Do Not Borrow 13p

Artificial Intelligence Project--RLE and MIT 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 taut-
ology according to Wang's algorithm. Wang's algorithm is an
excellent example of the kind of algorithm which is convenient-
ly 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 pre-
ferable 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 $\sim$ (not),
$\&$ (conjunction), $\vee$ (disjunction), $\supset$ (implication), $\equiv$ (bi-
conditional), 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
$\sim\phi$, $\phi\,\&\,\psi$, $\phi\vee\psi$, $\phi\supset\psi$, $\phi\equiv\psi$ 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 $\pi\longrightarrow\rho$ 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. La-
ter on, a proof will be given of their completeness, i.e., all
intuitively valid dequents are provable, and of their consist-
ency, i.e., all provable sequents are intuitively valid.

P1. Initial rule: if λ, ζ are strings of atomic formulae, then
$\lambda \longrightarrow \zeta$ is a theorem if some atomic formula occurs on both
sides of the arrow.

In the ten rules listed below, λ and ζ 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 P1 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 , it is simpler, though not necessary, to replace the main connective by an arrow and proceed. For example:

*2.45. $\vdash\ :\ \sim(P\vee Q)\cdot\supset\cdot\sim P,$

*5.21: $\vdash\ :\ \sim P\ \&\ \sim Q\cdot\supset\cdot P\equiv Q$

can be rewritten and proved as follows.

*2.45 $\sim(P\vee Q)\longrightarrow\ \sim P$ (1)

     (1) $\longrightarrow\ \sim P, P\vee Q$ (2)

     (2) $P\longrightarrow P\vee Q$ (3)

     (3) $P\longrightarrow P,Q$

        VALID

                                     QED

*5.21. $\longrightarrow\ \sim P\ \&\ \sim Q\cdot\supset\cdot P\equiv Q$ (1)

     (1) $\sim P\ \&\ \sim Q\longrightarrow P\equiv Q$ (2)

     (2) $\sim P,\ \sim Q\longrightarrow P\equiv Q$ (3)

     (3) $\sim Q\longrightarrow P\equiv Q,P$ (4)

     (4) $\longrightarrow P\equiv Q,P,Q$ (5)

     (5) $P\longrightarrow Q,P,Q$

        VALID

     (5) $Q\longrightarrow P,P,Q$

        VALID

P2a.   Rule $\longrightarrow\sim$: If $\phi,\zeta\longrightarrow\lambda,\rho$, then $\zeta\longrightarrow\lambda,\sim\phi,\rho.$

P2b.   Rule $\sim\longrightarrow$: If $\lambda,\rho\longrightarrow\tau,\phi$, then $\lambda,\sim\phi,\rho\longrightarrow\tau.$

P3a.   Rule $\longrightarrow\&$: If $\zeta\longrightarrow\lambda,\phi,\rho$ and $\zeta\longrightarrow\lambda,\psi,\rho$, then $\zeta\longrightarrow\lambda,\phi\ \&\ \psi,\rho.$

P3b.   Rule $\&\longrightarrow$: If $\lambda,\phi,\psi,\rho\longrightarrow\tau$, then $\lambda,\phi\ \&\ \psi,\rho\longrightarrow\tau.$

P4a.   Rule $\longrightarrow\vee$: If $\zeta\longrightarrow\lambda,\phi,\psi,\rho$, then $\zeta\longrightarrow\lambda,\phi\vee\psi,\rho.$

P4b.   Rule $\vee\longrightarrow$: If $\lambda,\phi,\rho\longrightarrow\tau$ and $\lambda,\psi,\rho\longrightarrow\tau$, then $\lambda,\phi\vee\psi,\rho\longrightarrow\tau.$

P5a.   Rule $\longrightarrow\supset$: If $\zeta,\phi\longrightarrow\lambda,\psi,\rho$, then $\zeta\longrightarrow\lambda,\phi\supset\psi,\rho.$

P5b.   Rule $\supset\longrightarrow$: If $\lambda,\psi,\rho\longrightarrow\tau$ and $\lambda,\rho\longrightarrow\tau,\phi$, then $\lambda,\phi\supset\psi,\rho\longrightarrow\tau.$

P6a.   Rule $\longrightarrow\equiv$: If $\phi,\zeta\longrightarrow\lambda,\psi,\rho$ and $\psi,\zeta\longrightarrow\lambda,\phi,\rho$, then $\zeta\longrightarrow\lambda,\phi\equiv\psi,\rho.$

P6b.   Rule $\equiv\longrightarrow$: If $\phi,\psi,\lambda,\rho\longrightarrow\tau$ and $\lambda,\rho\longrightarrow\tau,\phi,\psi$, then $\lambda,\phi\equiv\psi,\rho\longrightarrow\tau.$

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

$$s: \quad \varphi_1, \ldots, \varphi_n \longrightarrow \psi_1, \ldots, \psi_m$$

is represented by the S-expression

$$s^1: \quad (\text{ARROW}, (\varphi_1^1, \ldots, \varphi_n^1), (\psi_1^1, \ldots, \psi_m^1))$$

where in each case the ellipsis ... 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	(NOT, φ^1)
2. $\varphi \,\&\, \psi$	becomes	$(\text{AND}, \varphi^1, \psi^1)$
3. $\varphi \lor \psi$	becomes	$(\text{OR}, \varphi^1, \psi^1)$
4. $\varphi \supset \psi$	becomes	$(\text{IMPLIES}, \varphi^1, \psi^1)$
5. $\varphi \equiv \psi$	becomes	$(\text{EQUIV}, \varphi^1, \psi^1)$

Thus the sequent

$$\sim P \,\&\, \sim Q \longrightarrow P \equiv Q, R \lor S$$

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:

theorem $[s]$ = th1$[\text{NIL}; \text{NIL}; \text{cadr}[s]; \text{caddr}[s]]$

th1$[a1; a2; a; c]$ = $[\text{null}[a] \longrightarrow \text{th2}[a1; a2; \text{NIL}; \text{NIL}; c]; T \longrightarrow$
member$[\text{car}[a]; c] \lor [\text{atom}[\text{car}[a]] \longrightarrow$
th1$[[\text{member}[\text{car}[a]; a1] \longrightarrow a1; T \longrightarrow \text{cons}[\text{car}[a]; a1]]; a2$
cdr$[a]; c]; T \longrightarrow \text{th1}[a1; [\text{member}[\text{car}[a]; a2] \longrightarrow a2;$
$T \longrightarrow \text{cons}[\text{car}[a]; a2]]; \text{cdr}[a]; c]]]$

$$th2[a1;a2;c1;c2;c] = [null[e] \rightarrow th[a1;a2;c1;c2];$$
$$atom[car[c]] \rightarrow th2[a1;a2;[member[car[c];c1] \rightarrow c1;T \rightarrow$$
$$cons[car[c];c1]];c2;cdr[c]];T \rightarrow th2[a1;a2;c1;[$$
$$member[car[c];c2] \rightarrow c2;T \rightarrow$$
$$cons[car[c];c2]];cdr[c]]]$$
$$th[a1;a2;c1;c2] = [null[a2] \rightarrow \sim null[c2] \wedge thr[$$
$$car[c2];a1;a2;c1;cdr[c2]];T \rightarrow th\ell[car[a2];a1;$$
$$cdr[a2];c1;c2]]$$

th 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:

a1: atomic formulae on left side of arrow

a2: other formulae on left side of arrow

c1: atomic formulae on right side of arrow

c2: other formulae on right side of arrow

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 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 thℓ or thr. We have

$$th\ell[u;a1;a2;c1;c2] = [$$
$$car[u] = NOT \rightarrow th1r[cadr[u];a1;a2;c1;c2]$$
$$car[u] = AND \rightarrow th2\ell[cdr[u];a1;a2;c1;c2]$$
$$car[u] = OR \rightarrow th1\ell[cadr[u];a1;a2;c1;c2] \wedge th1\ell[$$
$$caddr[u];a1;a2;c1;c2]$$
$$car[u] = IMPLIES \rightarrow th1\ell[caddr[u];a1;a2;c1;c2] \wedge th1r[$$
$$cadr[u];a1;a2;c1;c2]$$
$$car[u] = EQUIV \rightarrow th2\ell[cdr[u];a1;a2;c1;c2] \wedge th2r[$$
$$cdr[u];a1;a2;c1;c2]$$
$$]$$

$$thr[u;a1;a2;c1;c2] = [$$

$$car[u] = NOT \rightarrow th1\ell[cadr[u];a1;a2;c1;c2]$$

$$car[u] = AND \rightarrow th1r[cadr[u];a1;a2;c1;c2] \wedge th1r[$$
$$caddr[u];a1;a2;c1;c2]$$

$$car[u] = OR \rightarrow th2r[cdr[u];a1;a2;c1;c2]$$

$$car[u] = IMPLIES \rightarrow th11[cadr[u];caddr[u];a1;a2;c1;c2]$$

$$car[u] = EQUIV \rightarrow th11[cadr[u];caddr[u];a1;a2;c1;c2] \wedge th11[$$
$$caddr[u];cadr[u];a1;a2;c1;c2]$$

The functions _th1ℓ_, _th1r_, _th2ℓ_, _th2r_, _th11_ distribute the parts of the reduced formula to the appropriate places in the reduced sequent.

These functions are

$$th1\ell[v;a1;a2;c1;c2] = [atom[v] \rightarrow member[v;c1] \vee$$
$$th[cons[v;a1];a2;c1;c2];T \rightarrow member[v;c2] \vee$$
$$th[a1;cons[v;a2];c1;c2]$$

$$th1r[v;a1;a2;c1;c2] = [atom[v] \rightarrow member[v;a1] \vee$$
$$th[a1;a2;cons[v;c1];c2];T \rightarrow member[v;a2] \vee$$
$$th[a1;a2;c1;cons[v;c2]]$$

$$th2\ell[v;a1;a2;c1;c2] = [atom[car[v]] \rightarrow member[car[v];c1] \vee$$
$$th1\ell[cadr[v];cons[car[v];a1];a2;c1;c2];T \rightarrow member[$$
$$car[v];c2] \vee$$
$$th1\ell[cadr[v];a1;cons[car[v];a2];c1;c2]$$

$$th2r[v;a1;a2;c1;c2] = [atom[car[v]] \rightarrow member[car[v];a1] \vee$$
$$th1r[cadr[v];a1;a2;cons[car[v];c1];c2];T \rightarrow member[$$
$$car[v];a2] \vee$$
$$thr[cadr[v];a1;a2;c1;cons[car[v];c2]]]$$

$$th11[v1;v2;a1;a2;c1;c2] = [atom[v1] \rightarrow member[v1;c1] \lor$$
$$thir[v2;cons[v1;a1];a2;c1;c2];T \rightarrow member[v1;c2] \lor$$
$$thir[v2;a1;cons[v1;a2];c1;c2]]$$

Finally the function member is defined by

$$member[x;u] = \sim null[u] \land [equal[x;car[u]] \lor member[x;cdr[u]]]$$

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

```
DEFINE ((
(THEOREM (LAMBDA (S) (TH1 NIL NIL (CADR S) (CADDR S))))

(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)))))

(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)))))))

(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)
(TH1L (CADDR U) A1 A2 C1 C2) ))
((EQ (CAR U) (QUOTE IMPLIES)) (AND (TH1L (CADDR U) A1 A2 C1
C2) (TH1R (CADR U) A1 A2 C1 C2) ))
((EQ (CAR U) (QUOTE EQUIV)) (AND (TH2L (CDR U) A1 A2 C1 C2)
(TH2R (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)
(TH1R (CADDR U) A1 A2 C1 C2) ))
((EQ (CAR U) (QUOTE OR)) (TH2R (CDR U) A1 A2 C1 C2))
((EQ (CAR U) (QUOTE IMPLIES)) (TH11 (CADR U) (CADDR U)
A1 A2 C1 C2))
((EQ (CAR U) (QUOTE EQUIV)) (AND (TH11 (CADR U) (CADDR U)
A1 A2 C1 C2) (TH11 (CADDR U) (CADR 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))))
    )))


(TH11 (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. Ac-
cidentally, it turns out that these arguments essentially con-
stitute a proof in Wang's style of the theorem.

In order to apply the method to the sequent

p → p∨q

we write

THEOREM
((ARROW,(P),((OR,P,Q))))
()

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 giv-
en 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 al-
gorithm in section 1 it is not specified how the term in the se-
quent to be acted on first is chosen. This choice does not af-
fect 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 ambigu-
ous 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 com-
pletely determined by the function f and the n-tuplet $x_1, \ldots x_n$
but may be any member of a set $R(\#, x_1, \ldots, x_n)$. No mechanism,
probabilistic or otherwise is provided for deciding which ele-
ment will be chosen. Ambiguous functions may be combined to
get new functions by composition, conditional expressions, and
recursion.

In order to consider the class $C\mathcal{Q}\{\mathcal{F}\}$ of ambiguous func-
tions computable in terms of a class of basic functions $\widetilde{\mathcal{F}}$,
(This corresponds to $C\{\mathcal{F}\}$ in [4]) we introduce the ambiguity
operator amb. 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

$$amblis[\ell] = [\sim null[\ell] \wedge null[cdr[\ell]] \rightarrow car[\ell];$$
$$\sim null[\ell] \wedge \sim null[cdr[\ell]] \rightarrow amb[car[\ell]; amblis[cdr[\ell]]]]$$

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 pos-
sible 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 reduce1[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

$$wang[s] = [reduced[s] \to dup[s]; T \to$$
$$wang[reduced1[s;amblis[terms[s]]]] \wedge$$
$$wang[reduced2[s;amblis[terms[s]]]]]$$

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

1. Wang, Hao, "Toward Mechanical Mathematics", IBM Journal of Research and Development, Vol.4, No.1, January 1960.

2. McCarthy, John, "Recursive Functions of Symbolic Expressions and their Computation by Machine", Quarterly Progress Report No.53, Research Laboratory of Electronics, MIT, April 1959.

3. McCarthy, John, Fox, Phyllis, et. al., "LISP Programmers' Manual", Artificial Intelligence Project, MIT, January 1960.

4. McCarthy, John, "Notes on the Theory of Computation", MIT Computation Center, to appear.

CS-TR Scanning Project
Document Control Form

Date : 11/30/95

Report # AIM – 14

Each of the following should be identified by a checkmark:
Originating Department:

☒ Artificial Intellegence 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 intstructions, etc... original pages only.

Originals are:

☒ Single-sided or

☐ Double-sided

Intended to be printed as :

☐ Single-sided or

☒ Double-sided

Print type:

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

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 :   Page Number:
IMAGE MAP: (1-13) 1-13
           (14-17) SCANCONTROL, TRGT'S (3)
_____
_____
_____

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

Scanning Agent Signature:_____Michael W. Cook_____   Rev 9/94 DS/LCS Document Control Form cstrform.vsd

Scanning Agent Identification Target

darptrgt.wpw Rev. 9/94