A Prover for General Inequalities

by

W.W. Bledsoe, Peter Bruell,
and Robert Shostak*

February 1979 ATP-40A

This work was supported by National Science Foundation Grant MSC 77-20701 & MCF76-81025

*SRI International, Menlo Park, California

A Prover for General Inequalities

by

W.W. Bledsoe, Peter Bruell

and Robert Shostak

Abstract: A variation of the prover described in ATP-17 is used to prove theorems about general inequalities, i.e., first order logic with equality where the only predicate symbols are $\leq$, $<$, and $=$, and where function symbols are admitted. Transcripts of several proofs are given in the paper.

Table of Contents

1. Introduction

The purpose of this paper is to describe a prover which we have used to prove a number of theorems about general inequalities.

In these theorems the only predicates symbols that can appear are $\leq$, $<$, and $=$, and the logical connectives $\wedge$, $\vee$, $\rightarrow$, $\sim$. Since we allow function symbols, this class of theorems is indeed <u>all</u> of those of first order logic, because if P is an n-ary predicate symbol then the expression $P(x_1,\ldots,x_n)$ can be replaced by an equivalent expression $f(x_1,\ldots,x_n) = 0$, where f is an n-ary function symbol. So this appears to be yet another prover for the first order logic. And indeed it is.

However, it is not our intent to prove, by these methods, theorems that have been concocted artificially to the general inequality form, but rather those that arise "naturally".

Furthermore, we have tried to make use of techniques already in use which lend themselves well to inequalities. Indeed, if we considered only ground theorems, theorems in which there are no variables to be instantiated (i.e., in which all variables are universally quantified), then the recently developed techniques for Presburger Arithmetic [2, 6-8] would apply very well. We use one of these [2] as a subroutine for attempting proofs of ground formulas. We also use the "Restriction Variable" techniques of [1, Sec. 4.2], and combine these two methods with other methods from [3].

This prover is a variation of our natural deduction prover described in [3]. As explained below, some additions and changes have been made.

It should be emphasized that this prover is completely automatic (i.e., not a man-machine prover), and all the examples were proved as indicated by the machine alone.

We describe in Section 2 the principal parts of the prover, including the function IMPLY and its auxiliary subroutines. And in Section 3 we give transcripts of proofs of several examples.

We believe the best way for the reader to proceed is to read the description of IMPLY and Data-Base carefully, skim the descriptions of the auxiliary functions, and then go right to the examples in Section 3; each of the auxiliary algorithms can then be studied as it is encountered in the examples.

2. The Principal Parts

(IMPLY DB H C TL LT PV)

This is the main routine of the Prover.

DB, the data base (which is described below), is a list

(A-UNIT RESTRICTION-LIST TYPELIST)

H is a WFF, the Hypothesis of the theorem being proved

C is a WFF, the Conclusion of the theorem being proved

TL is the theorem label

LT is a control parameter

PV is a list of variables ("protected" variables) (see Section 3)

We will suppress TL and LT in the following description, though the theorem label TL is used and explained in the examples.

ALGORITHM: (IMPLY DB H C PV)

	IF	RETURN and/or ACTION	NAME
1.	(PROVED-PREVIOUSLY DB H C)	(T NIL)	PROVED PREVIOUSLY
2.	(HIGHER-SUBGOAL DB H C)	NIL	HIGHER SUBGOAL FAILURE
3.		(REDUCE)	REDUCE
3.1	C ≡ T	(T NIL)	
3.2	C ≡ NIL	NIL	
4.	C ≡ A ∧ B	(AND-C C EXCLUDE)	AND-SPLIT
5.	C ≡ A ∨ B	(IMPLY DB H ((NOTL A) → B))	OR-SPLIT
6.	C ≡ (A → B)	Put θ: = (IMPLY DB (H ∧ A) B)	PROMOTE
	θ ≠ NIL	θ	
6.1	ELSE	(IMPLY DB (H ∧ (NOTL B)) (NOTL A))	REVERSE-PROMOTE
7.	C ≡ (≤ a b)	(PROVE-LE a b ≤)	INEQUALITY-PROVER
7.1	C ≡ (< a b)	(PROVE-LE a b <)	
8.	C ≡ (a ≠ b)	Put θ: = (CHOOSE a b (DB ∧ H))	SUB=
		(IMPLY DBθ Hθ (< a b))	
9.	ELSE	NIL	

The algorithm above lists only a few of the rules of the IMPLY used in our other provers [4,3], but these are the only ones needed for our General Inequality Prover.

We repeat this limited description of IMPLY here so the reader will not have to refer to earlier papers. Also, the earlier papers do not adequately describe the workings of the data base (see below) and have no mention of some of the inequality routines PROVE-LE, MATCH-LE, etc. (although the concepts for these routines are described in [2,3,4]).

A theorem (H → C), is first skolemized and then entered into the prover by a call to IMPLY

(IMPLY NIL NIL (H → C) NIL NIL NIL).

If H ≠ NIL, a call to IMPLY Rule 3 converts it immediately to

(IMPLY NIL H C (P →) NIL NIL).

The production-like rules of IMPLY are applied to obtain a substitution θ which "satisfies" the theorem. If indeed such a θ is obtained by IMPLY, we can be assured that (H → C) is a theorem, whereas if IMPLY fails, this formula may or may not be true. That is to say, IMPLY is sound but not complete.

IMPLY, and most of the subsidiary functions, return a doubleton

(θ (A-UNIT RL TY))

where the θ is the subsitution discussed in the preceeding paragraph, and

(A-UNIT RL TY)

are results for the data base that have been obtained from the current run of IMPLY. (See Data Base below.)

IMPLY returns NIL for failure. The substitution θ has the value "T" (for true) if no substitution is needed (as in a ground proof). Otherwise, it consists of a list of substitution units

$$(t_1/x_1\ t_2/x_2 \ldots t_n/x_n)$$

where the x_i are variables which are to be replaced by the terms t_i.

If there is no data base component then IMPLY obtains

$$(\theta\ \text{NIL})\ .$$

Thus the "most true" result from IMPLY is (T NIL).

IMPLY first (in Step 1) looks at its list of proved goals to see if C has been previously proved (with the same or weaker hypotheses). It then looks on the "stack" to see if C is a subgoal of itself; if so it returns NIL. This is called a "higher subgoal failure".

In step 3 it calls REDUCE which applies a set of rewrite rules to H and C (see [5]). For example, $(x \leq x)$ is converted to "T", and $(x < x)$ is converted to NIL. Also if n and m are actual numbers with n less than m then $(n < m)$ converts to "T", etc.

Step 4 is the "and-split" which divides a conjunction into two subgoals. See AND-C below.

In Step 5 IMPLY merely converts a disjunction $(A \vee B)$ into an implication $(\sim A \rightarrow B)$. This is done in order to take advantage of several things done in Step 6, "PROMOTE". The function NOTL, which is exaplined below, is used to push the negation sign, $\sim$, to the inside of a formula.

In Step 6 it "promotes", i.e., an expression

$$(H \rightarrow (A \rightarrow B))$$

is converted to the equivalent form

$$(H \wedge A \rightarrow B) .$$

If this fails to produce a proof then in Step 6.1 it "reverse promotes" by converting to the form

$$(H \wedge \sim B \rightarrow \sim A) .$$

In both cases when the formula A (or $\sim B$) is brought over and made part of the hypothesis H, any ground conjunct of A of the form $(\leq a\ b)$ or $(< a\ b)$ is inserted into the TYPELIST of the data base, and the routine CONTRADICTION is called to see if TYPELIST then holds a contradiction. If so the proof is successfully terminated. (See descriptions Data Base and CONTRADICTION below.) (See Example 5 for an example of reverse promotion.)

In Steps 7 and 7.1 the inequality prover PROVE-LE is called. See the descriptions of it and other inequality routines below.

Step 8 is a mechanism for substituting equals. A more extensive one is used in SUB=LE and PROVE-LE-GROUND-CASE. In Step 8 we use the fact that $(a \neq b)$ in the conclusion C, is the same as $(a = b)$ in the hypothesis H. The routine CHOOSE is called to select one of a or b to substitute for the other in DB and H. A call is then made to IMPLY

$$(\text{IMPLY DB}\theta\ \ H\theta\ (< a\ b))$$

which will (through CONTRADICTION) check for a contradiction in DBθ. The reason that $(< a\ b)$ is used here is so that control will be passed back to PROVE-LE through Step 7.1.

Data Base

The data base, DB, like the other arguments of IMPLY is "dynamic" or "contextual", in that it may change as the proof progresses. It has three parts: A-UNIT, RESTRICTION-LIST (or RL), and TYPELIST (or TY).

A-UNIT is the place where information about set variables (see [1]) is stored. An entry of the form

(1) ('set' A Z P(Z))

in A-UNIT, means that the set variable A (which is to be instantiated, or bound to a value) has not yet been given a definite value but has been restricted to be a subset of the set

$$\{Z: P(Z)\} .$$

If later, A is bound to a particular value C, then C must satisfy the condition that

$$C \subseteq \{Z: P(Z)\} .$$

RESTRICTION-LIST (or RL) is the place where restrictions on variables are recorded. See below the description of the function RESTRICTION-LE, and Section 4.2 of [1]. If, for example, an entry

(2) ('int' x ($<$ a) ($\leq$ b))

is present in RL, it means that the variable x (which is to be instantiated, or bound to a value) has not yet been given a definite value but has been restricted to reside in the interval

$$a < x \leq b .$$

If later, x is bound to a particular value c, then c must satisfy this same constraint,

$$a < c \leq b \ .$$

Other restrictions like, $a \leq x \leq b$, $a \leq x < b$, etc. are possible.

TYPELIST (or TY) is the place where ground inequality information is stored. See below the description of the functions PROVE-LE-GROUND-CASE and CONTRADICTION, and also [2]. If an entry

(3) $$('int'\ x\ (\leq a)\ (\leq b))$$

is present in TY it means that one of our hypotheses is

$$a \leq x \leq b \ .$$

Notice the fundamental difference between this and the (2) of RL. In RL, x is a variable to be instantiated, whereas in (3) it is a constant. RL represents knowledge about the "solution" to their variables (i.e., the bindings for the variables), whereas TY represents given knowledge or hypothesis knowledge, that can be used to obtain the solution. Getting a contradiction in TY is desirable in that it completes the proof of the current theorem, whereas a contradiction in RL is undesirable in that it indicates a failure to find an acceptable solution (binding) for x. An entry like (2) represents a whole interval of acceptable solutions for x (provided that it can be proved that $a < b$).

On the other hand we notice the great similarity between RL and A-UNIT, because in each case a variable (x for RL, and A for A-UNIT) which is to be later bound to a value, has been restricted.

(AND-C C EXCLUDE)

Called by IMPLY.

EXCLUDE is a list of bindings a/x, b/y, etc., which are forbidden to be used by UNIFY. (It is originally set to NIL.)

$C \equiv A \wedge B$ is from IMPLY

DB and H are from IMPLY

ALGORITHM: (AND-C C EXCLUDE)

IF	RETURN and/or ACTION
	Put θ: = (IMPLY DB H A PV')*
$\theta \equiv$ NIL	NIL
ELSE	Put λ: = (IMPLY (DB-A θ DB) H Bθ PV')
$\lambda \not\equiv$ NIL	$\theta \circ \lambda$**
ELSE	Put σ: = (IMPLY DB H B PV')
$\sigma \equiv$ NIL	NIL
ELSE	Put EXCLUDE': = EXCLUDE $\cup$ (CONFLICT θ σ) (AND-C C EXCLUDE')

---

* PV' and PV need not concern us here. (See Section 3.) PV' = (PV $\cup$ CV), where PV is the current list of "protected variables", and CV is the list of variables common to both A and B.

** If θ and λ conflict on a variable occurring in PV then NIL is returned instead of $\theta \circ \lambda$. Thus a conflicting substitution is never returned by IMPLY.

AND-C handles the case when C is a conjunction $A \wedge B$. If A and B have no variable in common there is a clean split with no difficulty. But if they have one or more variables in common then the substitution θ obtained from

(1) (IMPLY DB H A)

must be applied to B, as $B\theta$, and to DB, by (DB-A θ DB), before IMPLY is called on B. That is, the second call is

(2) (IMPLY (DB-A θ DB) H $B\theta$) .

BACKTRACKING

For (2) to fail means that either

(3) (IMPLY DB H B)

has no solution, (in which case we fail) or its solution λ has a conflict with θ. For example if θ is a/x and λ is b/x, with $a \neq b$, then there is such a conflict. In such a case the function

(CONFLICT θ λ)

is called to select a conflicting entry (binding) from θ (or from λ). This conflicting entry is added to the list EXCLUDE, and a new call is made to

(AND-C EXCLUDE) .

Now in UNIFY the conflicting binding will be avoided at both steps (1) and (2), and any subgoals of them. Of course, other conflicts may arise and they too can be added to EXCLUDE. This backtracking procedure, with the use of EXCLUDE, has been used in many proofs and is exhibited in the examples below. (See the note at the end of Example 5, and the remark on backtracking at the end of Example 7.)

(PROVE-LE A B S) Abbreviation: PLE

Called by IMPLY

A and B are terms (arithmetic)

S is either '$\leq$' or '$<$'

H is the hypothesis from IMPLY

C = (S A B) is the conclusion from IMPLY

ALGORITHM: (PROVE-LE A B S)

RETURN the result obtained if not NIL, ELSE try the next.

- (LESS= (S A) (S B))
- (RESTRICTION-LE A B S)
- (UNIFY A B)
- If (GROUND C), (PROVE-LE-GROUND-CASE)
- If Not (GROUND C), (MATCH-IN-TYPELIST)
- (MATCH-LE H C NIL)
- NIL

(LESS= A B)

Called by PROVE-LE

A has the form $(\leq a)$, $(< a)$,

$(\max A_1, A_2)$, or $(\min A_1, A_2)$

B has the form $(\leq b)$, $(< b)$,

$(\max B_1, B_2)$, or $(\min B_1, B_2)$.

If $A \equiv (\leq a)$, $B \equiv (\leq b)$, this routine tries to decide whether $a \leq b$ by computing $b - a$ and comparing it with 0. Similar for $(< a)$, $(< b)$. It first handles the cases when a or b is either $+\infty$ or $-\infty$ by requiring $a \leq \infty$, $-\infty \leq b$, etc. It treates the case $A \equiv (\leq a)$, $B \equiv (\min\ (\leq b_1)\ (\leq b_2))$, by requiring (LESS= $(\leq a)$ $(\leq b_1)$) and (LESS= $(\leq a)$ $(\leq b_2)$); and similarly for other cases involving "max" and "min".

It is not necessary for a and b to be numbers. For example, it can handle the theorem $(x + 3 \leq x + 5)$ by subtracting the right side from the left and detecting that the difference (-2) is ≤ 0.

(RESTRICTION-LE A B S) Abbreviation: RLE

Called from PROVE-LE

A and B are terms,

S is '$\leq$' or '$<$'.

If A is an atom, and therefore a variable to be instantiated, and if A does not occur in the term B, then an "interval",

(1) ('int' A $-\infty$ (S B))

is placed in the restriction list, RL, of the data base. (This means that the variable A lies in the interval $(-\infty, B]$ if S $\equiv$ '$\leq$', and $(-\infty, B)$ if S = '$<$'.)

If A is already represented in the data base by an entry

(2) ('int' A a b),

then the two 'intervals' (1) and (2) are intersected to get a resulting entry

(3) ('int' A a (min b (S B))).

(In some cases the expressions like (min b (S B)) are reduced to simpler ones. For example, if b $\equiv$ ($\leq$ 5), (S B) $\equiv$ ($\leq$ 7), then (min b (S B)) is changed to ($\leq$ 5). Similarly for (max a b).)

If the resulting interval is empty then NIL is returned for RESTRICTION-LE. Thus it is necessary to check (by a call to IMPLY) that (a $\leq$ B) (or a $<$ B if S = '$<$') to insure that (3) is not empty.

Also if B is an atom (variable) and B does not occur in A, we insert the "interval",

(4) ('int' B (S A) ∞)

* also we require that A not occur in any entry (int x a b) of the restriction list RL, for which x occurs in B.

into the data base. If both A and B are atoms then both intervals (1) and (4) are inserted into the data base.

If later the variable A in (3) is instantiated with a value c, then a check must be made (by a call to IMPLY) to verify that c lies in the interval, $a \leq c \leq b$ (see examples 4, 6, and 7 below). This call to IMPLY is made with the same hypotheses that obtained when the original restriction on x was made. See Section 4.2 of [1] for a further explanation of this concept.

(PROVE-LE-GROUND-CASE) Abbreviation: GLE

Called by PROVE-LE

A and B are terms from PROVE-LE ,

S is either ' $\leq$ ' or ' $<$ ', from PROVE-LE ,

Z, obtained from CONTRADICTION, is either NIL, 'HSF', or has the form (TY, EQ).

ALGORITHM: (PROVE-LE-GROUND-CASE) GLE

IF	RETURN or ACTION
	Put Z: = (CONTRADICTION (TY DB) (S A B))*
Z ≡ 'T'	(T NIL) (SUCCESS)
Z ≡ 'HSF'	NIL (Higher subgoal failure)

(Z now has the form (TY EQ))

(TY is the updated TYPELIST)

Place TY into the Data Base

EQ ≡ NIL	NIL	(failure; but the TYPELIST has been altered)
ELSE	(SUB=LE EQ)	
	(CONTRADICTION (TY DB) NIL)	

*(TY DB) is the TYPELIST portion of the data base, DB; i.e., it is its 3rd member.

This routine, which is called when A and B are ground terms, tries to prove the inequality (S A B) (i.e., $A \leq B$ or $A < B$) by placing its negation in the TYPELIST and searching for a contradiction. Since TYPELIST is maintained as a set of ground inequalities, the contradiction (or lack of it) is obtained by standard Presburger Methods. (See [2,3,6,7].)

If CONTRADICTION returns "T", the proof is complete; if it returns 'HSF' the proof fails by a higher subgoal failure, (see CONTRADICTION); otherwise it returns a doubleton

$$Z = (TY, EQ)$$

where TY is the updated typelist and EQ is either NIL or a set of equality units. TY is now placed in the data base as a replacement for TYPELIST. If $EQ \equiv NIL$ then NIL is returned and the ground proof fails, but nevertheless, the altered form of TYPELIST remains in the data base for use when other proof methods are called from PROVE-LE. But if $EQ \not\equiv NIL$, then the routine, SUB=LE, is invoked which causes this set of equality units to be applied to H,C, and the data base, and CONTRADICTION is again called to see if a new contradiction has now appeared in the TYPELIST. (See Examples 2,3.)

(CONTRADICTION TY C)

Called from PROVE-LE-GROUND-CASE

TY is a typelist, i.e., an encoded list of ground inequality intervals. (See [3] and the section on Data Base in this paper.)

C a ground atomic expression of the form

$$(\leq a\ b) \quad \text{or} \quad (< a\ b)$$

See [2,3] for reference.

This routine takes $\sim C$, the negation of C, (i.e., $(< b\ a)$ if $C \equiv (\leq a,b)$, or $(\leq b\ a)$ if $C \equiv (< a\ b)$), and inserts it into the TYPELIST (if $C \equiv$ NIL this step is omitted), and then checks for a contradiction in TYPELIST.

TYPELIST is a list of ground "intervals",

(1) ('int' x A B)

where A and B have the form $(\leq a)$, $(< a)$, $(\max A_1\ A_2)$, or $(\min A_1, A_2)$, and where A_1 and A_2 can again have the form $(\leq a)$, etc. There is only one such "interval" for a given x.

These intervals represent hypotheses for the theorem being proved. For example,

(2) ('int' x (≤ 3) (< 7))

would represent the hypothesis

$$(3 \leq x < 7) .$$

Suppose that ~ C is

$$(\leq b\ a)\ .$$

The way it is inserted in the TYPELIST is as follows. Two new "intervals"

(3) $$('int'\ b\ (<-\infty)\ (\leq a))$$

and

(4) $$('int'\ a\ (\leq b)\ (<\infty))$$

are created, and if TYPELIST has no entry of the form

(5) $$('int'\ b\ A\ B)$$

then (3) is simply placed in TYPELIST. But if it already had an entry (5) then the intersection of (3) and (5) is placed in TYPELIST. Similarly for the placement of (4).

After the insertion of ~ C, CONTRADICTION looks for a contradiction in TYPELIST. If one is found the proof is complete, and "T" is returned.

If TYPELIST experienced no change by the insertion of ~ C, this means

$$(TYPELIST \rightarrow \sim C)$$

is true, and hence that

$$(TYPELIST \rightarrow C)$$

is not, so the routine returns 'HSF' to indicate this failure.

If, on the other hand, TYPELIST is changed by the insertion of ~ C, this changed value is returned as TY, the first entry of Z = (TY, EQ). EQ, the other value of Z, is the set of equality units, if any, that can be inferred from TYPELIST.

(MATCH-IN-TYPELIST CC TY) Abbreviation: MTY

Called from PROVE-LE

CC is of the form $(\leq a\ b)$ or $(< a\ b)$, and is not ground

TY is a typelist, i.e., an encoded list of ground inequality intervals.

This routine tries to find a match for CC from the unencoded entries of TY, and returns a substitution σ for such a match if one is found.

(MATCH-LE A C D) Abbreviation: MLE

Called by PROVE-LE

A is a WFF, whose only predicates are $\leq$ and $<$. The initial value of A is the hypothesis H from IMPLY. A can have the form $(A_1 \wedge A_2)$, $(A_1 \vee A_2)$, $(A_1 \rightarrow A_2)$, $(\leq a_1\ a_2)$, or $(< a_1\ a_2)$.

C is an atomic formula of the form $(\leq c_1\ c_2)$ or $(< c_1\ c_2)$. Its initial value is the conclusion C from IMPLY.

D is also a WFF. It starts as NIL.

The result is a substitution σ or NIL if failure.

ALGORITHM: (MATCH-LE A C D)

	IF	RETURN and/or ACTION
1.	$A \equiv$ ATOM	NIL
2. 2.1	$A \equiv (A_1 \wedge A_2)$	(MATCH-LE A_1 C D), if not NIL, ELSE (MATCH-LE A_2 C D)
3.	$A \equiv (A_1 \vee A_2)$	(MATCH-LE $(\sim A_1 \rightarrow A_2)$ C D)*.
4. 4.1	$A \equiv (A_1 \rightarrow A_2)$	(MATCH-LE A_2 C $(A_1 \wedge D)$), if not NIL, ELSE (MATCH-LE $\sim A_1$ C $(\sim A_2 \wedge D)$).
5.	$A \equiv (\leq a_1\ a_2)$ or $A \equiv (< a_1\ a_2)$	PUT σ: (UNIFY $(a_1\ a_2)$ $(c_1\ c_2)$). where $C \equiv (\leq c_1\ c_2)$ or $C \equiv (< c_1\ c_2)$.
6.	$\sigma \equiv$ NIL	NIL
6.1	$C \equiv (\leq c_1\ c_2)$	(Append σ (IMPLY H Dσ))
6.2	$C \equiv (< c_1\ c_2)$	PUT D: = $(D \wedge (c_1\sigma \neq c_2\sigma))$, (Append σ (IMPLY H D)).

*In every case the "not" symbol "$\sim$" is pushed to the inside by the routine NOTL.

This routine tries to prove the implication $(A \rightarrow C)$ by matching A, or a part of A, against C.

If A has the form $(A_1 \rightarrow A_2)$ then the routine "backchains" by trying to prove C from A_2 and storing A_1 in D to be proved later by IMPLY. If that fails it trys again with $(\sim A_2 \rightarrow \sim A_1)$.

Backchaining also takes place when A has the form $(A_1 \vee A_2)$, in which case $(A_1 \vee A_2)$ is treated like $(\sim A_1 \rightarrow A_2)$.

The process of backchaining is repeated in cases such as when A has the form $(A_1 \rightarrow (A_{21} \rightarrow A_{22}))$. Then A_{22} is used to prove C, and $(A_1 \wedge A_{21})$ is stored in D to be proved later, etc.

In Step 5, when two expressions are unified, any binding of a variable x must be checked against a possible restriction on x in the Restriction list, RL, of the data base. (See Data Base.) For example, if UNIFY selects the binding c/x and if RL has an entry

$$('int'\ x\ (\leq a)\ (< b))$$

then before the binding c/x is allowed, it must first check that

$$a \leq c < b\ .$$

This is done by a call to IMPLY. (See Examples 4,6,7 below.)

If A has the form $(\leq a_1\ a_2)$ or $(< a_1\ a_2)$, then the routine attempts to unify $(a_1\ a_2)$ and $(c_1\ c_2)$, obtaining a substitution σ. <u>But</u> if C has the form $(< c_1\ c_2)$ and A has the form $(\leq a_1\ a_2)$, (i.e., it is trying to prove an implication of the form

$$(a_1 \leq a_2 \rightarrow c_1 < c_2) \qquad),$$

then the routine stores in D the inequality $(c_1\sigma \neq c_2\sigma)$ to be proved later by Rule 8 of IMPLY. (This will have the effect of reproving of the theorem $(H \rightarrow C)$ with the additional hypothesis $(c_1\sigma = c_2\sigma)$.)