III

The Complete Program for the Logic Theorist

This Section is divided into two parts. The first part constitutes the program as described in the text, including the following routines: Ex; MCh, MDt, MSb; LMc, LSb, LRp→v, LRpv→, VV, VCt; CX; CSm, CD, D, FK, FH, FJ. These routines are preceded by a list of the most important primitive IP's--those that are used in several routines. Following each routine is a supplementary list of primitive IP's used in the definition of that routine.

The second part of this Section consists of routines for five IP's--those Store instructions that are marked with asterisks (*)--which up to this point have been treated as primitives.

Principal Primitive Instructions

A	OPER	L	C	R	B	
	B				b	Branch to b (→ b).
	BHB					In higher instruction, → b.
	BHN					In higher instruction, → next.
	FEF	x		y	b	Find the first E in A(x) and put in y; if none, → b.
	FEN	x		y	b	Find the E in A(x) next after E(y), put in y; then → b. If none (end of list), next.
	FL	x		y		Find EL(x) and put in y; if none, leave y blank.
	FR	x		y		Find ER(x) and put in y; if none, leave y blank.
	PE	x		y		Put E(x) in E(y). E(x) remains.
	S	x				Store E(x) back in A(x) (match on P); if not there, store E(x) at end of A(x).
	SEN	x		y		Store E(x) as next E in A(y); E(x) becomes last item in A(y).
	*SY	x		y		Store a copy of X(x) at (new) A(y). E(x) = M.
	TC	x			b	If C(x) = → (implies), → b.
	TV	x			b	If E(x) = V, → b.

A	OPER	L	C	R	B	Seg.	Executive routine
	Ex						
	(Read problem X)					R	
	(Put EM(X) in 1)						
	-MSb	1			G	MSb	
A	-MDt	1			G	MDt	
	-MCh	1			G	MCh	
	SEN	1	Q				X(1) is finished.
	CWG				H	CW	
B	FEF	P	1		H	CK	Find problem with lowest K.
	NK	1					
C	-FEN	P	2		D		
	NK	2					
	CKG	2	1		C		
	PE	2	1				
	PK	2	1				
	B				C		
D	E	1	P			CX	Remove duplicates of previous problems.
	FEF	Q	3		F		
E	CX	1	3		B		
	FEN	Q	3		E		
F	B				A		
G	(Write proof.)					WP	Succeeds in proving P.
	(X(1) a theorem)					ST	
	(Stop)						
H	(Write:no proof)					WNP	Fails to find proof.
	(Stop)						

Primitives

CKG	x y	b	If K(x)>K(y), →b.
CWG			If W (work done) > limit, →b.
E	x y		Erase E(x) in A(y).

Note: There are six IP's in the executive routine that are not formally defined in LT. These are written in parentheses above: read problem, find problem and put in working memory 1, write proof, store expression as theorem, write "no proof", and stop.

A	OPER	L	C	R	B	Seg.	
							Chaining method
							If can't prove X(x) by chaining, →b; Store new problems in P.
	MCh	x			b		
	-TC→	L			D	T	C(x) must be →.
	VV	L					
	FL	L	1				
	FR	L	2				
	FEF	T	3		D		
A	-TC→	3			C		T must have C = →
	VV	3					
	SX	3	4				Copy, so as to work on T.
	FL	4	5				
	FR	4	6				
	-CSm	1	5		B	SmF	
	-LMc	5	1		E	McF	
B	-CSm	2	6		C	SmB	
	-LMc	6	2		F	McB	
C	FEN	T	3		A		Find next T and repeat.
D	BHB						
E	PE	2	5			P	Put E(2) and E(6) in
	PE	6	1				proper wkg. memory.
F	AM	7					Create EM for new X.
	PC→	7					Fix connective.
	S	7					Store parts.
	SEN	7	P				
	SXL	1	7				
	SXR	5	7				
	MSb	7			C	MSb	
	BHN						

Primitives

PC→	x	Put C(x) = → (implies).
*SXL	x y	Store X(x) in A(y) as XL(y).
*SXR	x y	Store X(x) in A(y) as XR(y).

A	OPER	L	C	R	B	Seg.
	MDt	x		b		
	FEF	T	1	C	T	
A	TC→	1		B		
	VV	1				
	FR	1	2			
	VV	L				
	CSm	L	2	D	SmV	
	VCt	L				
	CSm	L	2	D	SmCt	
B	FEN	T	1	A		
C	BHB					
D	SX	1	3			
	FR	3	4			
	LMc	4	L	B	Mc	
	FL	3	5		P	
	SXM	5	6			
	S	6				
	SEN	6	P			
	MSb	6		B	MSb	
	BHN					

Detachment method
If can't prove X(x) by detachment→b. Store new problems in P.

T must have C =→.

Change view.

Find next T and repeat.

Copy so can work on T.

Create new X
Store away fixed ME

Primitives

*SXM	x y	Store X(x) at (new) A(y) as main expression.

A	OPER	L	C	R	B	Seg.
	MSb	x		b		
	NAW					NAW
	VV	L				Sm
	FEF	T	1	C		
A	VV	1				
	CSm	L	1	D		
B	FEN	T	1	A		
C	BHB					
	SX	1	2			Mc
	LMc	2	L			
	BHN					

Substitution method
If can't prove X(x) by substitution→b.

Count one unit of work.

Find next T and repeat.

Primitives

NAW		Add one to W (work done).

```
A  OPER  L C R B    Seg.

                    Matching routine
                    Match X(x) to X(y); if
   LMc   x y   b    can't, →b.

   CGG   C L   A    T
   CGG   L C   C
   TV    L     E
   TV    C     D
  ¬CC    L C   F
   FL    L 1       LMc
   FL    C 2
   LMc   1 2   H        Mc left subexpression.
   FR    L 3
   FR    C 4
   LMc   3 4   H        Mc right subexpression.
   BHN

A  TV    L      H   Sby
B ¬TF    L      H
   NSGG  L C
   FM    L 5          Assures Sb everywhere.
   LSb   C L 5
   BHN

C  TV    C      H   Sbx
D ¬TF    C      H
   NSGG  C L
   FM    C 5          Assures Sb everywhere.
   LSb   L C 5
   BHN

E ¬TV    C      B   CN
  ¬CN    L C    B
   BHN

F ¬LRp→v L      G   Rp   LRp's are self-testing.
   LRpv→ L      H
G  LMc   L C    H

H  BHB

Primitives

   CC     x y   b   If C(x) = C(y), →b.
   CGG    x y   b   If G(x) > G(y), →b.
   CN     x y   b   If N(x) = N(y), →b.
   FM     x y      Find EM(x) and put in y.
   NSGG   x y      Subtract G(x) from G(y).
   TF     x     b   If E(x) is free, →b.
```

A	OPER	L	C	R	B	Seg.	
	LSb	x	y	z			Substitution routine Substitute X(x) for E(y) (=V) in X(z) (=M).
	FEF	L	1		F	F	
A	CPS	1	1		B		E(1) must belong to X(x)
	CN	1	C		G		
B	FEN	L	1		A		
C	FEF	R	2		F	Sb	Search through X(z)
D	CN	2	C		E		
	PE	L	3				
	NAGG	2	3				G's add in Sb
	SXE	3	2				
E	FEN	R	2		D		Find next E(z), repeat
F	BHN						
G	AN	4				LSb	
	LSb	4	C	R			
	B				C		

Primitives

AN	x		Assign an unused name to E(r)
CN	x	b	If N(x) = N(y)→b.
CPS	x y	b	If E(x) subelement of E(y)→b (P(x)⊃ P(y)).
NAGG	x y		Add G(x) to G(y); result in G(y)
*SXE	x y		Store X(x) in A(y) in place of E(y) (=V).

A	OPER	L	C	R	B	Seg.	
	LRp→v	x			b		Replacement of with v. If C(x)=→, replace with v; if not →b.
	TC→	L			A	T	
	BHB						
A	PCv	L				Pv	Fix E'(x).
	S	L					
	FL	L	1				Fix EL(x)
	NAG	1					
	S	1					
	BHN						

Primitives

NAG	x	Add one to G(x)
PCv	x	Put C(x) = v.

A	OPER	L	C	R	B	Seg.
	LRpv→	x			b	
	-TCv	L		A		T
	FL	L	1			
	TG	1		C		
	-TV	1		A		
	-TSb	1		B		
A	BHB					
B	PE	1	2			Sb
	NAG	2				
	FM	2	1	3		
	FL	L	1			
C	PC→	L				P→
	S	L				
	NSG	1				
	S	1				
	BHN					

Replacement of v with →
If C(x)=v and G(EL(x)) >0, replace v with →; if not b.

Fix x

Primitives

FM	x y	Find EM(x) and put in y.
NAG	x	Add one to G(x)
NSG	x	Subtract one from G(x)
PC	x	Put C(x) = →
TGG	x b	If G(x) >0 →b.

A	OPER	L	C	R	B	Seg.
	VV	x				
	FEF	L	1			T
A	PUB	1				
	-TV	1			B	
	PU	1				
B	S	1				
	FEN	L	1		A	
	BHN					

View variables as units.

Erase old unit

Find next E and repeat

Primitives

PU	x	Put E(x) to be a unit. (U).
PUB	x	Put U(x) to be blank

A OPER L C R B Seg.

View as contrasted
Make units of binary expressions and isolated variables

VCt x

```
      TV    L     C   T
      FL    L 1       VCt
      FR    L 2
      TV    1     B
      VCt   1               Recursion
      TV    2     E
A     VCt   2               Recursion
      PUB   L
      S     L
      BHN

B     TV    2     D
      PUB   1         Ct    Blank V's of Ct unit
      S     1
      PUB   2
      S     2
      TN    L     C         Give it a name if not
      AN    L                have one
C     PU    L
      S     L
      BHN

D     PU    1         VV    Make left (isolated)
      S     1                variable a unit
      B           A         XR(x) still to be done

E     PU    2
      S     2               Make right (isolated)
      BHN                    variable a unit
```

Primitives

```
AN    x          Assign E(x) an unused name.
(See VV for PU and PUB)
TN    x    b     If E(x) has a name  b.
```

A	OPER	L	C	R	B	Seg.	
							Compare expressions routine
	CX	x	y		b		Compare X(x) with X(y); if they match, → b.
	CGG	L	C		B	T	
	CGG	C	L		B		G(L) = G(R), otherwise B.
	TV	L			A		
	TV	C			B		
	-CC	L	C		B		C(L) = C(R)
	FL	L	1			CX	Recursion down tree of
	FL	C	2				expressions.
	-CX	1	2		B		
	FR	L	3				
	FR	C	4				
	-CX	3	4		B		
	BHB						
A	-TV	C			B	CN	L and C both variables;
	-CN	L	C		B		with identical names.
	BHB						
B	BHN						

Primitives

(For CC, CGG, and CN, see LMc)

A	OPER	L	C	R	B	Seg.	
							Similar expressions test
	CSm	x	y		b		If DL(x) = DL(y) and DR(x) = DR(y) → b.
	FL	L	1			D	
	FR	L	2				
	D	1					
	D	2					
	FL	C	3				
	FR	C	4				
	D	3					
	D	4					
	-CD	1	3		A	CD	
	-CD	2	4		A		
	BHB						
A	BHN						

```
A   OPER  L C R B   Seg.
                                Compare descriptions
                                If K(x) = K(y), J(x) =
    CD    x y   L                 J(y), and H(x) = H(y)→ b.

    CK    L C   A                 Def: If K(x) = K(y) → b.
    CJ    L C   A                 Def: If J(x) = J(y) → b.
    CH    L C   A                 Def: If H(x) = H(y)→ b.
    BHB

A   BHN
```

```
A   OPER  L C R B   Seg.
                                Describe
A   D     x                     Describe

    NK    x
    NJ    x
    NH    x
    BHN   x
```

```
A   OPER  L C R B   Seg.

    NK    x                     Count levels

    TU    L       A   T
    TB    L       B
    FL    L 1         NK
    NK    1
    FR    L 2
    NK    2
    CKG   2 1   C     CK
    PK    1 L         KL
A   NAK   L
B   BHN

C   PK    2 L         KR
    B           A
```

Primitives

```
    (See Ex for CKG)
    NAK   x           Add one to K(x)
    PK    x y         Put K(x) in K(y)
    TB    x     b     If E(x) is blank →b.
    TU    x     b     If E(x) is a unit →b.
```

A OPER L C R B Seg.

NJ x

Count distinct variables

```
     AA   1
     FEF  L 2   E   F   Find first E of X(x)
A  -CPS   2 L   D
   -TU    2     D
     FEF  1 3   C       Find first V of list
B    CN   2 3   D   CN
     FEN  1 3   B       Find next V of list
C    SEN  2 1
     NAJ  L         A
     FEN  L 2   A       Find next E of X(x)
E    BHN
```

Primitives

```
     AA   x          Assign an unused list to A(x)
     CN   x y   b    If N(x) = N(y)→ b.
     CPS  x y   b    If E(x) subelement of E(y)→ b.
                      (P(x) ⊃ P(y)).
     NAJ  x          Add one to J(x)
     TU   x     b    If E(x) is a unit →b.
```

A OPER L C R B Seg.

NH x

Count variable places

```
     FEF  L 1   C
A  -CPS   1 L   B
   -TU    1     B
     NAH  L
B    FEN  L 1   A
C    BHN
```

Primitives

```
     CPS  x y   b    If E(x) subelement of E(y)→ b.
                      (P(x) ⊃ P(y)).
     NAH  x          Add one to H(x)
     TU   x     b    If E(x) is a unit →b.
```

PART 2: Reduction of procedural processes [*S]

The Store instructions that rewrite expressions in various ways can be reduced to processes more like the rest of the primitive set. The new primitives required are (a) two (PA and CP) which belong to types of operations already considered, and (b) four of a new type to manipulate the P sequences. The latter operations insert and delete subsequences from the front end of a given sequence. Thus if P = LRRL and P' = LRRLRLR, then P'' = P' = P = RLR and P'' + P = LRRLRLR. Observe that subtraction can only be performed when the subtrahend is an initial segment of the minuend, and also that addition is not commutative. All these routines involve bringing in the elements, one by one, modifying them and storing them in the new list.

Store a copy of X(x) at (new) A(y) (E(x) = M).

```
A   OPER  L C R B
A   OPER  L C R B

    SX    x y

      AA    C
      FEF   L 1   B
A     PE    1 2
      PM    C 2
      S     2
      FEN   L 1   A
A     BHN
```

Store X(x) at (new) A(y) as main expression

```
A   OPER  L C R B

    SXM   x y

      AA    C
      FEF   L 1   C
A     CPS   1 L   B
      PE    1 2
      PM    C 2
      HSPP  L 2
      S     2
B     FEN   L 1   A
C     BHN
```

Store X(x) in A(y) in place of E(y) (E(y) =V) (take E(x) from w.m.)

```
A   OPER  L C R B

    SXE   x y

      FEF   L 1   D
A     CP    L 1   E
      CPS   1 L   C
      PE    1 2
B     PM    C 2
      HSPP  L 2
      HAPP  C 2
      S     2
C     FEN   L 1   A
D     BHN

E     PE    L 2
      B           B
```

Store X(x) in A(y) as XL(y).

A	OPER	L	C	R	B
	SYL	x	y		
	FEF	L	1		C
A	CPS	1	L		B
	PE	1	2		
	PM	C	2		
	HSPP	L	2		
	HAPL	2			
	HAPP	C	2		
	S	2			
B	FEN	L	1		A
C	BHN				

Store X(x) in A(y) as XR(y).

A	OPER	L	C	R	B
	SXR	x	y		
	FEF	L	1		C
	CPS	1	L		B
	PE	1	2		
	PM	C	2		
	HSPP	L	2		
	HAPR	2			
	HAPP	C	2		
	S	2			
	FEN	L	1		
	BHN				

Primitives

AA	x		Assign an unused list to A(x)
CP	x	y	If P(y) = P(y)→ b (locates "name" - element even though V_0, G_0 etc. have been modified).
CPS	x	y	If E(x) subelement of E(y)→ b (P(x) ⇒ P(y)).
HAPL	x		Add a Left to front of P(x).
HAPR	x		Add a Right to front of P(x).
HAPP	x	y	Add P(x) to front of P(y).
HSPP	x	y	Subtract P(x) from front of P(y).
PA	x	y	Put A(x) in A(y).

Conclusion

In this paper we have specified in detail an information processing system that is able to discover, using heuristic methods, proofs for theorems in symbolic logic. We have confined ourselves to description, and have not attempted to generalize in abstract form about complex information processing. Because of the nature of the description, involving considerable rigor and detail, it may be useful to set out in conclusion the main features of LT, especially as these appear to reflect basic characteristics of complex systems.

First of all, LT can be specified at all only because its structure is basically hierarchical, and makes repeated use of both iteration and recursion. So true is this, that one of LT's main features, the use of a problem-subproblem hierarchy, is hardly visible in the program at all.

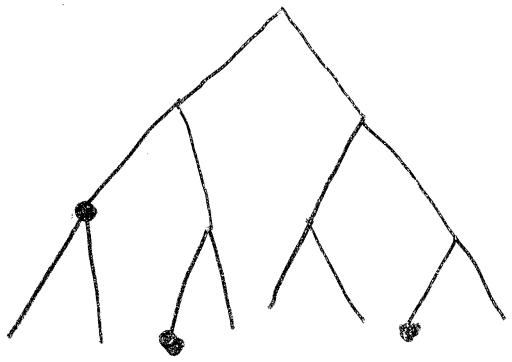
LT offers no guarantee of finding a proof; on the other hand, it brings to its task a number of different heuristic methods for achieving its goals. All of these methods are important in making LT sufficiently powerful to find proofs in most cases, and to find them with a reasonable amount of computation, but not all of them are essential. Without chaining, for instance, LT could still function. The methods MSb and MDt still provide it with ways to prove theorems--and even some theorems more easily provable by MCh would yield to the more directly "brute force" approach of the other two.

LT uses similarity-testing and matching as a multi-stage search and selection process. The questions of efficiency involved in such processes have already been commented upon in Section II. Additional variation and complexity enters the program through the alternative modes, VV

and VCt, for perceiving the logic expressions in the course of testing similarity and of matching.

In these and other ways the logic theorist is an instructive instance of a complex information process. We expect to learn more about such processes when we have realized the logic theorist in a computer and studied its operations empirically; and when the logic theorist will have been joined by similar systems capable of performing other complex information processing tasks.

Investigation trees with encouragement structures.



Suppose at any given time we have some choices as to what to investigate next. Carrying out an elementary investigating process can put any point on the tree once removed from those we already occupy, in our grasp.

Certain points are called definite results. At these points there are encouragement cues which may take the forms

a. irrelevant
b. definitely part of the answer
c. probability of being on the right track.

If a cue says probability p of being on the right track, subsequent definite results have a certain probability of giving definite cues. As you branch away from the right track, the density of encouraging cues drops, as may also the density of definite results.

Questions:

1. With only the cue structure, how fast can one move down the tree?
2. My impression is that there should be a sharp drop from easy problems to impossible ones.
3. Also a small improvement in the probabilities doesn't help much if the probabilities are around 1/10, 1/100.
4. An improvement in the density of cue points and in the number of decisive cues helps a lot.
5. If the model does not have the desired properties, how can it be carpentered into one which does?