the right answer.

All this suggests that the concept of ambiguous function may have some importance in the theory of computation.
REFERENCES


CS-TR Scanning Project
Document Control Form

Report # Aim - 14

Each of the following should be identified by a checkmark:
Originating Department:
☒ Artificial Intelligence Laboratory (AI)
☐ Laboratory for Computer Science (LCS)

Document Type:
☐ Technical Report (TR) ☒ Technical Memo (TM)
☐ Other: ____________________________

Document Information

Number of pages: 13 (17 images)
Not to include DOD forms, printer instructions, etc... original pages only.
Intended to be printed as:
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☒ Unknown ☒ Other: MIMESGRAPH

Check each if included with document:
☐ DOD Form ☐ Funding Agent Form ☐ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: __________________________________

Page Data:

Blank Pages (by page number): ____________________________

Photographs/Tonal Material (by page number): ____________________________

Other (note description/page number):

Description: 

IMAGE MAP: (1 - 13) 1 - 13

(14 - 17) SCANCONTROL, TRST'S (3)

Scanning Agent Signoff:
Date Received: 11/30/95 Date Scanned: 12/13/95 Date Returned: 12/14/95

Scanning Agent Signature: _______________

Rev 9/94 DSLCS Document Control Form crcdform.wpd
Scanning Agent Identification Target

Scanning of this document was supported in part by the Corporation for National Research Initiatives, using funds from the Advanced Research Projects Agency of the United States Government under Grant: MDA972-92-J1029.

The scanning agent for this project was the Document Services department of the M.I.T Libraries. Technical support for this project was also provided by the M.I.T. Laboratory for Computer Sciences.

Scanned
Date: 12/13/1995
M.I.T. Libraries
Document Services