11 July 78

ATP 42

Agenda Handling And Other Routines For The New Prover

W. W. Bledsoe and A. Michael Ballantyne

ATP 42 July 1978

This work was supported by NSF grant MCS77-20701, and ARPA Grant F446-20-73-C-0074 (Carngie-Mellon University).

1. Introduction

This paper gives the algorithms for some of the routines in our new prover, and explains their use. Most of the algorithms given here are in LISP or abbreviated LISP.

For a general description of our new prover, the reader is referred to [5], and to [1,2,4] for background on our earlier provers, which form a basis for the new one.

2. Algorithms

The main routine in the prover is IMPLY. A theorem to be proved is passed to IMPLY via CYCLE which skolemizes the theorem and sets up initial conditions for the proof.

This prover is designed to "stand alone" (i.e., not interactive) and all explanations are given with that in mind. However, it retains many of the interactive features of earlier provers [1,3]. For example it has an IMPLY-STOP feature whereby (if it is activated) the program will stop each time it enters the routine IMPLY to allow the human user to give directions, make changes, and obtain information. The user then presses the escape key on the console to cause the program to continue.

IMPLY

IMPLY is a lot like the IMPLY of our previous provers [1-4], except that now it operates with an agenda and utilizes several additional parameters.

Briefly, the parameters are:

SL - Subgoal label - For identifying the particular subgoal on which it is working. (This was called "TL" in previous provers).

DB - Contexual Data Base (See [2]).

H - The hypothesis of the current subgoal. It may be NIL.

C - The hypothesis of the current subgoal.

A - Authority - Authority for IMPLY to set up an extended agenda. See below.

AGENDA - A list of tasks on which IMPLY will work in order to try to prove the current subgoal. See below.

TIMEL - Timelimit allotted for this subgoal.

PV - List of "Protected Variables" (See [2] Section 3)

CS - Critical subgoal parameter

```
1     (IMPLY  (λ (SL  DB  H  C  A  AGENDA  TIMEL  PV  CS)
2        (PROG  (ANSWER  ANSWERS  X  Y  RESULT-TREE)
      IMPLY  STOP
      (RETURN  (COND
3     [(AND  (SETQ  RESULT-TREE  (ASSOC  UNIVERSAL-RESULTS-LIST  C))  NIL)]
4     [(KNOWN  C)]
5     [(AND  (SETQ  ANSWER (IMPLY2  AGENDA  TIME2))  NIL]
6     [(SUCCEED  ANSWER)  ANSWER]
7     [(FAIL  ANSWER)  ANSWER]
8     [(TIME-FAILURE  ANSWER)  (COND
9        [(EQUAL  A1)  (POST-MORTEM  ANSWER)
10                     (IMPLY  A = 0)] *
11       [T  (LIST  'TL (LIST  'AGENDA  AGENDA))] ) ]
12    [(UNPROVED  ANSWER)  (COND
13       [(EQUAL  A 1)  (POST-MORTEM  ANSWER)
14                      (IMPLY  A = 0)]
15       [(T  (LIST 'UP (LIST 'AGENDA  AGENDA))] ) ]
16     )))
```

When IMPLY is called with these 9 parameters it first checks URL (the universal results list) in Steps 3-4 to see whether this subgoal has already been proved or whether there is a counterexample for it. If either, it returns that fact.

At Step 5 IMPLY calls the routine IMPLY2 to process the whole agenda (with a given timelimit TIME2, which is computed by IMPLY). The answer from this call

---

*We indicate by (IMPLY A = 0) a call to IMPLY with A = 0 and all other parameters with their current values.

to IMPLY2 is stored in "ANSWER"; it will have one of four forms:

(θ Other information)

(NIL Other information)

('TL Other information)

('UP Other information)

The last three are failures:

NIL represents an absolute failure where a counterexample has been produced.

TL represents a timelimit failure.

UP represents an "unproved" situation where the prover has run out of things to do.

"Other information" represents additional facts about a proof or failure which are brought back and used by the POST-MORTEM operation.

θ , (or λ, etc.) represent a success case, where the subgoal has been proved. In this case, θ itself a doubleton θ = (σ(A-UNIT RL TY)) where σ is an ordinary substitution (or "T" for the empty substitution). See Section 2 of [2] for further explantion.

IF IMPLY succeeds, then its answer is returned to the routine which called it (Step 6), similarly for an outright failure (Step 7). But if it experiences a time failure (TL) or if the subgoal is unproved (UP), its action will depend on the value of the parameter A.

A is a parameter of IMPLY which indicates how far down the authority has been passed to set up an extended agenda. If

A = 0 it already has an extended agenda,

A = 1 it has authority to extend the agenda now,

A > 1 authority has not yet been given.

If IMPLY is called from IMPLY or from a routine called by IMPLY, the parameter A is increased by 1.

So when IMPLY fails by TL or UP, if $A = 1$ it first uses POST-MORTEM to critique its results and start an extended agenda (with $A = 0$). But if $A = 0$ or $A > 1$ it simply returns the whole agenda as part of its answer. (Steps 8-15). We will see what it means to "set up an extended agenda" when we discuss the routine IMPLY2.

IMPLY2

It is the job of IMPLY2 to process the AGENDA of IMPLY. It does this by evaluating its tasks one at a time.

IMPLY2 has two arguments, AG and TIME2. TIME2 is the timelimit imposed on it by IMPLY, and AG is initially the AGENDA from IMPLY. When $A \geq 1$, this will be the "standard agenda", i.e., the proof steps that were tried by our earlier provers [1-4] with some additions. This standard agenda is listed below and will be explained later. But after A has become 0 (see Steps 9-10, 13-14 of IMPLY), then the standard agenda is allowed to expand to include other tasks such as COUNTEREXAMPLE (i.e., try for a counterexample), LEMMAS, HEURISTIC, etc. which will be explained below.

```
1       (IMPLY2 (λ( AG TIME2)  (COND
2       [(NULL AG) (LIST 'UP ANSWERS)]
3       [(TIMELIMIT) (LIST 'TL ANSWERS)]
        [(AND (SETQ TASK (COND
4            [(EQUAL A 0) (HIGHEST AG)]
5            [T (CAR AG) )) NIL)]
6       [(AND (EQUAL A 0) (NOT TASK)) (LIST 'UP ANSWERS)]
7       [(AND (SETQ X (EVAL TASK)) NIL]
8       [(SUCCEED X) X ]
9       [(AND (EQ 'IMPLY (CARR TASK)) (POST-MORTEM-IMPLY X) NIL)]
10      [(FAIL X) (COND
11           [(EQUAL A 0) (IMPLY)]
12           [T (IMPLY2 (CDR AG) TIME22)] ) ]
13      [(AND (SETQ Y (COND [(EQUAL A 0) (IMPLY  )]
                            [T (IMPLY2 (CDR AG)) ) TIME22)) NIL)]
14      [(SUCCEED Y) Y]
15      [(FAIL Y) X ]
16      [(TIME-FAILURE X) (LIST 'TL (LIST 'X X) (LIST 'Y Y)]
17      [T                (LIST 'UP (LIST 'X X) (LIST 'Y Y)]
         )))
```

IMPLY2 (Steps 4-5) selects a task from AG and evaluates it (Step 7). This task is just the first one on the agenda unless $A = 0$ in which case it is the one with highest priority (Step 5). It sets x equal to this evaluation.

If x succeeds, then it returns x (Step 8). If not is post-mortems the results and updates the agenda (Step 9). If x fails and $A = 0$ it simply recalls IMPLY (Step 11) with the same arguments (but AGENDA has been updated), and if $A \geq 1$ it discards that task and proceeds to recall IMPLY2 with the remainder of

the tasks on AG (Step 12).

In case x has neither succeeded or failed (i.e., is either TL or UP) then (in Step 13) it either recalls IMPLY with the same arguments (but AGENDA has been updated) if $A = 0$, or calls IMPLY2 on the remainder of AG if $A \geq 1$, and sets the result to y. If y succeeds then y is returned (Step 14), but if it fails (Step 15), then x is returned (which is either unproved or a time failure).

At Step 16, each of x and y is either unproved or a time failure, so (in Steps 16-17) it returns the value for both x and y to be processed by POST-MORTEM (if $A = 0$).

STANDARD AGENDA

The Standard Agenda contains (essentially) the steps used by IMPLY in our earlier provers [1-4]. They are:

STANDARD AGENDA

(

3. AND-SPLIT
4. HSF (Higher Subgoal Failure) (if $A \geq 1$)
5. REDUCE (if $A \geq 1$)
6. PROVE $\leq$
7. OR-FORK
8. PROMOTE
9. HOA (if $A \geq 1$)
10. If $A = 1$ call (IMPLY $A = 0$)

)

Briefly these are as follows.

3. AND-SPLIT treats the case when the conclusion C is a conjunction $(A \wedge B)$. See "AND-SPLIT" below. (This was called AND-C in earlier

papers).

4. HSF returns failure in case this same subgoal has been encountered in a higher subgoal position.

5. REDUCE applies a set of rewrite rules to H and C.

6. PROVE $\leq$ is a special routine for handling inequalities (See [2]).

7. OR-FORK handles the case when C is a disjunction $(A \vee B)$.

8. PROMOTE is called when C has the form $(P \rightarrow Q)$. It simply recalls IMPLY with H replaced by $(H \wedge P)$ and C replaced by Q. (If that fails, it "reverse promotes", by replacing H by $(H \wedge \sim Q)$ and C by $\sim P$.

9. HOA is a routine charged mainly with manipulating the hypothesis H. See "HOA" below.

Step 10 is not a step of our earlier provers; it is used to convert from a standard agenda to an "extended" agenda when A = 1. It does this simply by putting A to 0 and recalling IMPLY.

In actuality, there are other items on our standard agenda (items 1,2,11-14), but they are "asleep" when $A \geq 1$. (Similarly, items 4,5,9 are put to sleep when A = 0).

These additional items (or tasks) are given below; others may be added later.

EXTENDED AGENDA ITEMS

(

1. TIMELIMIT
2. CE (counterexample) (if A = 0)

⋮

11. HOA-TASKS (if A = 0)
12. DEFINE-C (if A = 0)
13. LEMMAS (if A = 0)
14. HEURISTICS (if A = 0)

)

Items 11 - 14 act to generate other tasks to be added to the agenda. Accordingly they are called 'generators". They are allowed to act only if A = 0 , and then the one with highest priority acts first. After a generator has acted, it is then put to "sleep" (given a very low priority) or given a low priority, in which case it may be called upon again later to generate more tasks. Usually the new tasks that the generators place on the agenda, are calls to IMPLY with its various parameters specified.

CE is a routine which attempts to produce a counterexample to the subgoal (H → C). If one is found then absolute failure, (NIL NIL) is returned for this subgoal and the fact is stored on the URL. It is not called if the current subgoal is a "critical subgoal", a subgoal that must be true in order for the theorem to be true. The parameter CS (of IMPLY) will be equal to "T" for critical subgoals and "NIL" for non-critical ones. (See [6].)

HOA-TASKS is a replacement for HOA in case A = 0. It puts on the agenda a series of tasks (calls to IMPLY), which correspond to the calls to IMPLY that would have been made by HOA if it were allowed to operate. Once these additional tasks have been put on the agenda, HOA-TASKS is put to sleep. (See HOA-TASKS below).

DEFINE-C acts to expand the definition of the conclusion C, (or its main connective) (See [1].) It too is put to sleep when this is done and the corresponding task has been put on the agenda.

LEMMAS - acts to find one or more lemmas in the structured Data Base, and to put on the agenda tasks corresponding to the use of these lemmas. It may be called again later to find more lemmas (if possible). See LEMMAS below.

HEURISTICS - acts to use one or more heuristics to decide on a plan of action for proving the current subgoal. It too may be called again later.

Thus the agenda has 14 main items, 8 of which are used in the standard mode, and 4 are generators which generate other tasks for the agenda. These extended agendas then can get quite long, and it is important to carefully govern their use by the handling of their priority numbers, and in the allotting of timelimits to them when they are permitted to act.

HOA

HOA is an item on the standard agenda. (See above.) It is used mainly for manipulating the hypothesis. (See [1].) It has two parameters B and C. C is the conclusion of the current subgoal and B is the hypothesis. B starts as the H of IMPLY when HOA is first called.

```
(HOA  (λ (B  C   )   (COND
[(MATCH  B  C)]
[(EQ 'λ (CAR  B))  (AND-H)]
[(EQ 'v (CAR  B))  (OR-H EXCLUDE)]
[(EQ '→ (CAR  B))  (BACKCHARN  EXCLUDE)]
[(EQ '= (CAR  B))  (SUB = )]
[(AND (EQ '~ (CAR  B)) (NOT (ATOM(CADR  B))) (EQ '=  (CAADR  B))]
      (MATCH (CADADR  B)(CADDR( CADR  B))) )]
 )))
```

MATCH simply tries to unify B and C; if a (most general) substitution θ is found for which $B\theta \equiv C\theta$, then θ is returned. AND-H is used when B has the form $(B_1 \wedge B_2)$; it first tries proving C by B_1 and then by B_2.

OR-H treats the case when B has the form $(B_1 \vee B_2)$; this results in a split which is treated entirely similar to AND-SPLIT (below). BACKCHAIN treats the case when B has the form $(P \rightarrow Q)$; it tries to use Q to prove C, and if that is successful (with substitution θ) then it can finish the proof by proving $P\theta$. SUB= treats the case when B has the form $(a = b)$; it substitutes either a for b or b for a throughout and tries to prove the result. The final step of HOA tries to match B_1 and B_2 in case B has the form $(B_1 \neq B_2)$.

HOA-TASKS

HOA is only called when the parameter A is not 0. When $A > 0$, HOA-TASKS is called instead. Its purpose is to add a list of tasks to the (extended) AGENDA. It has two parameters B and C which are the hypothesis and conclusion of the current subgoal.

HOA-TASKS uses subroutines H-TASKS, AND-H-TASKS, BACK-C-TASKS, SUB=TASKS, and SUB=T, which are listed below.

```
(HOA-TASKS (λ (B C)  (COND
(SETQ  TASKS  (H-TASKS  B  C))  (COND
   [(EQ 'SUCCEED (CAR TASKS))  (CDR TASKS)]
   [T(SETQ  AGENDA  (APPEND  AGENDA  TASKS))
     (SETQ PRIORITY LOW FOR HOA-TASKS)
     (IMPLY) ] ) ]
[T  (SETQ PRIORITY LOW FOR HOA-TASKS)  (IMPLY)] )))
```

The purpose of HOA-TASK is to add a list of tasks to the AGENDA. It does this by calling H-TASKS. Sometimes H-TASK succeeds in actually proving the subgoal (instead of collecting tasks) and in this case HOA-TASKS returns this successful answer. But normally H-TASK returns a collection of tasks and these

are placed on the AGENDA.

H-TASK does its work by mimicking the actions of HOA, except that in a case where HOA would make a call

(1) (IMPLY ...)

to IMPLY, H-TASKS does not actually execute (1) but merely places it on a list of tasks to be put on the AGENDA.

For example, in backchaining, if $B \equiv (P(x) \rightarrow Q(x))$, $C \equiv Q(a)$, then H-TASK would (via BACK-C-TASKS) match Q(x) and Q(a) getting a/x, and place on its list the task

((IMPLY ... H Q(a) ...) 10)

which will be placed on the AGENDA later.

The reason for using HOA-TASKS instead of HOA is to avoid trying to prove the same tasks over and over. Once HOA-TASKS has acted (and is put to sleep), each of the tasks it has generated will be tried in order of their priority, and if they fail, will themselves to put to sleep or given a low priority, thereby preventing repeated attempts to perform them.

```
(H-TASKS(λ (B  C)  (COND
[(MATCH B  C) (LIST 'SUCCESS  (MATCH  A  B))]
[(EQ ' ∧(CAR B)) (AND-H-TASKS)]
[(EQ ' ∨(CAR B)) NIL]
[(AND (EQ ' → (CAR B)) (ANDS (CADDR B)C)) (BACK-C-TASKS)]
[(EQ '= (CAR B))  (SUB = TASKS)]
[(AND (EQ '~ (CAR B)) (NOT (ATOM (CADR B))) (EQ '= (CAADR B))
      (MATCH (CADAR B) (CADDR (CADR B))))
 (LIST 'SUCCESS (MATCH (CADAR B) (CADDR (CADR B))))]
)))
```

```
(AND-H-TASKS (λ (  ) (PROG (X  Y  ) (RETURN  (COND
  [(SETQ X (H-TASKS (CADR B) C))   (COND
      [(EQ 'SUCCESS (CAR X)) X ]
      [(SETQ Y (H-TASKS (CADDR B) C))   (COND
          [(EQ 'SUCCESS (CAR Y)) Y]
          [T (APPEND X Y)] ) ]
      [T X ]) ]
[T(H-TASKS (CADDR B) C)]   ))))
```

```
(BACK-C-TASKS (λ (  )
(LIST (LIST
   '(IMPLY  C = (APPLY-SIG (ANDS (CADDR B)C))
            SL = (A NEW SUBGOAL LABEL   )
    10   ))   ))
(10 is a priority; it should depend on other things)
```

```
(SUB=TASKS (λ (  )   (COND
[(EQUAL (CADR B) (CADDR B))  NIL]
[T (SUB=T  DB  (PURGE H B) C (CADR B) (CADDR B))])))
```

```
(SUB = T  (λ  (DB  H  C  L  R)  (COND
[(CHOOSESUBST  L  R )  (OR
    (AND  (TY  DB) (EQ  T (CONTRADICTION (TY  DB) NIL))
          (LIST 'SUCCESS  (ANS  T  NIL)))
    (LIST  (LIST
       '(IMPLY  SL = CHANGE LABEL  DB  H  C    )  10 ))))]
[T  NIL]   )))
```

AND-SPLIT

AND-SPLIT is an item on the standard agenda. It is called (with highest priority) when the conclusion C is a conjunction $(A \wedge B)$. This routine is very much like the AND-C of our previous answers. (See [2].)

(AND-SPLIT LL EXCLUDE) is called with two parameters LL and EXCLUDE. LL will be explained later. EXCLUDE is a list of substitution units which are not permitted in an answer from IMPLY. It is used to efficiently control backtracking when a conflict is encountered in proving each of the two conjuncts A and B. (See [2].)

We will give two versions of AND-SPLIT: Our earlier version, and the current version. In each case we assume that the conclusion C has the form $C = (A \wedge B)$, and that

$$x = (\text{IMPLY } A) = (\theta)$$

$$y = (\text{IMPLY } B) = (\lambda)$$

$$y2 = (\text{IMPLY } B\theta) = (\sigma)$$

$$E' = \text{EXCLUDE} \cup (\text{Conflict } \theta\ \sigma)$$

where (IMPLY A) means a call to IMPLY which $C = A$ and the other parameters unchanged. (Conflict θ σ) selects a substitution unit from θ which is in conflict with σ.

These algorithms for AND-SPLIT are given in tabular form, which we believe is easisr to understand than the more precise LISP version that follows.

(AND-SPLIT EXCLUDE)

(Earlier Version)

IF	RETURN
x = (NIL ...)	(NIL NIL)
y = (NIL ...)	(NIL NIL)
x = (θ ...)	
y2 = (λ ...)	(θ λ ...)
y2 = (NIL ...)	
y = (σ ...)	(AND-SPLIT E')

The idea here is simple: in proving (A ∧ B), it first trys to prove A, getting the substitution θ , and then trys to prove Bθ getting a substitution λ . If both succeed then it returns θ λ . If the proof of A succeeds but the proof of Bθ fails then it tries proving only B (with θ not applied to it). If that succeeds with σ, then θ and σ must be in conflict, and it calls AND-SPLIT again but this time with an additional item in EXCLUDE (namely (conflict θ σ), the substitution unit in θ which caused the conflict between θ and σ).

In the current version of AND-SPLIT we use additional parameters LL which (if not NIL) contains information about a previous call to AND-SPLIT for this same subgoal, and a timelimit, TIMEL. If LL is not NIL it will contain values of x, y, y2 and EXCLUDE which are to be used, instead of getting these values by calls to IMPLY as indicated in (1) above. So one of our prime concerns in our present version, is to return these values of x, y, etc., that might be used in a later call to AND-SPLIT.

(AND-SPLIT LL EXCLUDE TIMEL)

(Current Version)

x = (NIL ...) y = (NIL ...)	(NIL NIL) (NIL NIL)
x = (θ ...) y2 = (λ ...)	(COMPOSE x y2)
y2 = (NIL ...) y = (σ ...)	(AND-SPLIT LL E')*
y = (TL ...) or (UP ...) CEY ≠ NIL	(NIL CEY)**
CEY = NIL	$\binom{TL}{UP}$ x y y2 E)***

y2 = (TL ...) or (UP ...)

Is this an old value of y2 = (TL ...) or (UP ...) from LL?

OLD

y = (σ ...)	(AND-SPLIT LL E')
y = (NIL ...)	(NIL NIL)
y = (TL ...) or (UP ...)	$\binom{TL}{UP}$ x y y2 E)

NEW

A = 0 or A = 1 A > 1	$\binom{TL}{UP}$ x y y2 E)
y = (σ ...) ELSE	(AND-SPLIT LL E') (NIL NIL)

X = (TL ...) or (UP ...) CEX ≠ NIL	(NIL CEX
CEX = NIL	$\binom{TL}{UP}$ x E)

---

*Actually what we want here is the answer from (AND-SPLIT LL E') augmented by the current values of x, y, and y2.

**CEX = (CE A) Where CE is a routine which tries for a counterexample. If one is found it is returned and also stored in the URL (Universal Results List), if not NIL is returned. Similary for CEy = (CE B) and CEy2 = (CE Bθ).

***Here we use an abbreviated notation to simplify the presentation. What we want returned is "TL" (or "UP" depending on the case we are working with) to indicate that this is a time-failure, and the values of x y y2 E, properly tagged by 'x, 'y, 'y2, and 'E. Thus by (TL x y y2 E) we mean (LIST 'TL (LIST 'x x) (LIST 'y y) (LIST 'y2 y2) (LIST 'E EXCLUDE)). Similarly, for (UP x y E), etc.

Our current version of AND-SPLIT is a lot like the previous version. Again, in proving $(A \wedge B)$ it first tries to prove A, getting $x = (\theta \ldots)$, and then tries to prove B, getting $y = (\sigma \ldots)$, and $B\theta$, getting $yz = (\lambda \ldots)$. (Actually it first checks whether these values of x, y, and y2 have already been computed from an earlier call to AND-SPLIT, and are stored in LL; if so it uses those values of x, y, and y2.)

If both x and y2 succeed then it returns (Compose x y2) = $(\theta \ \lambda \ldots)$. If either x or y fails it returns (NIL NIL). If x succeeds and y2 fails and y succeeds then θ and σ must have a conflict and AND-SPLIT is again called with the (conflict θ σ) added to EXCLUDE.

If x or y fails because of timelimit or unproved, then a try is made for a counterexample.

If x succeeds and y2 has a timelimit failure or is unproved, then it acts differently depending on whether y2 is a previously proved value, taken from LL, or is a newly computed value. This feature allows the post-mortem routine to store the values of x, y, y2, and E, for possible future use.

AND-SPLIT (LISP VERSION)

```
(AND-SPLIT  (λ  (LL  EXCLUDE  TIMEL)
   (PROG (X  Y  Y2  E  XX  YY  YY2  AC  A  B  CED    U)
1  (SETQ  TIMEX  ...) (SETQ  TIMEY ...) (SETQ  TIMEY2 ...)
2  (SETQ  A  (CADR  C)) (SETQ  B  (CADDR  C))
   (RETURN  (COND
3  [(AND  LL  (EQ 'E  (CAAR  LL))
      (AND-SPLIT  (CDR  LL) (CADAR  LL)  TIMEL)]
4  [(AND  LL  (SETQ  XX  (CAR  LL)) (SETQ  X  (CDR  XX))
           (SETQ  LL  (CDR  LL))  NIL)]
5  [(AND  (NOT  XX) (SETQ  X (IMPLY ... A ... TIMEX)) NIL)]*
6  [(AND  LL  (SETQ  YY  (CAR  LL)) (SETQ  Y  (CDR YY))
           (SETQ  LL  (CDR  LL))  NIL)]
7  [(AND  (NOT YY) (SETQ  Y  (IMPLY ... B ... TIMEY)) NIL)]
8  [(AND  (SUCCEED  X)  LL (SETQ  YY2  (CAR LL)) (SETQ  Y2 (CAR YY2))
           (SETQ  LL  (CDR  LL))  NIL)]
9  [(AND  (SUCCEED  X) (NOT  LL)
           (SETQ  Y2  (IMPLY ... (APPLY-SIG  X  B) ...  TIMEY2))  NIL)]
10 [(OR  (FAIL  X) (FAIL  Y))  (ANS '(NIL  NIL))]
11 [(SUCCEED  X)  (COND
12    [(SUCCEED  Y2)  (COMPOSE  X  Y2)]
13    [(FAIL  Y2)  (COND
14       [(SUCCEED  Y)  (AND-SPLIT2)]
15       [T (CE-ANS-SPLIT  B (ANS '((CAR  Y)  X  Y  Y2  E )))])]
16    [YY2   (COND
17       [(SUCCEED  Y) (AND-SPLIT2)]
18       [(FAIL  Y)    (ANS ' (NIL  NIL))]
```

*By (IMPLY...A...TIMEX) we mean a call to IMPLY with C A, TIMEL TIMEX and the other parameters unchanged.

```
19              [T  (ANS ' ((CAR  Y)   X Y  Y2  E))])]
          [T  (COND
20              [(OR  (EQUAL  A  0) (EQUAL  A  1))
                      (ANS  ' ((CAR  Y2)  X Y  E))]
21              [(SUCCEED  Y) (AND-SPLIT2)
22              [T  (ANS  ' (NIL  NIL))])])]
23      [T  (CE-AND-SPLIT  A  (ANS  '((CAR  X)  X E  )))]    )))))
```

The LISP version of AND-SPLIT first calculates (in Step 1) timelimits that will be used in calls to IMPLY. Then in Step 3, if the first element of LL is ('E ...) then it recalls IMPLY with the rest of LL.

Steps 4-9 give values to x, y, and y2; these are taken from LL if possible, if not they are obtained by calls to IMPLY with the subgoals A, B, and Bθ respectively, where x = (θ ...).

Since the LISP version parallels the tabular version the explanation on pp. suffices for Steps 10-23. The subroutines ANS, ANS2, AND-SPLIT2 and CE-AND-SPLIT are given and explained below.

```
(ANS  (λ  (L)  (CONS  (CAR  L)  (ANS2  (CDR  L)))))
(ANS2  (λ (L)  (COND
[(NULL  L)  NIL]
[(EQ  'E  (CAR  L)) (CONS  (LIST  'E EXCLUDE) (ANS2  (CDR  L)))]
[T  (CONS  (LIST  ' (CAR  L) (EUAL  (CAR  L)) (ANS2  (CDR  L)))])))
```

ANS and its auxillary subroutine ANS2, organizes the answers from AND-SPLIT. For example (ANS '(TL X Y E)) returns ('TL ('X (Value of X)) ('Y (Value of Y)) ('E (Value of EXCLUDE))).

```
(AND-SPLITZ  (λ  ( )  (COND
[(SETQ  U  (CONFLICT  X  Y  B))
   (SETQ  E  (CONS  U  EXCLUDE))
   (SETQ  TIMEL2 ...)
   (SETQ  AC  (AND-SPLIT  LL  E  TIMEL2))
   (COND
      [(SUCCEED  AC)  AC
      [T  (APPEND  (ANS  ' ((CAR  AC)  X  Y  Y2  E)) (CDR  AC))])]
[T  (ANS  ' (NIL  NIL))]
```

B is a formula, $x = (\theta \ldots)$, $y = (\sigma \ldots)$. AND-SPLIT2 find a substitution unit u in θ which conflicts with λ in B, adds u to EXCLUDE and recalls AND-SPLIT with a new timelimit TIMEL2.

```
(CE-AND-SPLIT  (λ  (D  ANS)  (COND
[(SETQ  CED  (CE  D)) (PUT-IN  URL  (LIST  'CE  CED)) (ANS ' (NIL  CED))]
[T  ANS]     )))
```

This tries to find a counter example to the formula D. If one is returned and also found, it is stored on the Universal Results List. Otherwise ANS is returned.

To be added later:

POST-MORTEM-IMPLY

POST-MORTEM-AND-SPLIT

CE (counterexample)

LEMMAS

HEURISTICS

Handling Priority Numbers

Calculating and Assignment Timelimits.

References

1. ATP 17A UT Prover

2. ATP 40 General Inequalities

3. A Man-machine Theorem Proving System.

4. ATP 33A, MI9 Set Variables

5. ATP 43 Design for a Prover

6. ATP 41 Unskolemizing

7. Don Cohen's Prover

8. Peter Broell's Prover

9. Malcolm Tyson's Prover