File created: 30-Sep-86 14:09:43 **{ERIS}<LISPUSERS>LISPCORE>BOYERMOORE.;1**

changes to: (VARS BASISCOMS CODE-S-ZCOMS)
(FNS BM-UPCASE BM-PRIN1 BM-NTH BM-COUNT CREATE-EVENT BM-NEGATE BM-PPR BM-REDUCE BM-SUBST MAKE-LIB)
(FUNCTIONS BM-MATCH)

previous date: 6-Jul-86 10:30:08 {ERIS}<LISPUSERS>KOTO>BOYERMOORE.;5

Read Table: OLD-INTERLISP-FILE

Package: INTERLISP

Format: XCCS

;;


```
(RPAQQ BOYERMOORECOMS
       [(* * The Boyer Moore Theorem Prover -- By Boyer and Moore -- Translated from Zetalisp to Interlisp-D by
           Kelly Roach. *)
        (COMS (* My personal hacks to BOYERMOORE. *)
              (INITVARS (DEBUGFLG T))
              (FNS UNDEFN UNPROVE-LEMMA))
        (COMS * BASISCOMS)
        (COMS * CODE-1-ACOMS)
        (COMS * CODE-B-DCOMS)
        (COMS * CODE-E-MCOMS)
        (COMS * CODE-N-RCOMS)
        (COMS * CODE-S-ZCOMS)
        (COMS * EVENTSCOMS)
        (COMS * GENFACTCOMS)
        (COMS * IOCOMS)
        (COMS * PPRCOMS)
        (FILES COMPILEBANG)
        (DECLARE: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS
               (ADDVARS (NLAMA TOGGLE REFLECT PROVE-LEMMA ENABLE DISABLE DEFN DCL ADD-SHELL ADD-AXIOM)
                      (NLAML)
                      (LAMA])
```

(* * The Boyer Moore Theorem Prover -- By Boyer and Moore --
Translated from Zetalisp to Interlisp-D by Kelly Roach. *)

(* * My personal hacks to BOYERMOORE. *)

```
(RPAQ? DEBUGFLG T)

(DEFINEQ
```

(UNDEFN

```
  [LAMBDA (NAME)                                                    (* kbr: " 6-Jul-86 09:46")
                                                                    (* A personal hack. Take back a BOYERMOORE defined
                                                                    function. *)
    (PROG (*1*NAME)
          (SETQ PROVED-THMS (for THM in PROVED-THMS when (NOT (AND (EQ (CAR THM)
                                                                       (QUOTE DEFN))
                                                                   (EQ (CADR THM)
                                                                       NAME)))
                                  collect THM))
          (SETQ *1*NAME (PACK* "*1*" NAME))
          (SETQ LIB-ATOMS-WITH-PROPS (DREMOVE NAME LIB-ATOMS-WITH-PROPS))
          (SETQ LIB-ATOMS-WITH-PROPS (DREMOVE *1*NAME LIB-ATOMS-WITH-PROPS))
          (SETQ LIB-ATOMS-WITH-DEFS (DREMOVE NAME LIB-ATOMS-WITH-DEFS))
          (SETQ CHRONOLOGY (DREMOVE NAME CHRONOLOGY))
          (for PROP in LIB-PROPS do (REMPROP NAME PROP))
          (for PROP in LIB-PROPS do (REMPROP *1*NAME PROP))
          (REMPROP *1*NAME (QUOTE EXPR))
          (PUTD *1*NAME NIL])
```

(UNPROVE-LEMMA

```
  [LAMBDA (NAME)                                                    (* kbr: " 6-Jul-86 09:47")
                                                                    (* A personal hack. Take back a BOYERMOORE lemma *)
    (PROG (TERM)
          [SETQ TERM (CADDDR (GETPROP NAME (QUOTE EVENT]
          [for X in (UNPRETTYIFY TERM) do (SETQ CONCL (CDR X))
                                          (COND
                                             ((GETPROP (TOP-FNNAME CONCL)
                                                       (QUOTE LEMMAS))
                                              (PUTPROP (TOP-FNNAME CONCL)
                                                       (QUOTE LEMMAS)
                                                       (for RULE in (GETPROP (TOP-FNNAME CONCL)
                                                                             (QUOTE LEMMAS))
                                                          when (NOT (EQ (fetch (REWRITE-RULE NAME) of OLD-RULE)
                                                                        NAME))
                                                          collect RULE]
```

```
          (SETQ PROVED-THMS (for THM in PROVED-THMS when (NOT (EQ TERM THM)) collect THM))
          (SETQ LIB-ATOMS-WITH-PROPS (DREMOVE NAME LIB-ATOMS-WITH-PROPS))
          (SETQ CHRONOLOGY (DREMOVE NAME CHRONOLOGY))
          (SETQ PROCESS-HIST (DREMOVE NAME PROCESS-HIST))
          (for PROP in LIB-PROPS do (REMPROP NAME PROP])

)

(RPAQQ BASISCOMS
       [(* BASIS *)
        (FUNCTIONS GET1 *1*IF ADD-SUB-FACT-BODY ACCESS ARGN BINDINGS CHANGE DISABLEDP FARGN FARGS FCONS-TERM
               FCONS-TERM* FFN-SYMB FN-SYMB FQUOTEP LOGBIT LOGDIFF BM-MATCH MATCH! NVARIABLEP PQUOTE PRIND
               QUOTEP TYO1 TYPE-PRESCRIPTION VALUEP VARIABLEP)
        [VARS (THEOREM-PROVER-FILES (QUOTE (BASIS GENFACT EVENTS CODE-1-A CODE-B-D CODE-E-M CODE-N-R CODE-S-Z IO
                                                 BM-PPR)))
              (ALPHABETIC-CASE-AFFECTS-STRING-COMPARISON T)
              (PROVE-FILE NIL)
              (ADD-TERM-TO-POT-LST-TEMP (LIST NIL))
              (ALL-LEMMAS-USED NIL)
              (ALMOST-SUBSUMES-CONSTANT (CONS NIL NIL))
              (ANCESTORS NIL)
              (ARITY-ALIST NIL)
              (BOOK-SYNTAX-FLG NIL)
              BOOT-STRAP-INSTRS
              (BOOT-STRAP-MACRO-FNS (QUOTE (GREATERP LEQ GEQ)))
              (BROKEN-LEMMAS NIL)
              (CLAUSE-ALIST NIL)
              (COMMUTED-EQUALITY-FLG NIL)
              (CULPRIT-FUNCTION NIL)
              (CURRENT-ATM 0)
              (CURRENT-LIT 0)
              (DO-NOT-USE-INDUCTION-FLG NIL)
              (DOTCONS (LIST NIL NIL))
              (ELIM-VARIABLE-NAMES (QUOTE (X Z V W D C X1 Z1 V1 W1 D1 C1 X2 Z2 V2 W2 D2 C2)))
              (ELIM-VARIABLE-NAMES1 NIL)
              (EXECUTE-PROCESSES (QUOTE (SIMPLIFY-CLAUSE SETTLED-DOWN-CLAUSE FERTILIZE-CLAUSE
                                                 ELIMINATE-DESTRUCTORS-CLAUSE GENERALIZE-CLAUSE
                                                 ELIMINATE-IRRELEVANCE-CLAUSE STORE-SENT)))
              (EXPAND-LST NIL)
              (FAILED-THMS NIL)
              (FAILURE-MSG "************** F A I L E D **************")
              (FALSE (QUOTE (QUOTE *1*FALSE)))
              (FNS-TO-BE-IGNORED-BY-REWRITE NIL)
              (FORCEIN 38)
              (GEN-VARIABLE-NAMES (QUOTE (Y A U B E G H P Q R S)))
              (GENERALIZE-LEMMA-NAMES NIL)
              (GENERALIZING-SKOS NIL)
              (HEURISTIC-TYPE-ALIST NIL)
              (HINT NIL)
              [HINT-VARIABLE-ALIST (QUOTE ((DISABLE TEMPORARILY-DISABLED-LEMMAS NIL NIL)
                                           (EXPAND HINTED-EXPANSIONS T NIL)
                                           (HANDS-OFF FNS-TO-BE-IGNORED-BY-REWRITE NIL NIL)
                                           (NO-BUILT-IN-ARITH NO-BUILT-IN-ARITH-FLG NIL NIL]
              (HINTED-EXPANSIONS NIL)
              (INDUCTION-HYP-TERMS NIL)
              (IN-ADD-AXIOM-FLG NIL)
              (IN-BOOT-STRAP-FLG NIL)
              (IN-REDO-UNDONE-EVENTS-FLG NIL)
              (IN-PROVE-LEMMA-FLG NIL)
              (IO-FN (QUOTE IO1))
              (IOTHMTIME 0)
              (IPOSITION-ALIST NIL)
              (LAST-PRINEVAL-CHAR (QUOTE %.))
              (LAST-PROCESS NIL)
              (LEFTMARGINCHAR NIL)
              (LEMMA-DISPLAY-FLG NIL)
              (LEMMA-TYPES (QUOTE (REWRITE ELIM GENERALIZE META)))
              (LITATOM-FORM-COUNT-ALIST NIL)
              (LITS-THAT-MAY-BE-ASSUMED-FALSE NIL)
              (LITS-TO-BE-IGNORED-BY-LINEAR NIL)
              (META-NAMES (QUOTE (APPLY MEANING MEANING-LST ARITY FORMP FORM-LSTP)))
              (MUST-BE-FALSE NIL)
              (MUST-BE-TRUE NIL)
              (NILCONS (CONS NIL NIL))
              (NO-BUILT-IN-ARITH-FLG NIL)
              (OBVIOUS-RESTRICTIONS NIL)
              (ORIGEVENT NIL)
              [PPR-MACRO-LST (QUOTE ((NOT . CONVERT-NOT)
                                     (CONS . CONVERT-CONS)
                                     (CAR . CONVERT-CAR-CDR)
                                     (CDR . CONVERT-CAR-CDR)
                                     (QUOTE . CONVERT-QUOTE]
              (PPRFIRSTCOL 35)
              (PPRMAXLNS 10000)
              (PRINEVAL-FNS (QUOTE (IEQP AND EQUAL OR NOT EQ EQLENGTH !CLAUSE !CLAUSE-SET !PPR LENGTH
                                         LENGTH-TO-ATOM !PPR-LIST !LIST PLURALP QUOTE QUOTE PQUOTE CAR CDR
                                         FN-SYMB FFN-SYMB ARGN FARGN ARGS FARGS QUOTEP FQUOTEP)))
```

```
              (PROVED-THMS NIL)
              (R-ALIST NIL)
              (STACK NIL)
              (TAB-SIZE 8.0)
              (TEMPORARILY-DISABLED-LEMMAS NIL)
              (TERMS-TO-BE-IGNORED-BY-REWRITE NIL)
              (TRANSLATE-TO-LISP-TIME 0)
              (TRUE (QUOTE (QUOTE *1*TRUE)))
              (TRUE-CLAUSE (LIST TRUE))
              (TRUE-TYPE-ALIST NIL)
              (TTY-FILE NIL)
              (TYPE-ALIST NIL)
              (UN-PRODUCTIVE-PROCESSES (QUOTE (SETTLED-DOWN-CLAUSE STORE-SENT POP SUBSUMED-ABOVE
                                                   SUBSUMED-BY-PARENT SUBSUMED-BELOW FINISHED)))
              (UNDONE-BATCH-COMMANDS NIL)
              (UNDONE-EVENTS-STACK NIL)
              (USE-NO-LEMMAS-FLG NIL)
              (WELL-ORDERING-RELATIONS (QUOTE (LESSP LEX2 LEX3)))
              (ZERO (QUOTE (QUOTE 0]
       (CONSTANTS (EVENT-SEPARATOR-STRING (CHARACTER (CHARCODE CR)))
              (*1*F (QUOTE *1*FALSE))
              (*1*SHELL-QUOTE-MARK (QUOTE *1*QUOTE))
              (*1*T (QUOTE *1*TRUE))
              (PARAGRAPH-INDENT 5)
              (STRING-WEIRD (QUOTE *1*))
              (STRING-WEIRD2 (QUOTE *2*))
              (STRING-WEIRD3 (QUOTE *3*))
              (TREE-INDENT 2)
              (TREE-LINES 2)
              (TYPE-SET-BOOLEAN 3)
              (TYPE-SET-CONS 16)
              (TYPE-SET-FALSE 1)
              (TYPE-SET-LITATOMS 8)
              (TYPE-SET-NEGATIVES 32)
              (TYPE-SET-NUMBERS 4)
              (TYPE-SET-TRUE 2)
              (TYPE-SET-UNKNOWN -1))
       (INITVARS (LIB-FILE NIL)
              (LIB-VARS NIL)
              (LIB-ATOMS-WITH-PROPS NIL)
              (LIB-ATOMS-WITH-DEFS NIL)
              (LIB-PROPS NIL)
              (*ALIST* NIL)
              (*ARGLIST* NIL)
              (*CONTROLLER-COMPLEXITIES* NIL)
              (*FILE* NIL)
              (*FNNAME* NIL)
              (*INDENT* 0)
              (*TYPE-ALIST* NIL)
              (*1*BTM-OBJECTS NIL)
              (ABBREVIATIONS-USED NIL)
              (ADD-EQUATIONS-TO-DO NIL)
              (ALIST NIL)
              (ALISTS NIL)
              (ALL-FNS-FLG NIL)
              (ALMOST-SUBSUMES-LITERAL NIL)
              (ANS NIL)
              (ARGS NIL)
              (CHRONOLOGY NIL)
              (CL2 NIL)
              (COMMONSUBTERMS NIL)
              (CURRENT-CL NIL)
              (CURRENT-SIMPLIFY-CL NIL)
              (CURRENT-TYPE-NO NIL)
              (DECISIONS NIL)
              (DEFINITELY-FALSE NIL)
              (DEFN-FLG NIL)
              (DESCENDANTS NIL)
              (DISABLED-LEMMAS NIL)
              (DLHDFMLA NIL)
              (ELAPSEDTHMTIME NIL)
              (ENDLIST NIL)
              (EVENT-LST NIL)
              (FAILURE-ACTION NIL)
              (FALSE-TYPE-ALIST NIL)
              (FILE NIL)
              (FLATSIZE NIL)
              (FMLA NIL)
              (FNS NIL)
              (FNSTACK NIL)
              (FORM NIL)
              (GEN-VARIABLE-NAMES1 NIL)
              (GENERALIZE-LEMMAS NIL)
              (GENRLTLIST NIL)
              (HIGHER-PROPS NIL)
              (HINTS NIL)
              (HIST-ENTRY NIL)
```

```
               (ID-IFF NIL)
               (INDENT NIL)
               (INDUCTION-CONCL-TERMS NIL)
               (INST-HYP NIL)
               (LAST-CLAUSE NIL)
               (LAST-EXIT NIL)
               (LAST-HYP NIL)
               (LAST-PRIN5-WORD NIL)
               (LAST-PRINT-CLAUSES NIL)
               (LINEARIZE-ASSUMPTIONS-STACK NIL)
               (LEMMA-STACK NIL)
               (LEMMAS-USED-BY-LINEAR NIL)
               (LINEAR-ASSUMPTIONS NIL)
               (MAIN-EVENT-NAME NIL)
               (MARG2 NIL)
               (MASTER-ROOT-NAME NIL)
               (MATCH-TEMP NIL)
               (MATCH-X NIL)
               (MINREM NIL)
               (NAME NIL)
               (NAMES NIL)
               (NEXT-MEMO-KEY NIL)
               (NEXT-MEMO-VAL NIL)
               (NEXTIND NIL)
               (NEXTNODE NIL)
               (NONCONSTRUCTIVE-AXIOM-NAMES NIL)
               (NUMBER-OF-VARIABLES NIL)
               (OBJECTIVE NIL)
               (ORIG-LEMMA-STACK NIL)
               (ORIG-LINEARIZE-ASSUMPTIONS-STACK NIL)
               (ORIGTHM NIL)
               (PARENT NIL)
               (PARENT-HIST NIL)
               (POS NIL)
               (PPR-MACRO-MEMO NIL)
               (PPRFILE NIL)
               (PROCESS NIL)
               (PROCESS-CLAUSES NIL)
               (PROCESS-HIST NIL)
               (PROP NIL)
               (PROPLIST NIL)
               (PROVE-TERMINATION-LEMMAS-USED NIL)
               (RECOGNIZER-ALIST NIL)
               (RECORD-DECLARATIONS NIL)
               (RECORD-TEMP NIL)
               (RELIEVE-HYPS-NOT-OK-ANS NIL)
               (REMAINDER NIL)
               (SCRIBE-FLG NIL)
               (SETQ-LST NIL)
               (SHELL-ALIST NIL)
               (SHELL-POCKETS NIL)
               (SIMPLIFY-CLAUSE-MAXIMALLY-CLAUSES NIL)
               (SIMPLIFY-CLAUSE-MAXIMALLY-HIST NIL)
               (SIMPLIFY-CLAUSE-POT-LST NIL)
               (SINGLETON-TYPE-SETS NIL)
               (SPACELEFT NIL)
               (STARTLIST NIL)
               (T2 NIL)
               (TEMP-TEMP NIL)
               (TEMP1 NIL)
               (TEST-LST NIL)
               (THM NIL)
               (TYPE-SET-TERM1 NIL)
               (UNDONE-EVENTS NIL)
               (UNIFY-SUBST NIL)
               (UNIVERSE NIL)
               (VAL NIL)
               (VAR-ALIST NIL))
         (RECORDS CANDIDATE GENERALIZE-LEMMA JUSTIFICATION LINEAR-LEMMA LINEAR-POT MEASURE-RULE POLY REWRITE-RULE
                TESTS-AND-ALISTS TESTS-AND-CASE TESTS-AND-CASES TYPE-PRESCRIPTION-NAME-AND-PAIR TYPE-RESTRICTION)
         (FNS BM-UPCASE COMPILE-IF-APPROPRIATE-AND-POSSIBLE COPYLIST EXTEND-FILE-NAME FIND-CHAR-IN-FILE
              FIND-STRING-IN-FILE GET-TOTAL-STATS GET-FROM-FILE GET-PLIST-FROM-FILE GET-STATS-FILE BM-PRIN1
              PRINT-SYSTEM PRINT-DATE-LINE RANDOM-INITIALIZATION RANDOM-NUMBER READ-FILE REMQ STORE-DEFINITION
              SWAP-OUT R-LOOP TIME-IT TIME-IN-60THS XSEARCH *1*CAR *1*CDR ADD-TO-SET ARGN-MACRO BINDINGS-MACRO
              CELL CREATE-LEMMA-STACK CREATE-LINEARIZE-ASSUMPTIONS-STACK CREATE-STACK1 FARGN-MACRO FN-SYMB-MACRO
              HLOAD IPOSITION ITERPRI ITERPRIN ITERPRISPACES IPRIN1 IPRINC IPRINT ISPACES KILL-DEFINITION LINEL
              MAKE-LIB MATCH-MACRO MATCH!-MACRO MATCH1-MACRO MATCH2-MACRO NOTE-LIB BM-NTH PREPARE-FOR-THE-NIGHT
              SPELL-NUMBER SUB-PAIR UNIONQ)
         (P (SETQ LEMMA-STACK (CREATE-LEMMA-STACK 10))
            (SETQ LINEARIZE-ASSUMPTIONS-STACK (CREATE-LINEARIZE-ASSUMPTIONS-STACK 10])


           (* * BASIS *)

(DEFMACRO GET1 (ATM PROP)
   #M
```

```
   (BQUOTE (GET11 (\, ATM)
                  (\, PROP)))
   #Q
   (BQUOTE (GETPROP (\, ATM)
                  (\, PROP))))


(DEFMACRO *1*IF (X Y Z)
   [BQUOTE (COND
              ((EQ (\, X)
                   *1*F)
               (\, Z))
              (T (\, Y])


(DEFMACRO ADD-SUB-FACT-BODY X
   (GENERATE-ADD-SUB-FACT1 X))


(DEFMACRO ACCESS X
   (ACCESS-MACRO (CAR X)
          (CADR X)
          (CADDR X)))


(DEFMACRO ARGN TAIL
   (ARGN-MACRO TAIL))


(DEFMACRO BINDINGS TAIL
   (BINDINGS-MACRO TAIL))


(DEFMACRO CHANGE X
   (CHANGE-MACRO (CAR X)
          (CADR X)
          (CADDR X)
          (CADDDR X)))


(DEFMACRO DISABLEDP (NAME)
   [BQUOTE (OR (MEMB (SETQ TEMP-TEMP (\, NAME))
                     TEMPORARILY-DISABLED-LEMMAS)
               (CDDR (ASSOC TEMP-TEMP DISABLED-LEMMAS])


(DEFMACRO FARGN TAIL
   (FARGN-MACRO TAIL))


(DEFMACRO FARGS (X)
   (BQUOTE (CDR (\, X))))


(DEFMACRO FCONS-TERM TAIL
   (CONS (QUOTE CONS)
         TAIL))


(DEFMACRO FCONS-TERM* TAIL
   (CONS (QUOTE LIST)
         TAIL))


(DEFMACRO FFN-SYMB (X)
   (BQUOTE (CAR (\, X))))


(DEFMACRO FN-SYMB TAIL
   (FN-SYMB-MACRO TAIL))


(DEFMACRO FQUOTEP (X)
   (BQUOTE (EQ (CAR (\, X))
               (QUOTE QUOTE))))


(DEFMACRO LOGBIT (N)
   (BQUOTE (LSH 1 (\, N))))


(DEFMACRO LOGDIFF (X Y)
   (BQUOTE (BOOLE 4 (\, X)
                  (\, Y))))


(DEFMACRO BM-MATCH X
```

```
        (* Matchs FORM against PATTERN where X = (FORM PATTERN) FORM is evaluated and the free variables in PATTERN
        are bound to parts of FORM. (BM-MATCH (QUOTE (EQUAL 34 56))
        (EQUAL XX YY)) returns T and causes XX and YY to be bound to 34 and 56 *)

   (MATCH-MACRO (CAR X)
          (CADR X)))


(DEFMACRO MATCH! X
   (MATCH!-MACRO (CAR X)
          (CADR X)))


(DEFMACRO NVARIABLEP (X)
   (BQUOTE (LISTP (\, X))))


(DEFMACRO PQUOTE (X)
   (BQUOTE (QUOTE (\, X))))


(DEFMACRO PRIND (X FILE)
   [BQUOTE (LET ((TEMP (\, X)))
                (PRIN1 TEMP (\, FILE))
                (SETQ POS (IPLUS POS (NCHARS TEMP])


(DEFMACRO QUOTEP (X)
   [BQUOTE (AND (LISTP (SETQ TEMP-TEMP (\, X)))
                (EQ (CAR TEMP-TEMP)
                    (QUOTE QUOTE])


(DEFMACRO TYO1 (X FILE)
   [BQUOTE (PROGN (TYO (\, X)
                       (\, FILE))
                  (SETQ POS (ADD1 POS])


(DEFMACRO TYPE-PRESCRIPTION (X)
   [BQUOTE (CDAR (GETPROP (\, X)
                          (QUOTE TYPE-PRESCRIPTION-LST])


(DEFMACRO VALUEP (X)
   (BQUOTE (QUOTEP (\, X))))


(DEFMACRO VARIABLEP (X)
   (BQUOTE (NLISTP (\, X))))

(RPAQQ THEOREM-PROVER-FILES (BASIS GENFACT EVENTS CODE-1-A CODE-B-D CODE-E-M CODE-N-R CODE-S-Z IO BM-PPR))

(RPAQQ ALPHABETIC-CASE-AFFECTS-STRING-COMPARISON T)

(RPAQQ PROVE-FILE NIL)

(RPAQ ADD-TERM-TO-POT-LST-TEMP (LIST NIL))

(RPAQQ ALL-LEMMAS-USED NIL)

(RPAQ ALMOST-SUBSUMES-CONSTANT (CONS NIL NIL))

(RPAQQ ANCESTORS NIL)

(RPAQQ ARITY-ALIST NIL)

(RPAQQ BOOK-SYNTAX-FLG NIL)

(RPAQQ BOOT-STRAP-INSTRS
       ((ADD-SHELL0 FALSE NIL FALSEP NIL)
        (ADD-SHELL0 TRUE NIL TRUEP NIL)
        (DEFN0 NOT (P)
               (IF P (FALSE)
                   (TRUE))
               NIL T)
        (DEFN0 AND (P Q)
               (IF P (IF Q (TRUE)
                           (FALSE))
                   (FALSE))
               NIL T)
        (DEFN0 OR (P Q)
               (IF P (TRUE)
                   (IF Q (TRUE)
                       (FALSE)))
               NIL T)
```

```
        (DEFN0 IMPLIES (P Q)
               (IF P (IF Q (TRUE)
                           (FALSE))
                   (TRUE))
               NIL T)
        (ADD-SHELL0 ADD1 ZERO NUMBERP ((SUB1 (ONE-OF NUMBERP)
                                             ZERO)))
        (DEFN0 LESSP (X Y)
               [IF (OR (EQUAL Y 0)
                       (NOT (NUMBERP Y)))
                   (FALSE)
                   (IF (OR (EQUAL X 0)
                           (NOT (NUMBERP X)))
                       (TRUE)
                       (LESSP (SUB1 X)
                              (SUB1 Y]
               NIL T)
        (PUT1 LESSP 0 LEVEL-NO)
        (DEFN0 GREATERP (X Y)
               (LESSP Y X)
               NIL NIL)
        (DEFN0 LEQ (X Y)
               (NOT (LESSP Y X))
               NIL NIL)
        (DEFN0 GEQ (X Y)
               (NOT (LESSP X Y))
               NIL NIL)
        (DEFN0 LEX2 (L1 L2)
               [OR (LESSP (CAR L1)
                          (CAR L2))
                   (AND (EQUAL (CAR L1)
                               (CAR L2))
                        (LESSP (CADR L1)
                               (CADR L2]
               NIL NIL)
        (DEFN0 LEX3 (L1 L2)
               [OR (LESSP (CAR L1)
                          (CAR L2))
                   (AND (EQUAL (CAR L1)
                               (CAR L2))
                        (LEX2 (CDR L1)
                              (CDR L2]
               NIL NIL)
        (DEFN0 ZEROP (X)
               (IF (EQUAL X 0)
                   T
                   (IF (NUMBERP X)
                       F T))
               NIL T)
        (DEFN0 FIX (X)
               (IF (NUMBERP X)
                   X 0)
               NIL T)
        (DEFN0 PLUS (X Y)
               (IF (ZEROP X)
                   (FIX Y)
                   (ADD1 (PLUS (SUB1 X)
                               Y)))
               NIL T)
        (ADD-AXIOM1 COUNT-NUMBERP (REWRITE)
               (IMPLIES (NUMBERP I)
                        (EQUAL (COUNT I)
                               I)))
        (ADD-AXIOM1 COUNT-NOT-LESSP (REWRITE)
               (NOT (LESSP (COUNT I)
                           I)))
        (ADD-SHELL0 PACK NIL LITATOM ((UNPACK (NONE-OF)
                                              ZERO)))
        (ADD-SHELL0 CONS NIL LISTP ((CAR (NONE-OF)
                                         ZERO)
                                    (CDR (NONE-OF)
                                         ZERO)))
        (DEFN0 NLISTP (X)
               (NOT (LISTP X))
               NIL T)
        (ADD-SHELL0 MINUS NIL NEGATIVEP ((NEGATIVE-GUTS (ONE-OF NUMBERP)
                                                        ZERO)))
        (DEFN0 DIFFERENCE (I J)
               [IF (ZEROP I)
                   0
                   (IF (ZEROP J)
                       I
                       (DIFFERENCE (SUB1 I)
                                   (SUB1 J]
               NIL T)
        (DEFN0 TIMES (I J)
               (IF (ZEROP I)
```

```
                0
                (PLUS J (TIMES (SUB1 I)
                               J)))
        NIL T)
(DEFN0 QUOTIENT (I J)
        [IF (ZEROP J)
            0
            (IF (LESSP I J)
                0
                (ADD1 (QUOTIENT (DIFFERENCE I J)
                                J]
        NIL T)
(DEFN0 REMAINDER (I J)
        (IF (ZEROP J)
            (FIX I)
            (IF (LESSP I J)
                (FIX I)
                (REMAINDER (DIFFERENCE I J)
                       J)))
        NIL T)
(DEFN0 LEGAL-CHAR-CODES NIL
        (QUOTE (45 48 49 50 51 52 53 54 55 56 57 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
                   84 85 86 87 88 89 90))
        NIL NIL)
(DEFN0 ILLEGAL-FIRST-CHAR-CODES NIL (QUOTE (45 48 49 50 51 52 53 54 55 56 57))
        NIL NIL)
(DEFN0 LENGTH (LST)
        (IF (LISTP LST)
            (ADD1 (LENGTH (CDR LST)))
            0)
        NIL NIL)
(DEFN0 MEMBER (X LST)
        [IF (NLISTP LST)
            F
            (IF (EQUAL X (CAR LST))
                T
                (MEMBER X (CDR LST]
        NIL NIL)
(DEFN0 SUBSETP (X Y)
        (IF (NLISTP X)
            T
            (IF (MEMBER (CAR X)
                        Y)
                (SUBSETP (CDR X)
                        Y)
                F))
        NIL NIL)
(DEFN0 LAST (L)
        (IF (LISTP L)
            (IF (LISTP (CDR L))
                (LAST (CDR L))
                L)
            L)
        NIL NIL)
(DEFN0 LEGAL-CHAR-CODE-SEQ (LST)
        (AND (LISTP LST)
             (SUBSETP LST (LEGAL-CHAR-CODES))
             (NOT (MEMBER (CAR LST)
                          (ILLEGAL-FIRST-CHAR-CODES)))
             (EQUAL (CDR (LAST LST))
                    0))
        NIL NIL)
(DEFN0 SYMBOLP (X)
        (AND (LITATOM X)
             (LEGAL-CHAR-CODE-SEQ (UNPACK X)))
        NIL NIL)
(DEFN0 LOOKUP (X ALIST)
        [IF (NLISTP ALIST)
            NIL
            (IF (AND (LISTP (CAR ALIST))
                     (EQUAL X (CAAR ALIST)))
                (CDAR ALIST)
                (LOOKUP X (CDR ALIST]
        NIL NIL)
(DCL0 ARITY (X))
(DCL0 FORMP (X))
(DEFN0 FORM-LSTP (X)
        [IF (NLISTP X)
            (EQUAL X NIL)
            (AND (FORMP (CAR X))
                 (FORM-LSTP (CDR X]
        NIL NIL)
(DCL0 APPLY (X LST))
(DCL0 MEANING (X ALIST))
(DEFN0 MEANING-LST (X ALIST)
        (IF (NLISTP X)
            NIL
```

```
                  (CONS (MEANING (CAR X)
                                 ALIST)
                        (MEANING-LST (CDR X)
                                     ALIST)))
                NIL NIL)
        (SETUP-META-NAMES)
        (DEFN0 SPLIT (X)
               X NIL NIL)
        (DEFN0 CHECK (X)
               X NIL NIL)))
```

(RPAQQ **BOOT-STRAP-MACRO-FNS** (GREATERP LEQ GEQ))

(RPAQQ **BROKEN-LEMMAS** NIL)

(RPAQQ **CLAUSE-ALIST** NIL)

(RPAQQ **COMMUTED-EQUALITY-FLG** NIL)

(RPAQQ **CULPRIT-FUNCTION** NIL)

(RPAQQ **CURRENT-ATM** 0)

(RPAQQ **CURRENT-LIT** 0)

(RPAQQ **DO-NOT-USE-INDUCTION-FLG** NIL)

(RPAQ **DOTCONS** (LIST NIL NIL))

(RPAQQ **ELIM-VARIABLE-NAMES** (X Z V W D C X1 Z1 V1 W1 D1 C1 X2 Z2 V2 W2 D2 C2))

(RPAQQ **ELIM-VARIABLE-NAMES1** NIL)

(RPAQQ **EXECUTE-PROCESSES** (SIMPLIFY-CLAUSE SETTLED-DOWN-CLAUSE FERTILIZE-CLAUSE ELIMINATE-DESTRUCTORS-CLAUSE
GENERALIZE-CLAUSE ELIMINATE-IRRELEVANCE-CLAUSE STORE-SENT))

(RPAQQ **EXPAND-LST** NIL)

(RPAQQ **FAILED-THMS** NIL)

(RPAQ **FAILURE-MSG** "********** F A I L E D **********")

(RPAQQ **FALSE** (QUOTE *1*FALSE))

(RPAQQ **FNS-TO-BE-IGNORED-BY-REWRITE** NIL)

(RPAQQ **FORCEIN** 38)

(RPAQQ **GEN-VARIABLE-NAMES** (Y A U B E G H P Q R S))

(RPAQQ **GENERALIZE-LEMMA-NAMES** NIL)

(RPAQQ **GENERALIZING-SKOS** NIL)

(RPAQQ **HEURISTIC-TYPE-ALIST** NIL)

(RPAQQ **HINT** NIL)

```
(RPAQQ HINT-VARIABLE-ALIST ((DISABLE TEMPORARILY-DISABLED-LEMMAS NIL NIL)
                            (EXPAND HINTED-EXPANSIONS T NIL)
                            (HANDS-OFF FNS-TO-BE-IGNORED-BY-REWRITE NIL NIL)
                            (NO-BUILT-IN-ARITH NO-BUILT-IN-ARITH-FLG NIL NIL)))
```

(RPAQQ **HINTED-EXPANSIONS** NIL)

(RPAQQ **INDUCTION-HYP-TERMS** NIL)

(RPAQQ **IN-ADD-AXIOM-FLG** NIL)

(RPAQQ **IN-BOOT-STRAP-FLG** NIL)

(RPAQQ **IN-REDO-UNDONE-EVENTS-FLG** NIL)

(RPAQQ **IN-PROVE-LEMMA-FLG** NIL)

(RPAQQ **IO-FN** IO1)

(RPAQQ **IOTHMTIME** 0)

(RPAQQ **IPOSITION-ALIST** NIL)

(RPAQQ **LAST-PRINEVAL-CHAR** %.)

(RPAQQ **LAST-PROCESS** NIL)

(RPAQQ **LEFTMARGINCHAR** NIL)

```
(RPAQQ LEMMA-DISPLAY-FLG NIL)

(RPAQQ LEMMA-TYPES (REWRITE ELIM GENERALIZE META))

(RPAQQ LITATOM-FORM-COUNT-ALIST NIL)

(RPAQQ LITS-THAT-MAY-BE-ASSUMED-FALSE NIL)

(RPAQQ LITS-TO-BE-IGNORED-BY-LINEAR NIL)

(RPAQQ META-NAMES (APPLY MEANING MEANING-LST ARITY FORMP FORM-LSTP))

(RPAQQ MUST-BE-FALSE NIL)

(RPAQQ MUST-BE-TRUE NIL)

(RPAQ NILCONS (CONS NIL NIL))

(RPAQQ NO-BUILT-IN-ARITH-FLG NIL)

(RPAQQ OBVIOUS-RESTRICTIONS NIL)

(RPAQQ ORIGEVENT NIL)

(RPAQQ PPR-MACRO-LST ((NOT . CONVERT-NOT)
                      (CONS . CONVERT-CONS)
                      (CAR . CONVERT-CAR-CDR)
                      (CDR . CONVERT-CAR-CDR)
                      (QUOTE . CONVERT-QUOTE)))

(RPAQQ PPRFIRSTCOL 35)

(RPAQQ PPRMAXLNS 10000)

(RPAQQ PRINEVAL-FNS (IEQP AND EQUAL OR NOT EQ EQLENGTH !CLAUSE !CLAUSE-SET !PPR LENGTH LENGTH-TO-ATOM !PPR-LIST
                          !LIST PLURALP QUOTE QUOTE PQUOTE CAR CDR FN-SYMB FFN-SYMB ARGN FARGN ARGS FARGS QUOTEP
                          FQUOTEP))

(RPAQQ PROVED-THMS NIL)

(RPAQQ R-ALIST NIL)

(RPAQQ STACK NIL)

(RPAQQ TAB-SIZE 8.0)

(RPAQQ TEMPORARILY-DISABLED-LEMMAS NIL)

(RPAQQ TERMS-TO-BE-IGNORED-BY-REWRITE NIL)

(RPAQQ TRANSLATE-TO-LISP-TIME 0)

(RPAQQ TRUE (QUOTE *1*TRUE))

(RPAQ TRUE-CLAUSE (LIST TRUE))

(RPAQQ TRUE-TYPE-ALIST NIL)

(RPAQQ TTY-FILE NIL)

(RPAQQ TYPE-ALIST NIL)

(RPAQQ UN-PRODUCTIVE-PROCESSES (SETTLED-DOWN-CLAUSE STORE-SENT POP SUBSUMED-ABOVE SUBSUMED-BY-PARENT
                                    SUBSUMED-BELOW FINISHED))

(RPAQQ UNDONE-BATCH-COMMANDS NIL)

(RPAQQ UNDONE-EVENTS-STACK NIL)

(RPAQQ USE-NO-LEMMAS-FLG NIL)

(RPAQQ WELL-ORDERING-RELATIONS (LESSP LEX2 LEX3))

(RPAQQ ZERO (QUOTE 0))

(DECLARE: EVAL@COMPILE

(RPAQ EVENT-SEPARATOR-STRING (CHARACTER (CHARCODE CR)))

(RPAQQ *1*F *1*FALSE)

(RPAQQ *1*SHELL-QUOTE-MARK *1*QUOTE)

(RPAQQ *1*T *1*TRUE)

(RPAQQ PARAGRAPH-INDENT 5)
```

```
(RPAQQ STRING-WEIRD *1*)

(RPAQQ STRING-WEIRD2 *2*)

(RPAQQ STRING-WEIRD3 *3*)

(RPAQQ TREE-INDENT 2)

(RPAQQ TREE-LINES 2)

(RPAQQ TYPE-SET-BOOLEAN 3)

(RPAQQ TYPE-SET-CONS 16)

(RPAQQ TYPE-SET-FALSE 1)

(RPAQQ TYPE-SET-LITATOMS 8)

(RPAQQ TYPE-SET-NEGATIVES 32)

(RPAQQ TYPE-SET-NUMBERS 4)

(RPAQQ TYPE-SET-TRUE 2)

(RPAQQ TYPE-SET-UNKNOWN -1)

(CONSTANTS (EVENT-SEPARATOR-STRING (CHARACTER (CHARCODE CR)))
       (*1*F (QUOTE *1*FALSE))
       (*1*SHELL-QUOTE-MARK (QUOTE *1*QUOTE))
       (*1*T (QUOTE *1*TRUE))
       (PARAGRAPH-INDENT 5)
       (STRING-WEIRD (QUOTE *1*))
       (STRING-WEIRD2 (QUOTE *2*))
       (STRING-WEIRD3 (QUOTE *3*))
       (TREE-INDENT 2)
       (TREE-LINES 2)
       (TYPE-SET-BOOLEAN 3)
       (TYPE-SET-CONS 16)
       (TYPE-SET-FALSE 1)
       (TYPE-SET-LITATOMS 8)
       (TYPE-SET-NEGATIVES 32)
       (TYPE-SET-NUMBERS 4)
       (TYPE-SET-TRUE 2)
       (TYPE-SET-UNKNOWN -1))
)

(RPAQ? LIB-FILE NIL)

(RPAQ? LIB-VARS NIL)

(RPAQ? LIB-ATOMS-WITH-PROPS NIL)

(RPAQ? LIB-ATOMS-WITH-DEFS NIL)

(RPAQ? LIB-PROPS NIL)

(RPAQ? *ALIST* NIL)

(RPAQ? *ARGLIST* NIL)

(RPAQ? *CONTROLLER-COMPLEXITIES* NIL)

(RPAQ? *FILE* NIL)

(RPAQ? *FNNAME* NIL)

(RPAQ? *INDENT* 0)

(RPAQ? *TYPE-ALIST* NIL)

(RPAQ? *1*BTM-OBJECTS NIL)

(RPAQ? ABBREVIATIONS-USED NIL)

(RPAQ? ADD-EQUATIONS-TO-DO NIL)

(RPAQ? ALIST NIL)

(RPAQ? ALISTS NIL)

(RPAQ? ALL-FNS-FLG NIL)

(RPAQ? ALMOST-SUBSUMES-LITERAL NIL)

(RPAQ? ANS NIL)

(RPAQ? ARGS NIL)
```

(RPAQ? **CHRONOLOGY** NIL)

(RPAQ? **CL2** NIL)

(RPAQ? **COMMONSUBTERMS** NIL)

(RPAQ? **CURRENT-CL** NIL)

(RPAQ? **CURRENT-SIMPLIFY-CL** NIL)

(RPAQ? **CURRENT-TYPE-NO** NIL)

(RPAQ? **DECISIONS** NIL)

(RPAQ? **DEFINITELY-FALSE** NIL)

(RPAQ? **DEFN-FLG** NIL)

(RPAQ? **DESCENDANTS** NIL)

(RPAQ? **DISABLED-LEMMAS** NIL)

(RPAQ? **DLHDFMLA** NIL)

(RPAQ? **ELAPSEDTHMTIME** NIL)

(RPAQ? **ENDLIST** NIL)

(RPAQ? **EVENT-LST** NIL)

(RPAQ? **FAILURE-ACTION** NIL)

(RPAQ? **FALSE-TYPE-ALIST** NIL)

(RPAQ? **FILE** NIL)

(RPAQ? **FLATSIZE** NIL)

(RPAQ? **FMLA** NIL)

(RPAQ? **FNS** NIL)

(RPAQ? **FNSTACK** NIL)

(RPAQ? **FORM** NIL)

(RPAQ? **GEN-VARIABLE-NAMES1** NIL)

(RPAQ? **GENERALIZE-LEMMAS** NIL)

(RPAQ? **GENRLTLIST** NIL)

(RPAQ? **HIGHER-PROPS** NIL)

(RPAQ? **HINTS** NIL)

(RPAQ? **HIST-ENTRY** NIL)

(RPAQ? **ID-IFF** NIL)

(RPAQ? **INDENT** NIL)

(RPAQ? **INDUCTION-CONCL-TERMS** NIL)

(RPAQ? **INST-HYP** NIL)

(RPAQ? **LAST-CLAUSE** NIL)

(RPAQ? **LAST-EXIT** NIL)

(RPAQ? **LAST-HYP** NIL)

(RPAQ? **LAST-PRIN5-WORD** NIL)

(RPAQ? **LAST-PRINT-CLAUSES** NIL)

(RPAQ? **LINEARIZE-ASSUMPTIONS-STACK** NIL)

(RPAQ? **LEMMA-STACK** NIL)

(RPAQ? **LEMMAS-USED-BY-LINEAR** NIL)

(RPAQ? **LINEAR-ASSUMPTIONS** NIL)

(RPAQ? **MAIN-EVENT-NAME** NIL)

(RPAQ? **MARG2** NIL)

(RPAQ? **MASTER-ROOT-NAME** NIL)

(RPAQ? **MATCH-TEMP** NIL)

(RPAQ? **MATCH-X** NIL)

(RPAQ? **MINREM** NIL)

(RPAQ? **NAME** NIL)

(RPAQ? **NAMES** NIL)

(RPAQ? **NEXT-MEMO-KEY** NIL)

(RPAQ? **NEXT-MEMO-VAL** NIL)

(RPAQ? **NEXTIND** NIL)

(RPAQ? **NEXTNODE** NIL)

(RPAQ? **NONCONSTRUCTIVE-AXIOM-NAMES** NIL)

(RPAQ? **NUMBER-OF-VARIABLES** NIL)

(RPAQ? **OBJECTIVE** NIL)

(RPAQ? **ORIG-LEMMA-STACK** NIL)

(RPAQ? **ORIG-LINEARIZE-ASSUMPTIONS-STACK** NIL)

(RPAQ? **ORIGTHM** NIL)

(RPAQ? **PARENT** NIL)

(RPAQ? **PARENT-HIST** NIL)

(RPAQ? **POS** NIL)

(RPAQ? **PPR-MACRO-MEMO** NIL)

(RPAQ? **PPRFILE** NIL)

(RPAQ? **PROCESS** NIL)

(RPAQ? **PROCESS-CLAUSES** NIL)

(RPAQ? **PROCESS-HIST** NIL)

(RPAQ? **PROP** NIL)

(RPAQ? **PROPLIST** NIL)

(RPAQ? **PROVE-TERMINATION-LEMMAS-USED** NIL)

(RPAQ? **RECOGNIZER-ALIST** NIL)

(RPAQ? **RECORD-DECLARATIONS** NIL)

(RPAQ? **RECORD-TEMP** NIL)

(RPAQ? **RELIEVE-HYPS-NOT-OK-ANS** NIL)

(RPAQ? **REMAINDER** NIL)

(RPAQ? **SCRIBE-FLG** NIL)

(RPAQ? **SETQ-LST** NIL)

(RPAQ? **SHELL-ALIST** NIL)

(RPAQ? **SHELL-POCKETS** NIL)

(RPAQ? **SIMPLIFY-CLAUSE-MAXIMALLY-CLAUSES** NIL)

(RPAQ? **SIMPLIFY-CLAUSE-MAXIMALLY-HIST** NIL)

(RPAQ? **SIMPLIFY-CLAUSE-POT-LST** NIL)

(RPAQ? **SINGLETON-TYPE-SETS** NIL)

(RPAQ? **SPACELEFT** NIL)

(RPAQ? **STARTLIST** NIL)

(RPAQ? **T2** NIL)

```
(RPAQ? TEMP-TEMP NIL)

(RPAQ? TEMP1 NIL)

(RPAQ? TEST-LST NIL)

(RPAQ? THM NIL)

(RPAQ? TYPE-SET-TERM1 NIL)

(RPAQ? UNDONE-EVENTS NIL)

(RPAQ? UNIFY-SUBST NIL)

(RPAQ? UNIVERSE NIL)

(RPAQ? VAL NIL)

(RPAQ? VAR-ALIST NIL)

(DECLARE: EVAL@COMPILE

(DATATYPE CANDIDATE (SCORE CONTROLLERS CHANGED-VARS UNCHANGEABLE-VARS TESTS-AND-ALISTS-LST JUSTIFICATION
                           INDUCTION-TERM OTHER-TERMS))

(DATATYPE GENERALIZE-LEMMA (NAME TERM))

(DATATYPE JUSTIFICATION (SUBSET MEASURE-TERM RELATION LEMMAS))

(DATATYPE LINEAR-LEMMA (NAME HYPS CONCL MAX-TERM))

(DATATYPE LINEAR-POT (VAR POSITIVES NEGATIVES))

(DATATYPE MEASURE-RULE (CONDITION-LIST THE-LESSER STRENGTH-SIGN THE-GREATER INDUCTION-LEMMA-NAME MEASURE))

(DATATYPE POLY (CONSTANT ALIST ASSUMPTIONS LITERALS LEMMAS))

(DATATYPE REWRITE-RULE (NAME HYPS CONCL LOOP-STOPPER))

(DATATYPE TESTS-AND-ALISTS (TESTS ALISTS))

(DATATYPE TESTS-AND-CASE (TESTS CASE))

(DATATYPE TESTS-AND-CASES (TESTS CASES))

(DATATYPE TYPE-PRESCRIPTION-NAME-AND-PAIR (NAME PAIR))

(DATATYPE TYPE-RESTRICTION (TERM TYPE-SET DEFAULT))
)

(/DECLAREDATATYPE (QUOTE CANDIDATE)
       (QUOTE (POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER))
       ;; ---field descriptor list elided by lister---
       (QUOTE 16))

(/DECLAREDATATYPE (QUOTE GENERALIZE-LEMMA)
       (QUOTE (POINTER POINTER))
       ;; ---field descriptor list elided by lister---
       (QUOTE 4))

(/DECLAREDATATYPE (QUOTE JUSTIFICATION)
       (QUOTE (POINTER POINTER POINTER POINTER))
       ;; ---field descriptor list elided by lister---
       (QUOTE 8))

(/DECLAREDATATYPE (QUOTE LINEAR-LEMMA)
       (QUOTE (POINTER POINTER POINTER POINTER))
       ;; ---field descriptor list elided by lister---
       (QUOTE 8))

(/DECLAREDATATYPE (QUOTE LINEAR-POT)
       (QUOTE (POINTER POINTER POINTER))
       ;; ---field descriptor list elided by lister---
       (QUOTE 6))

(/DECLAREDATATYPE (QUOTE MEASURE-RULE)
       (QUOTE (POINTER POINTER POINTER POINTER POINTER POINTER))
       ;; ---field descriptor list elided by lister---
       (QUOTE 12))
```

```
(/DECLAREDATATYPE (QUOTE POLY)
       (QUOTE (POINTER POINTER POINTER POINTER POINTER))
       ;; ---field descriptor list elided by lister---
       (QUOTE 10))

(/DECLAREDATATYPE (QUOTE REWRITE-RULE)
       (QUOTE (POINTER POINTER POINTER POINTER))
       ;; ---field descriptor list elided by lister---
       (QUOTE 8))

(/DECLAREDATATYPE (QUOTE TESTS-AND-ALISTS)
       (QUOTE (POINTER POINTER))
       ;; ---field descriptor list elided by lister---
       (QUOTE 4))

(/DECLAREDATATYPE (QUOTE TESTS-AND-CASE)
       (QUOTE (POINTER POINTER))
       ;; ---field descriptor list elided by lister---
       (QUOTE 4))

(/DECLAREDATATYPE (QUOTE TESTS-AND-CASES)
       (QUOTE (POINTER POINTER))
       ;; ---field descriptor list elided by lister---
       (QUOTE 4))

(/DECLAREDATATYPE (QUOTE TYPE-PRESCRIPTION-NAME-AND-PAIR)
       (QUOTE (POINTER POINTER))
       ;; ---field descriptor list elided by lister---
       (QUOTE 4))

(/DECLAREDATATYPE (QUOTE TYPE-RESTRICTION)
       (QUOTE (POINTER POINTER POINTER))
       ;; ---field descriptor list elided by lister---
       (QUOTE 6))

(DEFINEQ
```

(BM-UPCASE

```
  [LAMBDA (N)                                              (* kbr: "19-Oct-85 16:31")
    (COND
       ((AND (IGEQ N (CHARCODE a))
             (ILEQ N (CHARCODE z)))
        (IDIFFERENCE N 32))
       (T N])
```

(COMPILE-IF-APPROPRIATE-AND-POSSIBLE

```
  [LAMBDA (FNS)                                            (* kbr: " 6-Jul-86 09:53")
```

(* If a function foo is defined in our theory, a function *1*foo is defined in Lisp.
Sometimes during the course of a proof, *1*foo may be executed to compute the value of foo on certain values.
There is a speed benefit to compiling *1*foo. In Maclisp, the compiler is not in the same Lisp with the theorem-prover in Zetalisp, the compiler is resident. The *.LISP files produced by MAKE-LIB may be compiled after loading the compilation of BASIS into the compiler. Hence it is possible to obtain the speed of compiled functions in the Maclisp version of the theorem-prover, at the expense of making a library, running a separate compilation, and using NOTE-LIB to load the *.LIB file and the compilation of the .LISP file. *)

```
    (for FN in FNS do (COND
                        ((NOT (CCODEP (GETD FN)))
                         (COMPILE! FN])
```

(COPYLIST

```
  [LAMBDA (L)                                              (* kbr: " 6-Jul-86 09:54")
                                                           (* Top level copy of list L *)
    (for X in L collect X])
```

(EXTEND-FILE-NAME

```
  [LAMBDA (FILE EXTENSION)                                 (* kbr: "19-Oct-85 16:31")
    (PACKFILENAME (APPEND (LIST (QUOTE EXTENSION)
                                EXTENSION)
                          (UNPACKFILENAME FILE])
```

(FIND-CHAR-IN-FILE

```
  [LAMBDA (CHAR FILE)                                      (* kbr: "25-Oct-85 14:33")
```

(* Assumes that FILE is a stream for a file. Searches for the next occurrence of CHAR past current position, if any.
If one is found, the file pointer is left just after the occurrence and the file pointer is returned.
Otherwise NIL is returned. *)

```
    (while T bind CH do (SETQ CH (TYI FILE -1))
                        (COND
                           ((IEQP CH -1)
                            (RETURN NIL))
                           ((IEQP CH CHAR)
                            (RETURN (FILEPOS FILE])
```

(FIND-STRING-IN-FILE

```
  [LAMBDA (STRING FILE)                                    (* kbr: "25-Oct-85 14:34")
    (LET [(STRING-LEN-1 (SUB1 (NCHARS STRING]
         (COND
            ((EQUAL STRING-LEN-1 -1)
             (GETFILEPTR FILE))
            (T (bind (POS _ (CHCON STRING))
                     (CHARS _ (CHCON STRING))
                     (FIRST-CHAR _ (CAR CHARS))
                     (OTHER-CHARS _ (CDR CHARS))
                     (*1*+FILE-LEN-STR-LEN _ (IPLUS (GETEOFPTR FILE)
                                                    (MINUS STRING-LEN-1)))
               while (SETQ POS (FIND-CHAR-IN-FILE FIRST-CHAR FILE))
               do (COND
                     ([AND (NOT (IGREATERP POS *1*+FILE-LEN-STR-LEN))
                           (for CHAR in OTHER-CHARS always (IEQP CHAR (TYI FILE]
                      (RETURN (SUB1 POS)))
                     (T (SETFILEPTR FILE POS])
```

(GET-TOTAL-STATS

```
  [LAMBDA (DIR)                                            (* kbr: "25-Oct-85 14:39")
    (PROG (TP-TIME IO-TIME STATS)
          (for ROOT in (QUOTE (PROVEALL RSA WILSON GAUSS FORTRAN CONTROLLER PR TMI UNSOLV ZTAK))
             do [SETQ STATS (SUM-STATS-ALIST (GET-STATS-FILE (PACKFILENAME (LIST (QUOTE DIRECTORY)
                                                                                 DIR
                                                                                 (QUOTE NAME)
                                                                                 ROOT
                                                                                 (QUOTE EXTENSION)
                                                                                 (QUOTE PROOFS]
                 (SETQ TP-TIME (IPLUS (CAR STATS)
                                      TP-TIME))
                 (SETQ IO-TIME (IPLUS (CADR STATS)
                                      IO-TIME)))
          (RETURN (LIST TP-TIME IO-TIME])
```

(GET-FROM-FILE

```
  [LAMBDA (ATM PROP)                                       (* kbr: "25-Oct-85 14:39")
    (for TAIL on (GET-PLIST-FROM-FILE ATM) by (QUOTE CDDR) when (EQ PROP (CAR TAIL))
       do (RETURN (CADR TAIL])
```

(GET-PLIST-FROM-FILE

```
  [LAMBDA (ATM)                                            (* kbr: " 6-Jul-86 09:57")
    (LET [(LOC (GETPROP ATM (QUOTE LIB-LOC]
         (COND
            ((NULL LOC)
             NIL)
            ((NOT (BOUNDP (QUOTE LIB-FILE)))
             NIL)
            (T (SETFILEPTR LIB-FILE LOC)
               (CADR (CADDR (READ LIB-FILE])
```

(GET-STATS-FILE

```
  [LAMBDA (FILE)                                           (* kbr: "25-Oct-85 14:41")

          (* Returns a list of triplets (event cpu io), where cpu is the number of elapsed seconds minus io seconds.
          *)

    (LET ((EVENT-CHAR (NTHCHARCODE EVENT-SEPARATOR-STRING 1))
          (EOF-CONS (CONS NIL NIL))
          TEMP TP-TIME IO-TIME)
         (SETQ FILE (OPENSTREAM FILE (QUOTE INPUT)))
         (SETFILEPTR FILE 0)
         (while [AND (FIND-CHAR-IN-FILE EVENT-CHAR FILE)
                     (NEQ EOF-CONS (SETQ TEMP (READ FILE EOF-CONS)))
                     (FIND-CHAR-IN-FILE #/ FILE)
                     (NUMBERP (SETQ TP-TIME (READ FILE EOF-CONS)))
                     (NUMBERP (SETQ IO-TIME (READ FILE EOF-CONS]
           collect (CONS TEMP (LIST TP-TIME IO-TIME])
```

(BM-PRIN1

```
  [LAMBDA (DATA FILE)                                      (* kbr: "19-Oct-85 16:31")
    (PATOM DATA FILE])
```

(PRINT-SYSTEM

```
[LAMBDA (FILE)                                                    (* kbr: "24-Oct-85 16:32")
  (PRIN1 (QUOTE SYSTEM)
         FILE)
  (TERPRI FILE)
  (PRIN1 "0.0 0.0" FILE)
  [for FL in THEOREM-PROVER-FILES do (PRINT (CDR (CAR (GETPROP FL (QUOTE FILEDATES]
  (PRIN1 MAKESYSNAME FILE)
  (PRIN1 " " FILE)
  (PRIN1 MAKESYSDATE FILE)
  (TERPRI FILE])
```

(PRINT-DATE-LINE

```
[LAMBDA NIL                                                       (* kbr: "19-Oct-85 16:31")
  (PRIN1 (GDATE)
         PROVE-FILE])
```

(RANDOM-INITIALIZATION

```
[LAMBDA (EVENT)                                                   (* kbr: "19-Oct-85 16:31")
  (RANDSET (QUOTE (14106 39422 64835 57733 34919 5315 12110 15116 10133 10816 60097 23088 5624 21340 53921
                         15106 4684 41739 64042 23754 16272 2980 45652 52345 2362 36658 65286 43318 47346
                         39405 37667 44583 35376 23651 7908 33877 32302 22146 63687 45438 51385 11636 43707
                         59964 45045 48385 64260 37257 4475 14893 14979 48212 48490 22654 29015])
```

(RANDOM-NUMBER

```
[LAMBDA (N)                                                       (* kbr: "24-Oct-85 17:15")
  (RAND 0 (SUB1 N])
```

(READ-FILE

```
[LAMBDA (FILE-NAME)                                               (* kbr: "25-Oct-85 14:42")
  (LET ((FILE (OPENSTREAM FILE-NAME (QUOTE INPUT)))
        (MY-CONS (CONS NIL NIL)))
       (bind TEMP while (NEQ MY-CONS (SETQ TEMP (READ FILE MY-CONS))) collect TEMP])
```

(REMQ

```
[LAMBDA (I L)                                                     (* kbr: "19-Oct-85 16:31")
  (for X in L unless (EQ X I) collect X])
```

(STORE-DEFINITION

```
[LAMBDA (ATM EXPR)                                                (* kbr: " 6-Jul-86 10:05")
  (PUTD ATM EXPR)
  (COMPILE! ATM])
```

(SWAP-OUT

```
[LAMBDA (NAME)                                                    (* kbr: " 3-Nov-85 17:04")
  (APPLY (FUNCTION NOTE-LIB)
         (MAKE-LIB NAME])
```

(R-LOOP

```
[LAMBDA NIL                                                       (* kbr: "20-Oct-85 17:25")
  (while T do (TERPRI NIL)
              (PRIN2 (QUOTE *)
                     NIL)
              (BM-PPR (R (PROG1 (READ NIL)
                                (TERPRI NIL)))
                     NIL])
```

(TIME-IT

```
[LAMBDA (FORM)                                                    (* kbr: "19-Oct-85 16:31")
  (LET ((START-TIME (TIME-IN-60THS)))
       (LIST (EVAL FORM)
             (QUOTIENT (DIFFERENCE (TIME-IN-60THS)
                                   START-TIME)
                     60.0])
```

(TIME-IN-60THS

```
[LAMBDA NIL                                                       (* kbr: " 6-Jul-86 10:10")
  (IQUOTIENT (ITIMES 60 (CLOCK 2))
         1000])
```

(XSEARCH

```
[LAMBDA (STRINGS FILE-SPECS)                                      (* kbr: "25-Oct-85 14:28")
  [COND
     ((NLISTP STRINGS)
      (SETQ STRINGS (LIST STRINGS]
```

```
    [COND
       ((NLISTP FILE-SPECS)
        (SETQ FILE-SPECS (LIST FILE-SPECS]
    (for L in (for FILE-SPEC in FILE-SPECS join (LIST (LIST FILE-SPEC))) bind NAME FILE
       when [AND (CAR L)
                 (PROGN (TERPRI T)
                        (SETQ NAME (CAR L))
                        (PRIN1 NAME T)
                        (SETQ FILE (OPENSTREAM NAME (QUOTE INPUT)))
                        (PROG1 (for STRING in STRINGS thereis (PROGN (SETFILEPTR FILE 0)
                                                                     (FIND-STRING-IN-FILE STRING FILE)))
                               (CLOSE? FILE]
       collect (PROGN (PRIN1 "Yes." T)
                      NAME])
```

(*1*CAR

```
  [LAMBDA (X1)                                                    (* kbr: "19-Oct-85 16:31")
    (COND
       ((NLISTP X1)
        0)
       ((EQ (CAR X1)
            (QUOTE *1*QUOTE))
        0)
       (T (CAR X1])
```

(*1*CDR

```
  [LAMBDA (X1)                                                    (* kbr: "19-Oct-85 16:31")
    (COND
       ((NLISTP X1)
        0)
       ((EQ (CAR X1)
            (QUOTE *1*QUOTE))
        0)
       (T (CDR X1])
```

(ADD-TO-SET

```
  [LAMBDA (X Y)                                                   (* kbr: "19-Oct-85 16:31")
    (COND
       ((MEMBER X Y)
        Y)
       (T (CONS X Y])
```

(ARGN-MACRO

```
  [LAMBDA (TAIL)                                                  (* kbr: "19-Oct-85 16:31")
    (COND
       [(FIXP (CADR TAIL))
        (SUB-PAIR (QUOTE (TERM CELL N))
               (LIST (CAR TAIL)
                     (CELL (CADR TAIL)
                           (QUOTE TEMP-TEMP))
                     (CADR TAIL))
               (QUOTE (COND ((NEQ (CAR (SETQ TEMP-TEMP TERM))
                                  (QUOTE QUOTE))
                             (CAR CELL))
                            (T (ARGN0 TEMP-TEMP N]
       (T (CONS (QUOTE ARGN0)
                TAIL])
```

(BINDINGS-MACRO

```
  [LAMBDA (TAIL)                                                  (* kbr: "19-Oct-85 16:31")
    (COND
       ((NLISTP TAIL)
        NIL)
       (T (BQUOTE (CONS (CONS (\, (CAR TAIL))
                              (\, (CADR TAIL)))
                        (\, (BINDINGS-MACRO (CDDR TAIL])
```

(CELL

```
  [LAMBDA (N FIELD)                                               (* kbr: "19-Oct-85 16:31")
    (COND
       ((IEQP N 0)
        FIELD)
       (T (LIST (QUOTE CDR)
                (CELL (SUB1 N)
                      FIELD])
```

(CREATE-LEMMA-STACK

```
  [LAMBDA (N)                                                     (* kbr: "19-Oct-85 21:55")
    (SETQ ORIG-LEMMA-STACK (SETQ LEMMA-STACK NIL))
    NIL])
```

(CREATE-LINEARIZE-ASSUMPTIONS-STACK

```
[LAMBDA (N)                                                    (* kbr: "19-Oct-85 22:00")
  (SETQ ORIG-LINEARIZE-ASSUMPTIONS-STACK (SETQ LINEARIZE-ASSUMPTIONS-STACK NIL))
  NIL])
```

(CREATE-STACK1

```
[LAMBDA (N)                                                    (* kbr: "19-Oct-85 16:31")
  (LET (STK)
       (SETQ STK (for I from 1 to (ITIMES 2 N) collect NIL))
       (for TAIL on STK by (QUOTE CDDR) until (NULL (CDDR TAIL)) do (RPLACA (CDDDR TAIL)
                                                                           TAIL))
       STK])
```

(FARGN-MACRO

```
[LAMBDA (TAIL)                                                 (* kbr: "20-Oct-85 13:40")
                                                               (* (BM-NTH (CADR TAIL) (CAR TAIL)) *)
  (COND
     [(FIXP (CADR TAIL))
      (LIST (QUOTE CAR)
            (CELL (CADR TAIL)
                  (CAR TAIL]
     (T (LIST (QUOTE BM-NTH)
              (CADR TAIL)
              (CAR TAIL])
```

(FN-SYMB-MACRO

```
[LAMBDA (TAIL)                                                 (* kbr: "19-Oct-85 16:31")
  (SUBST (CAR TAIL)
         (QUOTE TERM)
         (QUOTE (COND ((NEQ (QUOTE QUOTE)
                            (CAR (SETQ TEMP-TEMP TERM)))
                       (CAR TEMP-TEMP))
                      (T (FN-SYMB0 (CADR TEMP-TEMP])
```

(HLOAD

```
[LAMBDA (FILE)                                                 (* kbr: " 6-Jul-86 10:16")
                                                               (* Horrible LOAD. *)
  (PROG (STREAM EXPR)
        (SETQ STREAM (OPENSTREAM FILE (QUOTE INPUT)))
        (until (EQ (SETQ EXPR (HREAD STREAM))
                   (QUOTE STOP))
           do (EVAL EXPR))
        (RETURN (CLOSEF STREAM])
```

(IPOSITION

```
[LAMBDA (FILE N FLG)                                           (* kbr: "19-Oct-85 16:31")
  (LET (PAIR)
       [COND
          ((NULL (SETQ PAIR (ASSOC FILE IPOSITION-ALIST)))
           (SETQ IPOSITION-ALIST (CONS (SETQ PAIR (CONS FILE 0))
                                       IPOSITION-ALIST]
       (COND
          ((NULL N)
           (CDR PAIR))
          [FLG (PROG1 (CDR PAIR)
                      (RPLACD PAIR (IPLUS N (CDR PAIR))))]
          (T (PROG1 (CDR PAIR)
                    (RPLACD PAIR N])
```

(ITERPRI

```
[LAMBDA (FILE)                                                 (* kbr: "19-Oct-85 16:31")
  (IPOSITION FILE 0 NIL)
  (TERPRI FILE])
```

(ITERPRIN

```
[LAMBDA (N FILE)                                               (* kbr: "19-Oct-85 16:31")
  (for I from 1 to N do (ITERPRI FILE])
```

(ITERPRISPACES

```
[LAMBDA (N FILE)                                               (* kbr: "19-Oct-85 16:31")
  (ITERPRI FILE)
  (TABULATE N FILE])
```

(IPRIN1

```
[LAMBDA (X FILE)                                               (* kbr: "19-Oct-85 16:31")
  (IPOSITION FILE (NCHARS X)
```

```
          T)
    (PRIN2 X FILE])
```

(IPRINC

```
  [LAMBDA (X FILE)                                          (* kbr: "19-Oct-85 16:31")
    (IPOSITION FILE (NCHARS X)
          T)
    (PRIN1 X FILE])
```

(IPRINT

```
  [LAMBDA (X FILE)                                          (* kbr: "19-Oct-85 16:31")
    (IPOSITION FILE (NCHARS X)
          NIL)
    (PRINT X FILE])
```

(ISPACES

```
  [LAMBDA (N FILE)                                          (* kbr: "19-Oct-85 16:31")
    (COND
       ((ILEQ N 0)
        NIL)
       (T (IPOSITION FILE N T)
          (for I from 1 to N do (PRIN1 " " FILE])
```

(KILL-DEFINITION

```
  [LAMBDA (FN)                                              (* kbr: "17-Nov-85 15:37")
    (PUTD FN NIL])
```

(LINEL

```
  [LAMBDA (FILE N)                                          (* kbr: "19-Oct-85 16:31")
    (LINELENGTH N FILE])
```

(MAKE-LIB

```
  [LAMBDA (FILE)                                            (* kbr: "30-Sep-86 14:05")
    (PROG (TEMP PROP-FILE FN-FILE FILE-PLIST REVERSED-LIB-PROPS)
          (SETQ REVERSED-LIB-PROPS (REVERSE LIB-PROPS))
          (SETQ PROP-FILE (OPENSTREAM (EXTEND-FILE-NAME FILE (QUOTE LIB))
                                      (QUOTE OUTPUT)))
          (PRINT (LIST (QUOTE INIT-LIB)
                       (KWOTE LIB-PROPS)
                       (KWOTE LIB-VARS))
                 PROP-FILE)
          (for VAR in LIB-VARS do (PRINT (LIST (QUOTE SETQ)
                                               VAR
                                               (KWOTE (GETTOPVAL VAR)))
                                         PROP-FILE))
          (PRINT (LIST (QUOTE SETQ)
                       (QUOTE LIB-ATOMS-WITH-PROPS)
                       (KWOTE LIB-ATOMS-WITH-PROPS))
                 PROP-FILE)
          (PRINT (LIST (QUOTE SETQ)
                       (QUOTE LIB-ATOMS-WITH-DEFS)
                       (KWOTE LIB-ATOMS-WITH-DEFS))
                 PROP-FILE)
          (for ATM in LIB-ATOMS-WITH-PROPS
             do (HPRINT [LIST (QUOTE PUT1-LST)
                              (KWOTE ATM)
                              (KWOTE (for PROP in REVERSED-LIB-PROPS join (COND
                                                                            ((SETQ TEMP (MEMB PROP (GETPROPLIST
                                                                                                    ATM)))
                                                                             (LIST PROP (CADR TEMP]
                        PROP-FILE))
          (for ATM in (REVERSE LIB-ATOMS-WITH-DEFS)
             do (HPRINT [LIST (QUOTE PUT1-LST)
                              (KWOTE ATM)
                              (KWOTE (LIST (QUOTE SEXPR)
                                           (LIST (QUOTE LAMBDA)
                                                 [CADR (SETQ TEMP (GETPROP ATM (QUOTE SEXPR]
                                                 (CADDR TEMP]
                        PROP-FILE))
          (PRINT (QUOTE STOP)
                 PROP-FILE)
          (SETQ PROP-FILE (CLOSEF PROP-FILE))
          (SETQ FILECOMS (FILECOMS FILE))
          [SET FILECOMS (LIST (CONS (QUOTE FNS)
                                    (REVERSE LIB-ATOMS-WITH-DEFS]
          [SETQ FN-FILE (BCOMPL (MAKEFILE FILE (QUOTE NEW]
          (RETURN (LIST PROP-FILE FN-FILE])
```

(MATCH-MACRO

```
  [LAMBDA (TERM PAT)                                        (* kbr: "19-Oct-85 16:31")
```

```
    (COND
       ((LISTP TERM)
        (LIST (QUOTE PROGN)
              (LIST (QUOTE SETQ)
                    (QUOTE MATCH-TEMP)
                    TERM)
              (MATCH1-MACRO (QUOTE MATCH-TEMP)
                     PAT)))
       (T (MATCH1-MACRO TERM PAT])
```

(MATCH!-MACRO

```
  [LAMBDA (TERM PAT)                                        (* kbr: "24-Oct-85 16:37")
    (LIST (QUOTE OR)
          (MATCH-MACRO TERM PAT)
          (QUOTE (ERROR "MATCH! failed!"])
```

(MATCH1-MACRO

```
  [LAMBDA (TERM PAT)                                        (* kbr: "19-Oct-85 16:31")
    (LET (TEST-LST SETQ-LST)
         (MATCH2-MACRO TERM PAT)
         (LIST (QUOTE COND)
               (CONS (COND
                        ((NULL TEST-LST)
                         T)
                        ((NULL (CDR TEST-LST))
                         (CAR TEST-LST))
                        (T (CONS (QUOTE AND)
                                 TEST-LST)))
                     (NCONC1 SETQ-LST T])
```

(MATCH2-MACRO

```
  [LAMBDA (TERM PAT)                                        (* kbr: "19-Oct-85 16:31")
    (COND
       [(NLISTP PAT)
        (COND
           ((EQ PAT (QUOTE &))
            NIL)
           ((OR (EQ PAT T)
                (EQ PAT NIL))
            (PRIN1 "***** Attempt to smash T or NIL ignored *****" T)
            (TERPRI T)
            (SPACES 6 T)
            (PRIN2 (CONS (QUOTE BM-MATCH)
                         MATCH-X)
                   T)
            (ITERPRI T))
           [(LITATOM PAT)
            (SETQ SETQ-LST (NCONC1 SETQ-LST (LIST (QUOTE SETQ)
                                                  PAT TERM]
           (T (SETQ TEST-LST (NCONC1 TEST-LST (LIST (QUOTE EQUAL)
                                                    PAT TERM]
       ((EQ (QUOTE CONS)
            (CAR PAT))
        (SETQ TEST-LST (NCONC1 TEST-LST (LIST (QUOTE LISTP)
                                              TERM)))
        (MATCH2-MACRO (LIST (QUOTE CAR)
                            TERM)
              (CADR PAT))
        (MATCH2-MACRO (LIST (QUOTE CDR)
                            TERM)
              (CADDR PAT)))
       [(EQ (QUOTE QUOTE)
            (CAR PAT))
        (COND
           [(LITATOM (CADR PAT))
            (SETQ TEST-LST (NCONC1 TEST-LST (LIST (QUOTE EQ)
                                                  (LIST (QUOTE QUOTE)
                                                        (CADR PAT))
                                                  TERM]
           (T (SETQ TEST-LST (NCONC1 TEST-LST (LIST (QUOTE EQUAL)
                                                    (LIST (QUOTE QUOTE)
                                                          (CADR PAT))
                                                    TERM]
       (T [COND
             ((NEQ (CAR PAT)
                   (QUOTE LIST))
              (SETQ PAT (CONS (QUOTE LIST)
                              (CONS (LIST (QUOTE QUOTE)
                                          (CAR PAT))
                                    (CDR PAT]
              (for SUBPAT in (CDR PAT) do (SETQ TEST-LST (NCONC1 TEST-LST (LIST (QUOTE LISTP)
                                                                                TERM)))
                                          (MATCH2-MACRO (LIST (QUOTE CAR)
                                                              TERM)
```

```
                                        SUBPAT)
                                   (SETQ TERM (LIST (QUOTE CDR)
                                                    TERM)))
          (SETQ TEST-LST (NCONC1 TEST-LST (LIST (QUOTE EQ)
                                                TERM NIL])
```

(NOTE-LIB

```
  [LAMBDA (FILE)                                                    (* kbr: " 8-Nov-85 15:47")
    (PROG (FILE1 FILE2)
          (SETQ FILE1 (EXTEND-FILE-NAME FILE (QUOTE LIB)))
          (SETQ FILE2 (EXTEND-FILE-NAME FILE (QUOTE DCOM)))
          (COND
             ((BOUNDP (QUOTE LIB-FILE))
              (KILL-LIB)))
          (RETURN (LIST (SETQ LIB-FILE (HLOAD FILE1))
                        (LOAD FILE2])
```

(BM-NTH

```
  [LAMBDA (N LIST)                                                  (* kbr: "19-Oct-85 18:37")
    (CAR (NTH LIST N])
```

(PREPARE-FOR-THE-NIGHT

```
  [LAMBDA NIL                                                       (* kbr: "19-Oct-85 16:31")
    NIL])
```

(SPELL-NUMBER

```
  [LAMBDA (N)                                                       (* kbr: "26-Oct-85 16:31")
    (SELECTQ N
         (0 (QUOTE ZERO))
         (1 (QUOTE ONE))
         (2 (QUOTE TWO))
         (3 (QUOTE THREE))
         (4 (QUOTE FOUR))
         (5 (QUOTE FIVE))
         (6 (QUOTE SIX))
         (7 (QUOTE SEVEN))
         (8 (QUOTE EIGHT))
         (9 (QUOTE NINE))
         (10 (QUOTE TEN))
        N])
```

(SUB-PAIR

```
  [LAMBDA (L1 L2 X)                                                 (* kbr: " 6-Jul-86 10:21")

          (* * Substitution function. This is like (SUBLIS (PAIRLIST L1 L2) X) *)

    (COND
       ((for Z in L2 as Y in L1 when (EQUAL Y X) thereis (PROGN (SETQ TEMP-TEMP Z)
                                                                T))
        TEMP-TEMP)
       ((NLISTP X)
        X)
       (T (CONS (SUB-PAIR L1 L2 (CAR X))
                (SUB-PAIR L1 L2 (CDR X])
```

(UNIONQ

```
  [LAMBDA (LIST1 LIST2)                                             (* kbr: "17-Nov-85 15:41")
    (PROG (ANSWER)
          (SETQ ANSWER LIST2)
          (for ELEMENT in LIST1 when (NOT (MEMB ELEMENT ANSWER)) do (push ANSWER ELEMENT))
          (RETURN ANSWER])

)

(SETQ LEMMA-STACK (CREATE-LEMMA-STACK 10))

(SETQ LINEARIZE-ASSUMPTIONS-STACK (CREATE-LINEARIZE-ASSUMPTIONS-STACK 10))

(RPAQQ CODE-1-ACOMS
       ((* CODE-1-A *)
        (FNS *1*ADD1 *1*AND *1*CONS *1*COUNT *1*DIFFERENCE *1*EQUAL *1*FALSE *1*FALSEP *1*FIX *1*IMPLIES
             *1*LESSP *1*LISTP *1*LITATOM *1*MINUS *1*NEGATIVE-GUTS *1*NEGATIVEP *1*NLISTP *1*NOT *1*NUMBERP
             *1*OR *1*PACK *1*PLUS *1*QUOTIENT *1*REMAINDER *1*SUB1 *1*TIMES *1*TRUE *1*TRUEP *1*UNPACK *1*ZERO
             *1*ZEROP ABBREVIATIONP ABBREVIATIONP1 ACCEPTABLE-TYPE-PRESCRIPTION-LEMMAP ACCESS-ERROR ADD-AXIOM1
             ADD-DCELL ADD-ELIM-LEMMA ADD-EQUATION ADD-EQUATIONS ADD-EQUATIONS-TO-POT-LST ADD-FACT
             ADD-GENERALIZE-LEMMA ADD-LEMMA ADD-LEMMA0 ADD-LESSP-ASSUMPTION-TO-POLY ADD-LINEAR-TERM
             ADD-LINEAR-VARIABLE ADD-LINEAR-VARIABLE1 ADD-LITERAL ADD-META-LEMMA
             ADD-NOT-EQUAL-0-ASSUMPTION-TO-POLY ADD-NOT-LESSP-ASSUMPTION-TO-POLY ADD-NUMBERP-ASSUMPTION-TO-POLY
             ADD-PROCESS-HIST ADD-REWRITE-LEMMA ADD-SHELL-ROUTINES ADD-SHELL0 ADD-SUB-FACT ADD-TERM-TO-POT-LST
             ADD-TERMS-TO-POT-LST ADD-TO-SET-EQ ADD-TYPE-SET-LEMMAS ALL-ARGLISTS ALL-FNNAMES ALL-FNNAMES-LST
             ALL-FNNAMES1 ALL-FNNAMES1-EVG ALL-INSERTIONS ALL-PATHS ALL-PERMUTATIONS ALL-PICKS ALL-SUBSEQUENCES
```

```
            ALL-VARS ALL-VARS-BAG ALL-VARS-BAG1 ALL-VARS-LST ALL-VARS1 ALMOST-SUBSUMES ALMOST-SUBSUMES-LOOP
            ALMOST-VALUEP ALMOST-VALUEP1 APPLY-HINTS APPLY-INDUCT-HINT APPLY-USE-HINT ARG1-IN-ARG2-UNIFY-SUBST
            ARGN0 ARITY ASSOC-OF-APPEND ASSUME-TRUE-FALSE ATTEMPT-TO-REWRITE-RECOGNIZER)))

          (* * CODE-1-A *)

(DEFINEQ

(*1*ADD1
  [LAMBDA (X)                                                    (* kbr: "19-Oct-85 16:31")
    (COND
       ((AND (FIXP X)
             (LESSEQP 0 X))
        (ADD1 X))
       (T 1])

(*1*AND
  [LAMBDA (X Y)                                                  (* kbr: "19-Oct-85 16:31")
    (*1*IF X (*1*IF Y *1*T *1*F)
          *1*F])

(*1*CONS
  [LAMBDA (X Y)                                                  (* kbr: "19-Oct-85 16:31")
    (CONS X Y])

(*1*COUNT
  [LAMBDA (X)                                                    (* kbr: "19-Oct-85 16:31")
    (COND
       ((NLISTP X)
        (COND
           ((EQ X *1*T)
            0)
           ((EQ X *1*F)
            0)
           [(LITATOM X)
            (ADD1 (*1*COUNT (DTACK-0-ON-END (CHCON X]
           ((LESSP X 0)
            (ADD1 (MINUS X)))
           (T X)))
       [(EQ *1*SHELL-QUOTE-MARK (CAR X))
        (COND
           ((MEMB (CADR X)
                  *1*BTM-OBJECTS)
            0)
           (T (ADD1 (for ARG in (CDDR X) sum (*1*COUNT ARG]
       (T (ADD1 (PLUS (*1*COUNT (CAR X))
                      (*1*COUNT (CDR X])

(*1*DIFFERENCE
  [LAMBDA (I J)                                                  (* kbr: "19-Oct-85 16:31")
    (COND
       ((GREATERP (SETQ I (*1*FIX I))
                  (SETQ J (*1*FIX J)))
        (DIFFERENCE I J))
       (T 0])

(*1*EQUAL
  [LAMBDA (X Y)                                                  (* kbr: "19-Oct-85 16:31")
    (COND
       ((EQUAL X Y)
        *1*T)
       (T *1*F])

(*1*FALSE
  [LAMBDA NIL                                                    (* kbr: "19-Oct-85 16:31")
    *1*F])

(*1*FALSEP
  [LAMBDA (X)                                                    (* kbr: "19-Oct-85 16:31")
    (COND
       ((EQ X *1*F)
        *1*T)
       (T *1*F])

(*1*FIX
  [LAMBDA (X)                                                    (* kbr: "19-Oct-85 16:31")
    (COND
       ((AND (FIXP X)
```

```
            (LESSEQP 0 X))
       X)
      (T 0])
```

(***1*IMPLIES**

```
[LAMBDA (X Y)                                          (* kbr: "19-Oct-85 16:31")
  (*1*IF X (*1*IF Y *1*T *1*F)
         *1*T])
```

(***1*LESSP**

```
[LAMBDA (X Y)                                          (* kbr: "19-Oct-85 16:31")
  (COND
     ((LESSP (*1*FIX X)
             (*1*FIX Y))
      *1*T)
     (T *1*F])
```

(***1*LISTP**

```
[LAMBDA (X)                                            (* kbr: "19-Oct-85 16:31")
  (COND
     ((AND (LISTP X)
           (NEQ (CAR X)
                *1*SHELL-QUOTE-MARK))
      *1*T)
     (T *1*F])
```

(***1*LITATOM**

```
[LAMBDA (X)                                            (* kbr: "19-Oct-85 16:31")
  (COND
     ([OR (AND (LITATOM X)
               (NEQ X *1*T)
               (NEQ X *1*F))
          (AND (LISTP X)
               (EQ (CAR X)
                   *1*SHELL-QUOTE-MARK)
               (EQ (CADR X)
                   (QUOTE PACK]
      *1*T)
     (T *1*F])
```

(***1*MINUS**

```
[LAMBDA (X)                                            (* kbr: "19-Oct-85 16:31")
  (COND
     ((AND (FIXP X)
           (LESSP 0 X))
      (MINUS X))
     (T (LIST *1*SHELL-QUOTE-MARK (QUOTE MINUS)
              0])
```

(***1*NEGATIVE-GUTS**

```
[LAMBDA (X)                                            (* kbr: "19-Oct-85 16:31")
  (COND
     ((AND (FIXP X)
           (LESSP X 0))
      (MINUS X))
     (T 0])
```

(***1*NEGATIVEP**

```
[LAMBDA (X)                                            (* kbr: "19-Oct-85 16:31")
  (COND
     ([OR (AND (FIXP X)
               (LESSP X 0))
          (AND (LISTP X)
               (EQ (CAR X)
                   *1*SHELL-QUOTE-MARK)
               (EQ (CADR X)
                   (QUOTE MINUS]
      *1*T)
     (T *1*F])
```

(***1*NLISTP**

```
[LAMBDA (X)                                            (* kbr: "19-Oct-85 16:31")
  (COND
     ((AND (LISTP X)
           (NEQ (CAR X)
                *1*SHELL-QUOTE-MARK))
      *1*F)
     (T *1*T])
```

```
(*1*NOT
  [LAMBDA (X)                                                    (* kbr: "19-Oct-85 16:31")
    (*1*IF X *1*F *1*T])

(*1*NUMBERP
  [LAMBDA (X)                                                    (* kbr: "19-Oct-85 16:31")
    (COND
       ((AND (FIXP X)
             (LESSEQP 0 X))
        *1*T)
       (T *1*F])

(*1*OR
  [LAMBDA (X Y)                                                  (* kbr: "19-Oct-85 16:31")
    (*1*IF X *1*T (*1*IF Y *1*T *1*F])

(*1*PACK
  [LAMBDA (X)                                                    (* kbr: "19-Oct-85 16:31")
    (COND
       [[AND (LEGAL-CHAR-CODE-SEQ X)
             (EQUAL 0 (CDR (LAST X]
        (TP-IMPLODE (for TAIL on X until (NLISTP TAIL) collect (CHARACTER (CAR TAIL]
       (T (LIST *1*SHELL-QUOTE-MARK (QUOTE PACK)
                X])

(*1*PLUS
  [LAMBDA (X Y)                                                  (* kbr: "19-Oct-85 16:31")
    (PLUS (*1*FIX X)
          (*1*FIX Y])

(*1*QUOTIENT
  [LAMBDA (I J)                                                  (* kbr: "19-Oct-85 16:31")
    (COND
       ((EQUAL 0 (SETQ J (*1*FIX J)))
        0)
       (T (QUOTIENT (*1*FIX I)
                    J])

(*1*REMAINDER
  [LAMBDA (I J)                                                  (* kbr: "19-Oct-85 16:31")
    (COND
       ((EQUAL 0 (SETQ J (*1*FIX J)))
        (*1*FIX I))
       (T (REMAINDER (*1*FIX I)
                     J])

(*1*SUB1
  [LAMBDA (X)                                                    (* kbr: "19-Oct-85 16:31")
    (COND
       ((AND (FIXP X)
             (LESSP 0 X))
        (SUB1 X))
       (T 0])

(*1*TIMES
  [LAMBDA (I J)                                                  (* kbr: "19-Oct-85 16:31")
    (TIMES (*1*FIX I)
           (*1*FIX J])

(*1*TRUE
  [LAMBDA NIL                                                    (* kbr: "19-Oct-85 16:31")
    *1*T])

(*1*TRUEP
  [LAMBDA (X)                                                    (* kbr: "19-Oct-85 16:31")
    (COND
       ((EQ X *1*T)
        *1*T)
       (T *1*F])

(*1*UNPACK
  [LAMBDA (X)                                                    (* kbr: "19-Oct-85 16:31")
    (COND
       ((AND (LITATOM X)
             (NEQ X *1*T)
```

```
            (NEQ X *1*F))
       (LET ((TEMP (CHCON X)))
            (RPLACD (LAST TEMP)
                    0)
            TEMP))
      ((AND (LISTP X)
            (EQ *1*SHELL-QUOTE-MARK (CAR X))
            (EQ (CADR X)
                (QUOTE PACK)))
       (CADDR X))
      (T 0])
```

(*1*ZERO

```
[LAMBDA NIL                                                    (* kbr: "19-Oct-85 16:31")
  0])
```

(*1*ZEROP

```
[LAMBDA (X)                                                    (* kbr: "19-Oct-85 16:31")
  (COND
     ((FIXP X)
      (COND
         ((LESSP X 1)
          *1*T)
         (T *1*F)))
     (T *1*T])
```

(ABBREVIATIONP

```
[LAMBDA (VARS TERM)                                            (* kbr: "19-Oct-85 16:31")
```

(* Suppose VARS is the bag of vars in a term LHS. Then we say LHS=TERM is an abbreviation if the bag of vars occurring in TERM is a subbag of VARS and TERM contains no IF, AND, OR, NOT, or IMPLIES.
The property of VARS that we actually check is that the number of occurrences of vars in TERM is no greater than the length of VARS. *)

```
  (LET ((ANS VARS))
       (ABBREVIATIONP1 TERM])
```

(ABBREVIATIONP1

```
[LAMBDA (TERM)                                                 (* kbr: "19-Oct-85 16:31")
  (COND
     ((VARIABLEP TERM)
      (COND
         ((NLISTP ANS)
          NIL)
         (T (SETQ ANS (CDR ANS))
            T)))
     ((FQUOTEP TERM)
      T)
     ((MEMB (FFN-SYMB TERM)
            (QUOTE (IF AND OR NOT IMPLIES)))
      NIL)
     (T (for X in (FARGS TERM) always (ABBREVIATIONP1 X])
```

(ACCEPTABLE-TYPE-PRESCRIPTION-LEMMAP

```
[LAMBDA (HYPS CONCL)                                           (* kbr: "20-Oct-85 19:47")
```

(* If (IMPLIES HYPS CONCL) is a type prescription lemma for some function symbol, compute the function symbol and return the function symbol consed onto the type prescription described by the lemma.
Otherwise return NIL. *)

```
  (PROG (TERM RECOG CLAUSES VARS NEGFLG CONST ARG VAR)         (* Set TERM to the IF form of (IMPLIES HYP CONCL) . *)
        (SETQ TERM (EXPAND-NON-REC-FNS (FCONS-TERM* (QUOTE IF)
                                                    (CONJOIN HYPS T)
                                                    (FCONS-TERM* (QUOTE IF)
                                                                 CONCL TRUE FALSE)
                                                    TRUE)))
```

(* Acceptable type prescription lemmas must contain exactly one function symbol other than IF, EQUAL, recognizers and singleton constructors. *)

```
        (COND
           ([NOT (IEQP 1 (for FN in (ALL-FNNAMES TERM) count (AND (NOT (ASSOC FN RECOGNIZER-ALIST))
                                                                  (NOT (SINGLETON-CONSTRUCTOR-TO-RECOGNIZER
                                                                        FN]
            (RETURN NIL)))
```

(* Consider a clause in the clausification of a type prescription lemma.
You should be able to divide the literals into two sets. The first set should consist entirely of recognizers applied to some term (fn v1 ... vn) or of negations of recognizers applied to such a term.
The second set should consist entirely of equations between that term and some of the variables vi.
Actually, some literals are of the form (EQUAL term (TRUE)) but these are equivalent to
(TRUEP term) . *)

```
(SETQ CLAUSES (CLAUSIFY TERM))
```

(* We now map over CLAUSES and replace all atoms of the form
(EQUAL & (singleton)) by (singletonp &) just to reduce the number of cases.
*)

```
[SETQ CLAUSES (for CL in CLAUSES
                 collect (for LIT in CL collect (PROGN (SETQ NEGFLG (BM-MATCH LIT (NOT LIT)))
                                                       (SETQ LIT (COND
                                                                   ([OR [AND (BM-MATCH LIT
                                                                                       (EQUAL TERM CONST))
                                                                             (NVARIABLEP CONST)
                                                                             (SETQ TEMP-TEMP
                                                                              (
                                                                    SINGLETON-CONSTRUCTOR-TO-RECOGNIZER
                                                                               (FN-SYMB CONST]
                                                                         (AND (BM-MATCH LIT
                                                                                       (EQUAL CONST TERM))
                                                                             (NVARIABLEP CONST)
                                                                             (SETQ TEMP-TEMP
                                                                              (
                                                                    SINGLETON-CONSTRUCTOR-TO-RECOGNIZER
                                                                               (FN-SYMB CONST]
                                                                    (FCONS-TERM* TEMP-TEMP TERM))
                                                                   (T LIT)))
                                                       (COND
                                                          (NEGFLG (FCONS-TERM* (QUOTE NOT)
                                                                               LIT))
                                                          (T LIT]
```

(* We now try to find the function that this supposed type prescription is about.
We look at the first literal of the first clause and it had better be a recognizer applied to something, a NOT recognizer applied to something, or the equality of a non variable something and another term.
If we can find such a something, we set it to TERM. Otherwise, we say this is not a type prescription lemma.
*)

```
(COND
   ([NOT (AND (LISTP CLAUSES)
              (LISTP (CAR CLAUSES))
              (OR (AND (BM-MATCH (CAAR CLAUSES)
                                 (NOT (LIST RECOG TERM)))
                       (ASSOC RECOG RECOGNIZER-ALIST))
                  (AND (BM-MATCH (CAAR CLAUSES)
                                 (LIST RECOG TERM))
                       (ASSOC RECOG RECOGNIZER-ALIST))
                  (AND (BM-MATCH (CAAR CLAUSES)
                                 (EQUAL TERM &))
                       (NVARIABLEP TERM))
                  (AND (BM-MATCH (CAAR CLAUSES)
                                 (EQUAL & TERM))
                       (NVARIABLEP TERM]
    (RETURN NIL)))                                    (* TERM must be a function application to distinct variables.
                                                      *)
(COND
   ([NOT (AND (NVARIABLEP TERM)
              (for ARG in (SARGS TERM) always (VARIABLEP ARG))
              (NO-DUPLICATESP (SARGS TERM]
    (RETURN NIL)))
```

(* Every literal of every clause must be a recognizer applied to TERM, the negation of a recognizer applied to TERM, or the equality between TERM and one of the vars in its arglist. As a side-effect of this check, we collect in VARS all of the variables equated to TERM. *)

```
(COND
   ([NOT (for CL in CLAUSES always (for LIT in CL
                                    always (OR (AND (BM-MATCH LIT (LIST RECOG ARG))
                                                    (ASSOC RECOG RECOGNIZER-ALIST)
                                                    (EQUAL ARG TERM))
                                               (AND (BM-MATCH LIT (NOT (LIST RECOG ARG)))
                                                    (ASSOC RECOG RECOGNIZER-ALIST)
                                                    (EQUAL ARG TERM))
                                               (AND (BM-MATCH LIT (EQUAL ARG VAR))
                                                    (EQUAL ARG TERM)
                                                    (MEMB VAR (SARGS TERM))
                                                    (SETQ VARS (ADD-TO-SET VAR VARS)))
                                               (AND (BM-MATCH LIT (EQUAL VAR ARG))
                                                    (EQUAL ARG TERM)
                                                    (MEMB VAR (SARGS TERM))
                                                    (SETQ VARS (ADD-TO-SET VAR VARS]
    (RETURN NIL)))
```

(* Every clause must contain the same set of equations of TERM with vars.
Since VARS contains all of the vars ever equated with TERM in any clause, all that remains is to make sure that every clause contains an equation with each var in VARS. *)

```
(COND
```

```
            ([NOT (for VAR in VARS always (for CL in CLAUSES always (OR (MEMBER (FCONS-TERM* (QUOTE EQUAL)
                                                                                TERM VAR)
                                                                          CL)
                                                                    (MEMBER (FCONS-TERM* (QUOTE EQUAL)
                                                                                VAR TERM)
                                                                          CL]
             (RETURN NIL)))

        (* So we believe that (IMPLIES HYP CONCL) is a type prescription lemma.
        Return the function symbol of TERM, consed onto the type prescription.
        The type prescription is itself a cons of the type bits and flags indicating which args are in VARS.
        The type bits are obtained by anding together the disjunction of recognizers in each clause.
        *)

        (RETURN
         (CONS (FN-SYMB TERM)
               (CONS (for CL in CLAUSES bind (LOOP-ANS _ -1)
                        do [SETQ LOOP-ANS (LOGAND LOOP-ANS
                                                  (for LIT in CL bind (LOOP-ANS _ 0)
                                                     when (NEQ (FN-SYMB LIT)
                                                               (QUOTE EQUAL))
                                                     do [SETQ LOOP-ANS
                                                          (LOGOR LOOP-ANS (COND
                                                                            [(BM-MATCH LIT (NOT LIT))
                                                                             (LOGNOT (CDR (SASSOC (FN-SYMB
                                                                                                    LIT)

                                                                                                  RECOGNIZER-ALIST
                                                                                                  ]
                                                                            (T (CDR (SASSOC (FN-SYMB LIT)
                                                                                            RECOGNIZER-ALIST]
                                                     finally (RETURN LOOP-ANS]
                        finally (RETURN LOOP-ANS))
                     (for V in (SARGS TERM) collect (COND
                                                      ((MEMB V VARS)
                                                       T)
                                                      (T NIL])
```

(ACCESS-ERROR

```
[LAMBDA (REC)                                                    (* kbr: "19-Oct-85 16:31")
  (ERROR1 (PQUOTE (PROGN ATTEMPT TO USE A RECORD OF THE WRONG TYPE (!PPR REC NIL)))
          (BINDINGS (QUOTE REC)
                    REC)
          (QUOTE HARD])
```

(ADD-AXIOM1

```
[LAMBDA (NAME TYPES TERM)                                        (* kbr: "19-Oct-85 16:31")
```

(* Note that this function is not really a subroutine of ADD-AXIOM which must check that the term is a legal axiom of the types requested and then set up for an event. This function is used by ADD-SHELL0 and the boot strapping to add axioms without creating events. If the system were static those calls of ADD-AXIOM1 could be replaced by ADD-LEMMA0 since we know the lemmas we add are always acceptable. However, we still run the acceptability checks just in case we someday change the criteria for acceptance but forget to change the built in additions of axioms.
*)

```
  (CHK-ACCEPTABLE-LEMMA NAME TYPES TERM)
  (ADD-LEMMA0 NAME TYPES TERM])
```

(ADD-DCELL

```
[LAMBDA (NAME *1*NAME EXPR)                                      (* kbr: "19-Oct-85 16:31")
  (ADD-FACT NAME (QUOTE LISP-CODE)
            *1*NAME)
  (ADD-FACT *1*NAME (QUOTE DCELL)
            EXPR])
```

(ADD-ELIM-LEMMA

```
[LAMBDA (NAME TYPE TERM)                                         (* kbr: "20-Oct-85 15:45")
  TYPE
  (LET (HYPS CONCL REWRITE-RULE DESTS)
       (SETQ TEMP-TEMP (UNPRETTYIFY TERM))
       (SETQ HYPS (CAR (CAR TEMP-TEMP)))
       (SETQ CONCL (CDR (CAR TEMP-TEMP)))
       [SETQ DESTS (DESTRUCTORS (LIST (ARGN CONCL 1]
       (SETQ REWRITE-RULE (CREATE-REWRITE-RULE NAME HYPS CONCL NIL))
       [for X in DESTS do (ADD-FACT (FN-SYMB X)
                                    (QUOTE ELIMINATE-DESTRUCTORS-SEQ)
                                    REWRITE-RULE)
                          (ADD-FACT (FN-SYMB X)
                                    (QUOTE ELIMINATE-DESTRUCTORS-DESTS)
                                    (CONS X (REMOVE X DESTS]
       NIL])
```

(ADD-EQUATION

```
  [LAMBDA (EQUATION POT-LST)                                  (* kbr: "22-Oct-85 14:08")

          (* This function returns an EQ POT-LST in the event that EQUATION caused nothing to change.
          *)

    (LET (ADD-EQUATION-ANS TO-DO-NEXT NEW-POT-- NEW-POT-+)
         (COND
            ([OR (NULL POT-LST)
                 (NOT (TERM-ORDER (fetch (LINEAR-POT VAR) of (CAR POT-LST))
                                  (FIRST-VAR EQUATION]
             (SETQ ADD-EQUATIONS-TO-DO (COND
                                         ((SETQ TEMP-TEMP (CANCEL-POSITIVE EQUATION))
                                          (LIST TEMP-TEMP))
                                         (T NIL)))
             (CONS [COND
                      ((GREATERP (FIRST-COEFFICIENT EQUATION)
                                 0)
                       (create LINEAR-POT
                               VAR _ (FIRST-VAR EQUATION)
                               POSITIVES _ (LIST EQUATION)))
                      (T (create LINEAR-POT
                                 VAR _ (FIRST-VAR EQUATION)
                                 NEGATIVES _ (LIST EQUATION]
                   POT-LST))
            [(EQUAL (fetch (LINEAR-POT VAR) of (CAR POT-LST))
                    (FIRST-VAR EQUATION))
             (COND
                ([POLY-MEMBER EQUATION (COND
                                         ((GREATERP (FIRST-COEFFICIENT EQUATION)
                                                    0)
                                          (fetch (LINEAR-POT POSITIVES) of (CAR POT-LST)))
                                         (T (fetch (LINEAR-POT NEGATIVES) of (CAR POT-LST]
                 (SETQ ADD-EQUATIONS-TO-DO NIL)
                 POT-LST)
                (T (SETQ ADD-EQUATIONS-TO-DO (for EQUATION1 in [COND
                                                                 ((GREATERP (FIRST-COEFFICIENT EQUATION)
                                                                            0)
                                                                  (fetch (LINEAR-POT NEGATIVES)
                                                                     of (CAR POT-LST)))
                                                                 (T (fetch (LINEAR-POT POSITIVES)
                                                                       of (CAR POT-LST]
                                                    bind TEMP unless [OR (TO-BE-IGNOREDP EQUATION1)
                                                                         (NULL (SETQ TEMP (CANCEL EQUATION EQUATION1]
                                                    collect TEMP))
                   [COND
                      ((SETQ TEMP-TEMP (CANCEL-POSITIVE EQUATION))
                       (SETQ ADD-EQUATIONS-TO-DO (CONS TEMP-TEMP ADD-EQUATIONS-TO-DO]
                   (CONS [COND
                            [(GREATERP (FIRST-COEFFICIENT EQUATION)
                                       0)
                             (create LINEAR-POT
                                     VAR _ (fetch (LINEAR-POT VAR) of (CAR POT-LST))
                                     POSITIVES _ (CONS EQUATION (fetch (LINEAR-POT POSITIVES) of (CAR POT-LST)))
                                     NEGATIVES _ (fetch (LINEAR-POT NEGATIVES) of (CAR POT-LST]
                            (T (create LINEAR-POT
                                       VAR _ (fetch (LINEAR-POT VAR) of (CAR POT-LST))
                                       POSITIVES _ (fetch (LINEAR-POT POSITIVES) of (CAR POT-LST))
                                       NEGATIVES _ (CONS EQUATION (fetch (LINEAR-POT NEGATIVES)
                                                                     of (CAR POT-LST]
                         (CDR POT-LST]
            (T (SETQ ADD-EQUATION-ANS (ADD-EQUATION EQUATION (CDR POT-LST)))
               (SETQ TO-DO-NEXT NIL)
               (SETQ NEW-POT-+ (fetch (LINEAR-POT POSITIVES) of (CAR POT-LST)))
               (SETQ NEW-POT-- (fetch (LINEAR-POT NEGATIVES) of (CAR POT-LST)))
               [for EQUATION in ADD-EQUATIONS-TO-DO
                  do (COND
                        [(EQUAL (fetch (LINEAR-POT VAR) of (CAR POT-LST))
                                (FIRST-VAR EQUATION))
                         (for EQUATION1 in [COND
                                             ((GREATERP (FIRST-COEFFICIENT EQUATION)
                                                        0)
                                              (COND
                                                 ((POLY-MEMBER EQUATION NEW-POT-+)
                                                  NIL)
                                                 (T [COND
                                                       ((SETQ TEMP-TEMP (CANCEL-POSITIVE EQUATION))
                                                        (SETQ TO-DO-NEXT (CONS TEMP-TEMP TO-DO-NEXT]
                                                    (SETQ NEW-POT-+ (CONS EQUATION NEW-POT-+))
                                                    NEW-POT--)))
                                             (T (COND
                                                   ((POLY-MEMBER EQUATION NEW-POT--)
                                                    NIL)
                                                   (T (SETQ NEW-POT-- (CONS EQUATION NEW-POT--))
                                                      NEW-POT-+]
                                            bind TEMP unless [OR (TO-BE-IGNOREDP EQUATION1)
                                                                 (NULL (SETQ TEMP (CANCEL EQUATION EQUATION1]
```

```
                        do (SETQ TO-DO-NEXT (CONS TEMP TO-DO-NEXT]
                     (T (SETQ TO-DO-NEXT (CONS EQUATION TO-DO-NEXT]
               (SETQ ADD-EQUATIONS-TO-DO TO-DO-NEXT)
               (COND
                  ((AND (EQ ADD-EQUATION-ANS (CDR POT-LST))
                        (EQ (fetch (LINEAR-POT POSITIVES) of (CAR POT-LST))
                            NEW-POT-+)
                        (EQ (fetch (LINEAR-POT NEGATIVES) of (CAR POT-LST))
                            NEW-POT--))
```

(* This is where we make sure we return an EQ POT-LST if nothing happened.
*)

```
                POT-LST)
               (T (CONS (create LINEAR-POT
                                VAR _ (fetch (LINEAR-POT VAR) of (CAR POT-LST))
                                POSITIVES _ NEW-POT-+
                                NEGATIVES _ NEW-POT--)
                        ADD-EQUATION-ANS])
```

(ADD-EQUATIONS

```
[LAMBDA (EQUATIONS POT-LST)                                    (* kbr: "19-Oct-85 16:31")
  (LET (NEW-EQUATIONS ADD-EQUATIONS-TO-DO)
       (SETQ EQUATIONS (for EQUATION in EQUATIONS when (COND
                                                         ((IMPOSSIBLE-POLYP EQUATION)
                                                          (SETQ LINEAR-ASSUMPTIONS (fetch (POLY ASSUMPTIONS)
                                                                                          of EQUATION))
                                                          (SETQ LEMMAS-USED-BY-LINEAR
                                                           (UNIONQ (fetch (POLY LEMMAS) of EQUATION)
                                                                   (fetch (POLY LITERALS) of EQUATION)))
                                                          (RETFROM (QUOTE ADD-EQUATIONS)
                                                                 (QUOTE CONTRADICTION)))
                                                         ((TRUE-POLYP EQUATION)
                                                          NIL)
                                                         (T T))
                        collect EQUATION))
       (while EQUATIONS do (for EQUATION in EQUATIONS do (SETQ POT-LST (ADD-EQUATION EQUATION POT-LST))
                                                         (SETQ NEW-EQUATIONS (NCONC ADD-EQUATIONS-TO-DO
                                                                                    NEW-EQUATIONS)))
                           (SETQ EQUATIONS NEW-EQUATIONS)
                           (SETQ NEW-EQUATIONS NIL))
       POT-LST])
```

(ADD-EQUATIONS-TO-POT-LST

```
[LAMBDA (POLY-LST POT-LST ALL-NEW-FLG)                         (* kbr: "24-Oct-85 14:24")
  (PROG (NEW-POT-LST NEW-VARS LST)
        (SETQ NEW-POT-LST (ADD-EQUATIONS POLY-LST POT-LST))
        [COND
           ((EQ NEW-POT-LST (QUOTE CONTRADICTION))
            (RETURN (QUOTE CONTRADICTION]
    TOP (SETQ NEW-VARS (for X in NEW-POT-LST
                          when [AND (NOT (VARIABLEP (fetch (LINEAR-POT VAR) of X)))
                                    (OR ALL-NEW-FLG (NOT (for POT in POT-LST
                                                            thereis (EQUAL (fetch (LINEAR-POT VAR) of POT)
                                                                           (fetch (LINEAR-POT VAR) of X]
                          collect (fetch (LINEAR-POT VAR) of X)))
        (SETQ ALL-NEW-FLG NIL)
        (COND
           ((NULL NEW-VARS)
            (RETURN NEW-POT-LST)))
        (SETQ POT-LST NEW-POT-LST)
        [for VAR in NEW-VARS
           do
           (for LEMMA in (GETPROP (FN-SYMB VAR)
                                  (QUOTE LINEAR-LEMMAS))
              unless (DISABLEDP (fetch (LINEAR-LEMMA NAME) of LEMMA))
              do
```

(* We will rewrite the conclusion of the linear lemma and rewrite the hyps to relieve them.
This will generate both a list of lemmas used and some linear assumptions.
They will be collected in the frames pushed here and will be popped and smashed into the polys we add to the pot should
we succeed. *)

```
              (PUSH-LEMMA-FRAME)
              (PRINT-TO-DISPLAY (QUOTE LINEAR)
                     NIL NIL)
              (PUSH-LINEARIZE-ASSUMPTIONS-FRAME)
              (COND
                 [[AND (ONE-WAY-UNIFY (fetch (LINEAR-LEMMA MAX-TERM) of LEMMA)
                                      VAR)
                       (LET ((SIMPLIFY-CLAUSE-POT-LST NEW-POT-LST))
                            (RELIEVE-HYPS (fetch (LINEAR-LEMMA HYPS) of LEMMA)
                                   (fetch (LINEAR-LEMMA NAME) of LEMMA)))
                       (SETQ LST (LET ((SIMPLIFY-CLAUSE-POT-LST NEW-POT-LST))
                                      (LINEARIZE (REWRITE-LINEAR-CONCL (fetch (LINEAR-LEMMA CONCL)
```

```
                                                                    of LEMMA))
                                                T)))
                                    (NULL (CDR LST))
                                    (for POLY in (CAR LST)
                                       never (for PAIR1 in (fetch (POLY ALIST) of POLY)
                                                thereis (for POT in POT-LST
                                                           always (AND (NOT (EQUAL (CAR PAIR1)
                                                                                   (fetch (LINEAR-POT VAR) of POT)))
                                                                       (GREATEREQP (FORM-COUNT (CAR PAIR1))
                                                                              (FORM-COUNT (fetch (LINEAR-POT VAR)
                                                                                             of POT)))
                                                                       (WORSE-THAN-OR-EQUAL (CAR PAIR1)
                                                                              (fetch (LINEAR-POT VAR) of POT]
                                 [for POLY in (CAR LST) bind (LEMMAS _ (ADD-TO-SET (fetch (LINEAR-LEMMA NAME) of LEMMA)
                                                                                   (POP-LEMMA-FRAME)))
                                                            AND (HYPS _ (POP-LINEARIZE-ASSUMPTIONS-FRAME))
                                    do (replace (POLY LEMMAS) of POLY with LEMMAS)
                                       (replace (POLY ASSUMPTIONS) of POLY with (UNION-EQUAL HYPS (fetch (POLY ASSUMPTIONS)
                                                                                                    of POLY]
                                 (SETQ NEW-POT-LST (ADD-EQUATIONS (CAR LST)
                                                                  NEW-POT-LST))
                                 (COND
                                    ((EQ NEW-POT-LST (QUOTE CONTRADICTION))
                                     (RETFROM (QUOTE ADD-EQUATIONS-TO-POT-LST)
                                              (QUOTE CONTRADICTION]
                                (T (POP-LEMMA-FRAME)
                                   (POP-LINEARIZE-ASSUMPTIONS-FRAME]
          (GO TOP])
```

(ADD-FACT

```
(LAMBDA (ATM PROP VAL)                                  (* kbr: "19-Oct-85 16:31")
  (COND
     (ATM (GUARANTEE-CITIZENSHIP ATM)))
  (ADD-SUB-FACT ATM PROP VAL NIL NIL])
```

(ADD-GENERALIZE-LEMMA

```
(LAMBDA (NAME TYPE TERM)                                (* kbr: "19-Oct-85 16:31")
  TYPE
  (ADD-FACT NIL (QUOTE GENERALIZE-LEMMAS)
         (create GENERALIZE-LEMMA
                 NAME _ NAME
                 TERM _ TERM])
```

(ADD-LEMMA

```
(LAMBDA NIL                                             (* kbr: "19-Oct-85 16:31")
  (IPRINT (QUOTE (ADD-LEMMA IS UNDEFINED. USE EITHER ADD-AXIOM OR PROVE-LEMMA.))
         T])
```

(ADD-LEMMA0

```
(LAMBDA (NAME TYPES TERM)                               (* kbr: "17-Nov-85 15:45")
  (GUARANTEE-CITIZENSHIP NAME)
  (SETQ TYPES (SCRUNCH TYPES))
  (SETQ TERM (TRANSLATE TERM))
  (for TYPE in TYPES do (APPLY* (PACK (LIST (QUOTE ADD-)
                                            (COND
                                               ((LISTP TYPE)
                                                (CAR TYPE))
                                               (T TYPE))
                                            (QUOTE -LEMMA)))
                                NAME TYPE TERM])
```

(ADD-LESSP-ASSUMPTION-TO-POLY

```
(LAMBDA (X Y POLY)                                      (* kbr: "19-Oct-85 16:31")
                                                        (* We add the assumption (LESSP X Y) to POLY.
                                                        See the comment in
                                                        ADD-NUMBERP-ASSUMPTION-TO-POLY.
                                                        *)
  (PROG (TEMP TERM)
        [SETQ TEMP (TYPE-SET (SETQ TERM (FCONS-TERM* (QUOTE LESSP)
                                                     X Y]
        [COND
           ((IEQP TEMP TYPE-SET-TRUE)
            NIL)
           ((IEQP TEMP TYPE-SET-FALSE)
            (replace (POLY ASSUMPTIONS) of POLY with (LIST FALSE)))
           ((AND HEURISTIC-TYPE-ALIST (IEQP (LET ((TYPE-ALIST HEURISTIC-TYPE-ALIST))
                                                 (TYPE-SET TERM))
                                            TYPE-SET-FALSE))
            (replace (POLY ASSUMPTIONS) of POLY with (LIST FALSE)))
           [(SETQ TEMP-TEMP (for LIT in LITS-THAT-MAY-BE-ASSUMED-FALSE when (COMPLEMENTARYP TERM LIT)
                               do (RETURN LIT)))
            (replace (POLY LEMMAS) of POLY with (ADD-TO-SET-EQ TEMP-TEMP (fetch (POLY LEMMAS) of POLY]
```

```
            (T (replace (POLY ASSUMPTIONS) of POLY with (ADD-TO-SET TERM (fetch (POLY ASSUMPTIONS) of POLY]
         (RETURN POLY])
```

(ADD-LINEAR-TERM

```
[LAMBDA (TERM PARITY POLY)                                  (* kbr: "19-Oct-85 16:31")
  [COND
     ((VARIABLEP TERM)
      (ADD-LINEAR-VARIABLE TERM PARITY POLY))
     [(FQUOTEP TERM)
      (COND
         ((AND (FIXP (CADR TERM))
               (GREATERP (CADR TERM)
                         -1))
          (COND
             [(EQ PARITY (QUOTE POSITIVE))
              (replace (POLY CONSTANT) of POLY with (PLUS (fetch (POLY CONSTANT) of POLY)
                                                          (CADR TERM]
             (T (replace (POLY CONSTANT) of POLY with (DIFFERENCE (fetch (POLY CONSTANT) of POLY)
                                                                  (CADR TERM]
     (T (SELECTQ (FFN-SYMB TERM)
             (ADD1 [replace (POLY CONSTANT) of POLY with (COND
                                                           ((EQ PARITY (QUOTE POSITIVE))
                                                            (ADD1 (fetch (POLY CONSTANT) of POLY)))
                                                           (T (SUB1 (fetch (POLY CONSTANT) of POLY]
                   (ADD-LINEAR-TERM (FARGN TERM 1)
                                    PARITY POLY))
             (ZERO NIL)
             (SUB1 (COND
                      ((EQ PARITY (QUOTE POSITIVE))
                       (replace (POLY CONSTANT) of POLY with (SUB1 (fetch (POLY CONSTANT) of POLY)))
                       (ADD-LINEAR-TERM (FARGN TERM 1)
                                        PARITY POLY))
                      (T (ADD-NOT-LESSP-ASSUMPTION-TO-POLY (FARGN TERM 1)
                                                           (QUOTE (QUOTE 1))
                                                           POLY)
                         (replace (POLY CONSTANT) of POLY with (ADD1 (fetch (POLY CONSTANT) of POLY)))
                         (ADD-LINEAR-TERM (FARGN TERM 1)
                                          PARITY POLY))))
             (PLUS (ADD-LINEAR-TERM (FARGN TERM 2)
                                    PARITY POLY)
                   (ADD-LINEAR-TERM (FARGN TERM 1)
                                    PARITY POLY))
             (DIFFERENCE (COND
                            ((EQ PARITY (QUOTE POSITIVE))
                             (ADD-LINEAR-TERM (FARGN TERM 2)
                                              (QUOTE NEGATIVE)
                                              POLY)
                             (ADD-LINEAR-TERM (FARGN TERM 1)
                                              PARITY POLY))
                            (T (ADD-NOT-LESSP-ASSUMPTION-TO-POLY (FARGN TERM 1)
                                                                 (FARGN TERM 2)
                                                                 POLY)
                               (ADD-LINEAR-TERM (FARGN TERM 2)
                                                (QUOTE POSITIVE)
                                                POLY)
                               (ADD-LINEAR-TERM (FARGN TERM 1)
                                                PARITY POLY))))
             (ADD-LINEAR-VARIABLE TERM PARITY POLY]
   POLY])
```

(ADD-LINEAR-VARIABLE

```
[LAMBDA (VAR PARITY POLY)                                   (* kbr: "19-Oct-85 16:31")
  (LET (N TERM)
       [COND
          [(AND (BM-MATCH VAR (TIMES N TERM))
                (QUOTEP N)
                (FIXP (CADR N))
                (GREATERP (CADR N)
                          -1))
           (COND
              ((LOGSUBSETP TYPE-SET-NUMBERS (TYPE-SET TERM))
               (replace (POLY ALIST) of POLY with (ADD-LINEAR-VARIABLE1 (CADR N)
                                                                        TERM PARITY (fetch (POLY ALIST) of POLY]
              ((LOGSUBSETP TYPE-SET-NUMBERS (TYPE-SET VAR))
               (replace (POLY ALIST) of POLY with (ADD-LINEAR-VARIABLE1 1 VAR PARITY (fetch (POLY ALIST) of POLY]
       POLY])
```

(ADD-LINEAR-VARIABLE1

```
[LAMBDA (N VAR PARITY ALIST)                                (* kbr: "20-Oct-85 15:47")
  (COND
     ((NLISTP ALIST)
      (CONS [CONS VAR (COND
                         ((EQ PARITY (QUOTE POSITIVE))
                          N)
```

```
                          (T (MINUS N]
               NIL))
       [(TERM-ORDER VAR (CAAR ALIST))
        (COND
           ((EQUAL VAR (CAAR ALIST))
            [COND
               [(EQ PARITY (QUOTE POSITIVE))
                (RPLACD (CAR ALIST)
                        (PLUS N (CDR (CAR ALIST]
               (T (RPLACD (CAR ALIST)
                          (DIFFERENCE (CDR (CAR ALIST))
                                 N]
            ALIST)
           (T (RPLACD ALIST (ADD-LINEAR-VARIABLE1 N VAR PARITY (CDR ALIST]
       (T (CONS [CONS VAR (COND
                             ((EQ PARITY (QUOTE POSITIVE))
                              N)
                             (T (MINUS N]
                ALIST])
```

(ADD-LITERAL

```
[LAMBDA (LIT CL AT-END-FLG)                                    (* kbr: "19-Oct-85 16:31")
```

(* We assume that LIT has been subjected to NEGATE-LIT or PEGATE-LIT before passed to ADD-LITERAL, and that CL is the result of previous such ADD-LITERALS. Thus, we make the trivial checks that LIT is neither T nor F, but do not use a full blown FALSE-NONFALSEP. *)

```
   (COND
      ((EQUAL LIT FALSE)
       CL)
      ((EQUAL LIT TRUE)
       TRUE-CLAUSE)
      ((EQUAL CL TRUE-CLAUSE)
       TRUE-CLAUSE)
      ((for LIT2 in CL thereis (COMPLEMENTARYP LIT LIT2))
       TRUE-CLAUSE)
      ((MEMBER LIT CL)
       CL)
      (AT-END-FLG (APPEND CL (LIST LIT)))
      (T (CONS LIT CL])
```

(ADD-META-LEMMA

```
[LAMBDA (NAME TYPE TERM)                                       (* kbr: "19-Oct-85 16:31")
   (LET (FN)
        (BM-MATCH TERM (IMPLIES & (AND (EQUAL & (MEANING (LIST FN &)
                                                         &))
                                       &)))
        (for X in (CDR TYPE) do (ADD-FACT X (QUOTE LEMMAS)
                                          (create REWRITE-RULE
                                                  NAME _ NAME
                                                  CONCL _ (GETPROP FN (QUOTE LISP-CODE])
```

(ADD-NOT-EQUAL-0-ASSUMPTION-TO-POLY

```
[LAMBDA (TERM POLY)                                            (* kbr: "19-Oct-85 16:31")
                                                               (* We add the assumption (NOT (EQUAL TERM 0)) to POLY.
                                                               See the comment in
                                                               ADD-NUMBERP-ASSUMPTION-TO-POLY.
                                                               *)
   (LET (X Y TEMP EQUALITY)
        (COND
           ((BM-MATCH TERM (DIFFERENCE X Y))
            (ADD-LESSP-ASSUMPTION-TO-POLY Y X POLY))
           ((EQUAL TERM ZERO)
            (replace (POLY ASSUMPTIONS) of POLY with (LIST FALSE))
            POLY)
           ([OR (BM-MATCH TERM (ADD1 &))
                (AND (QUOTEP TERM)
                     (NOT (EQUAL (CADR TERM)
                                 0]
            POLY)
           (T (SETQ EQUALITY (FCONS-TERM* (QUOTE EQUAL)
                                          TERM ZERO))
              (SETQ TEMP (TYPE-SET EQUALITY))
              [COND
                 ((IEQP TEMP TYPE-SET-TRUE)
                  (replace (POLY ASSUMPTIONS) of POLY with (LIST FALSE)))
                 ((IEQP TEMP TYPE-SET-FALSE)
                  NIL)
                 ((AND HEURISTIC-TYPE-ALIST (IEQP (LET ((TYPE-ALIST HEURISTIC-TYPE-ALIST))
                                                       (TYPE-SET EQUALITY))
                                                  TYPE-SET-TRUE))
                  (replace (POLY ASSUMPTIONS) of POLY with (LIST FALSE)))
                 [(SETQ TEMP-TEMP (MEMBER EQUALITY LITS-THAT-MAY-BE-ASSUMED-FALSE))
                  (replace (POLY LEMMAS) of POLY with (ADD-TO-SET-EQ (CAR TEMP-TEMP)
```

```
                                                                (fetch (POLY LEMMAS) of POLY]
                (T (replace (POLY ASSUMPTIONS) of POLY with (ADD-TO-SET (FCONS-TERM* (QUOTE NOT)
                                                                                     EQUALITY)
                                                                        (fetch (POLY ASSUMPTIONS) of POLY]
            POLY])
```

(ADD-NOT-LESSP-ASSUMPTION-TO-POLY

```
  [LAMBDA (X Y POLY)                                                    (* kbr: "19-Oct-85 16:31")
                                                                        (* We add the assumption (NOT (LESSP X Y)) to POLY.
                                                                        See the comment in
                                                                        ADD-NUMBERP-ASSUMPTION-TO-POLY.
                                                                        *)
    (PROG (TEMP TERM)
          [COND
             ((EQUAL Y (QUOTE (QUOTE 1)))
              (COND
                 ((IEQP (TYPE-SET X)
                        TYPE-SET-NUMBERS)
                  (RETURN (ADD-NOT-EQUAL-0-ASSUMPTION-TO-POLY X POLY)))
                 ((SETQ TEMP-TEMP (for LIT in LITS-THAT-MAY-BE-ASSUMED-FALSE
                                     bind (TERM _ (FCONS-TERM* (QUOTE NUMBERP)
                                                               X))
                                     when (COMPLEMENTARYP TERM LIT) do (RETURN LIT)))
                  (replace (POLY LEMMAS) of POLY with (ADD-TO-SET-EQ TEMP-TEMP (fetch (POLY LEMMAS) of POLY)))
                  (RETURN (ADD-NOT-EQUAL-0-ASSUMPTION-TO-POLY X POLY]
          [SETQ TEMP (TYPE-SET (SETQ TERM (FCONS-TERM* (QUOTE LESSP)
                                                       X Y]
          [COND
             ((IEQP TEMP TYPE-SET-FALSE)
              NIL)
             ((IEQP TEMP TYPE-SET-TRUE)
              (replace (POLY ASSUMPTIONS) of POLY with (LIST FALSE)))
             ((AND HEURISTIC-TYPE-ALIST (IEQP (LET ((TYPE-ALIST HEURISTIC-TYPE-ALIST))
                                                (TYPE-SET TERM))
                                              TYPE-SET-TRUE))
              (replace (POLY ASSUMPTIONS) of POLY with (LIST FALSE)))
             [(SETQ TEMP-TEMP (MEMBER TERM LITS-THAT-MAY-BE-ASSUMED-FALSE))
              (replace (POLY LEMMAS) of POLY with (ADD-TO-SET-EQ (CAR TEMP-TEMP)
                                                                 (fetch (POLY LEMMAS) of POLY]
             (T (replace (POLY ASSUMPTIONS) of POLY with (ADD-TO-SET (FCONS-TERM* (QUOTE NOT)
                                                                                  TERM)
                                                                     (fetch (POLY ASSUMPTIONS) of POLY]
          (RETURN POLY])
```

(ADD-NUMBERP-ASSUMPTION-TO-POLY

```
  [LAMBDA (TERM POLY)                                                   (* kbr: "19-Oct-85 16:31")

          (* We add the assumption (NUMBERP TERM) to the assumptions field of POLY but we first check to see if the assumption
          is obviously true or false. We assume TYPE-ALIST is correctly set.
          If the HEURISTIC-TYPE-ALIST is set and says the assumption is false, we add the false assumption --
          this is sound, even though HEURISTIC-TYPE-ALIST may be irrelevant, because we can always add a false assumption to a
          poly which will prevent the poly from being used. We assume that LITS-THAT-MAY-BE-ASSUMED-FALSE is NIL unless we
          are under the ADD-TERMS-TO-POT-LST in SIMPLIFY-CLAUSE0.
          If the complement of the assumption we wish to add is in LITS-THAT-MAY-BE-ASSUMED-FALSE then the assumption is
          true but we record the literal that makes it true in the LEMMAS field of POLY.
          We assume that if (NUMBERP TERM) is in LITS-THAT-MAY-BE-ASSUMED-FALSE then it was false under the
          HEURISTIC-TYPE-ALIST and we do not bother to check. *)

    (LET (TEMP)
         (SETQ TEMP (TYPE-SET TERM))
         [COND
            ((IEQP TEMP TYPE-SET-NUMBERS)
             NIL)
            ((NOT (LOGSUBSETP TYPE-SET-NUMBERS TEMP))
             (replace (POLY ASSUMPTIONS) of POLY with (LIST FALSE)))
            ([AND HEURISTIC-TYPE-ALIST (NOT (LOGSUBSETP TYPE-SET-NUMBERS (LET ((TYPE-ALIST HEURISTIC-TYPE-ALIST)
                                                                               )
                                                                              (TYPE-SET TERM]
                                                                        (* On heuristic grounds, we here decide not to use this poly.
                                                                        *)
             (replace (POLY ASSUMPTIONS) of POLY with (LIST FALSE)))
            (T (SETQ TEMP (FCONS-TERM* (QUOTE NUMBERP)
                                       TERM))
               (COND
                  [(SETQ TEMP-TEMP (for LIT in LITS-THAT-MAY-BE-ASSUMED-FALSE when (COMPLEMENTARYP LIT TEMP)
                                      do (RETURN LIT)))
                   (replace (POLY LEMMAS) of POLY with (ADD-TO-SET-EQ TEMP-TEMP (fetch (POLY LEMMAS) of POLY]
                  (T (replace (POLY ASSUMPTIONS) of POLY with (ADD-TO-SET TEMP (fetch (POLY ASSUMPTIONS)
                                                                                      of POLY]
          POLY])
```

(ADD-PROCESS-HIST

```
  [LAMBDA (PROCESS PARENT PARENT-HIST DESCENDANTS HIST-ENTRY)    (* kbr: "19-Oct-85 16:31")
    (IO PROCESS PARENT PARENT-HIST DESCENDANTS HIST-ENTRY)
```

```
    (CONS (CONS PROCESS (CONS PARENT HIST-ENTRY))
          PARENT-HIST])
```

(ADD-REWRITE-LEMMA

```
  [LAMBDA (NAME TYPE TERM)                                          (* kbr: "19-Oct-85 16:31")
    TYPE
    (for X in (UNPRETTYIFY TERM) bind (LEMMA ALL-VARS-HYPS ALL-VARS-CONCL MAX-TERMS LST TEMP HYPS CONCL)
       do (SETQ HYPS (CAR X))
          (SETQ CONCL (CDR X))
          (COND
             [(SETQ TEMP (ACCEPTABLE-TYPE-PRESCRIPTION-LEMMAP HYPS CONCL))
              (ADD-FACT (CAR TEMP)
                     (QUOTE TYPE-PRESCRIPTION-LST)
                     (CONS NAME (CDR TEMP]
             ([AND (NOT NO-BUILT-IN-ARITH-FLG)
                   (OR (BM-MATCH CONCL (NOT (LESSP & &)))
                       (BM-MATCH CONCL (LESSP & &]
              (SETQ LST (EXTERNAL-LINEARIZE CONCL T))
              (SETQ ALL-VARS-HYPS (ALL-VARS-LST HYPS))
              (SETQ ALL-VARS-CONCL (ALL-VARS CONCL))
              (SETQ MAX-TERMS (for PAIR in (fetch (POLY ALIST) of (CAR (CAR LST)))
                                 when [AND (NVARIABLEP (CAR PAIR))
                                           (SUBSETP ALL-VARS-CONCL (UNIONQ (ALL-VARS (CAR PAIR))
                                                                           ALL-VARS-HYPS))
                                           (for PAIR2 in (fetch (POLY ALIST) of (CAR (CAR LST)))
                                              when (NEQ PAIR2 PAIR)
                                              never (AND (LESSP (FORM-COUNT (CAR PAIR))
                                                                (FORM-COUNT (CAR PAIR2)))
                                                         (SUBBAGP (ALL-VARS-BAG (CAR PAIR))
                                                                  (ALL-VARS-BAG (CAR PAIR2]
                                 collect (CAR PAIR)))
              (for TERM in MAX-TERMS do (SETQ LEMMA (create LINEAR-LEMMA
                                                            NAME _ NAME
                                                            HYPS _ (PREPROCESS-HYPS HYPS)
                                                            CONCL _ CONCL
                                                            MAX-TERM _ TERM))
                                        (ADD-FACT (FN-SYMB TERM)
                                                  (QUOTE LINEAR-LEMMAS)
                                                  LEMMA)))
              (T (for REWRITE-RULE in (MAKE-REWRITE-RULES NAME HYPS CONCL) do (ADD-FACT (TOP-FNNAME CONCL)
                                                                                        (QUOTE LEMMAS)
                                                                                        REWRITE-RULE])
```

(ADD-SHELL-ROUTINES

```
  [LAMBDA (SHELL-NAME BTM-FN-SYMB RECOGNIZER DESTRUCTOR-TUPLES)  (* kbr: "20-Oct-85 19:45")
    (PROG NIL
          (COND
             (IN-BOOT-STRAP-FLG [for NAME in [CONS SHELL-NAME (CONS RECOGNIZER (APPEND (for X in DESTRUCTOR-TUPLES
                                                                                          collect (CAR X))
                                                                                      (COND
                                                                                         (BTM-FN-SYMB
                                                                                          (LIST BTM-FN-SYMB))
                                                                                         (T NIL]
                                                do (ADD-FACT NAME (QUOTE LISP-CODE)
                                                             (PACK (LIST STRING-WEIRD NAME]
                                (RETURN NIL)))
          [ADD-DCELL
           SHELL-NAME
           (PACK (LIST STRING-WEIRD SHELL-NAME))
           (LIST
            (QUOTE LAMBDA)
            (for X in DESTRUCTOR-TUPLES collect (CAR X))
            (CONS (QUOTE LIST)
                  (CONS (QUOTE *1*SHELL-QUOTE-MARK)
                        (CONS (LIST (QUOTE QUOTE)
                                    SHELL-NAME)
                              (for TUPLE in DESTRUCTOR-TUPLES bind TEMP
                                 collect (PROGN [SETQ TEMP
                                                      (CONS (QUOTE OR)
                                                            (for R in (CDR (CADR TUPLE))
                                                               collect (LIST (QUOTE EQ)
                                                                             (QUOTE *1*T)
                                                                             (LIST (PACK (LIST STRING-WEIRD R))
                                                                                   (CAR TUPLE]
                                                (LIST (QUOTE COND)
                                                      (LIST (COND
                                                               ((EQ (CAR (CADR TUPLE))
                                                                    (QUOTE ONE-OF))
                                                                TEMP)
                                                               (T (LIST (QUOTE NOT)
                                                                        TEMP)))
                                                            (CAR TUPLE))
                                                      (LIST T (LIST (PACK (LIST STRING-WEIRD (CADDR TUPLE]
          [AND BTM-FN-SYMB (ADD-DCELL BTM-FN-SYMB (PACK (LIST STRING-WEIRD BTM-FN-SYMB))
                              (SUB-PAIR (QUOTE (*1*SHELL-QUOTE-MARK BTM))
```

```
                                    (LIST *1*SHELL-QUOTE-MARK BTM-FN-SYMB)
                                    (QUOTE (LAMBDA NIL (QUOTE (*1*SHELL-QUOTE-MARK BTM]
      [ADD-DCELL RECOGNIZER (PACK (LIST STRING-WEIRD RECOGNIZER))
             (COND
                [BTM-FN-SYMB (SUB-PAIR (QUOTE (SHELL-NAME BTM))
                                       (LIST SHELL-NAME BTM-FN-SYMB)
                                       (QUOTE (LAMBDA (X)
                                                      (COND ([AND (LISTP X)
                                                                  (EQ (CAR X)
                                                                      *1*SHELL-QUOTE-MARK)
                                                                  (OR (EQ (CADR X)
                                                                          (QUOTE SHELL-NAME))
                                                                      (EQ (CADR X)
                                                                          (QUOTE BTM]
                                                             *1*T)
                                                            (T *1*F]
                (T (SUB-PAIR (QUOTE (SHELL-NAME))
                             (LIST SHELL-NAME)
                             (QUOTE (LAMBDA (X)
                                            (COND ((AND (LISTP X)
                                                        (EQ (CAR X)
                                                            *1*SHELL-QUOTE-MARK)
                                                        (EQ (CADR X)
                                                            (QUOTE SHELL-NAME)))
                                                   *1*T)
                                                  (T *1*F]
      [for TUPLE in DESTRUCTOR-TUPLES as I from 2
         do (ADD-DCELL (CAR TUPLE)
                       (PACK (LIST STRING-WEIRD (CAR TUPLE)))
                       (SUB-PAIR (QUOTE (R CELL DV BTM))
                                 (LIST (PACK (LIST STRING-WEIRD RECOGNIZER))
                                       (CELL I (QUOTE X))
                                       (PACK (LIST STRING-WEIRD (CADDR TUPLE)))
                                      BTM-FN-SYMB)
                                 (COND
                                    [BTM-FN-SYMB (QUOTE (LAMBDA (X)
                                                                (COND ((AND (EQ (R X)
                                                                                *1*T)
                                                                            (NEQ (CADR X)
                                                                                 (QUOTE BTM)))
                                                                       (CAR CELL))
                                                                      (T (DV]
                                    (T (QUOTE (LAMBDA (X)
                                                      (COND ((EQ (R X)
                                                                 *1*T)
                                                             (CAR CELL))
                                                            (T (DV]
      (RETURN NIL])
```

(ADD-SHELL0

```
 [LAMBDA (SHELL-NAME BTM-FN-SYMB RECOGNIZER DESTRUCTOR-TUPLES)  (* kbr: "26-Nov-85 15:28")
   (LET (DEST-EXPRS-X SHELL-EXPR CURRENT-TYPE-NO DESTRUCTOR-ALIST RENAMED-SHELL-EXPR DESTRUCTOR-NAMES DV TERM
         NEW-TYPE-NO NAMES DEST-NAME ARG-NAME)
      (SETQ NEW-TYPE-NO (NEXT-AVAILABLE-TYPE-NO))
      (SETQ DESTRUCTOR-NAMES (for TUPLE in DESTRUCTOR-TUPLES collect (CAR TUPLE)))
      (ADD-SHELL-ROUTINES SHELL-NAME BTM-FN-SYMB RECOGNIZER DESTRUCTOR-TUPLES)
      [SETQ DESTRUCTOR-ALIST (for X in DESTRUCTOR-TUPLES collect (CONS (CAR X)
                                                                       (MAKE-TYPE-RESTRICTION (CADR X)
                                                                              (CADDR X)
                                                                              RECOGNIZER NEW-TYPE-NO]
      (ADD-TYPE-SET-LEMMAS SHELL-NAME BTM-FN-SYMB RECOGNIZER DESTRUCTOR-ALIST)
      [COND
         (DESTRUCTOR-NAMES
          (SETQ SHELL-EXPR (CONS-TERM SHELL-NAME DESTRUCTOR-NAMES))
          [for PAIR in DESTRUCTOR-ALIST
             do (SETQ DEST-NAME (CAR PAIR))
                (SETQ ARG-NAME DEST-NAME)
                (SETQ TERM (fetch (TYPE-RESTRICTION TERM) of (CDR PAIR)))
                (SETQ DV (fetch (TYPE-RESTRICTION DEFAULT) of (CDR PAIR)))
                [ADD-AXIOM1 (PACK (LIST DEST-NAME "-" SHELL-NAME))
                       (QUOTE (REWRITE))
                       (FCONS-TERM* (QUOTE EQUAL)
                              (FCONS-TERM* DEST-NAME SHELL-EXPR)
                              (COND
                                 ((EQUAL TERM TRUE)
                                  ARG-NAME)
                                 (T (FCONS-TERM* (QUOTE IF)
                                           (SUBST-VAR ARG-NAME (QUOTE X)
                                                  TERM)
                                          ARG-NAME DV]
                (ADD-AXIOM1 (PACK (LIST DEST-NAME (QUOTE -N)
                                        RECOGNIZER))
                       (QUOTE (REWRITE))
                       (FCONS-TERM* (QUOTE IMPLIES)
                              (FCONS-TERM* (QUOTE NOT)
                                     (FCONS-TERM* RECOGNIZER (QUOTE X)))
```

```
                        (FCONS-TERM* (QUOTE EQUAL)
                               (FCONS-TERM* DEST-NAME (QUOTE X))
                               DV)))
                [AND (NOT (EQUAL TERM TRUE))
                     (ADD-AXIOM1 (PACK (LIST DEST-NAME (QUOTE -TYPE-RESTRICTION)))
                            (QUOTE (REWRITE))
                            (FCONS-TERM* (QUOTE IMPLIES)
                                   (BM-NEGATE (SUBST-VAR DEST-NAME (QUOTE X)
                                                         TERM))
                                   (FCONS-TERM* (QUOTE EQUAL)
                                          SHELL-EXPR
                                          (SUBST-VAR DV DEST-NAME SHELL-EXPR]
                [ADD-AXIOM1 (PACK (LIST DEST-NAME (QUOTE -LESSP)))
                       (QUOTE (REWRITE))
                       (FCONS-TERM* (QUOTE IMPLIES)
                              [COND
                                 [BTM-FN-SYMB (FCONS-TERM* (QUOTE AND)
                                                     (FCONS-TERM* RECOGNIZER (QUOTE X))
                                                     (FCONS-TERM* (QUOTE NOT)
                                                            (FCONS-TERM* (QUOTE EQUAL)
                                                                   (QUOTE X)
                                                                   (CONS-TERM BTM-FN-SYMB NIL]
                                 (T (FCONS-TERM* RECOGNIZER (QUOTE X]
                              (FCONS-TERM* (QUOTE LESSP)
                                     (FCONS-TERM* (QUOTE COUNT)
                                            (FCONS-TERM* DEST-NAME (QUOTE X)))
                                     (FCONS-TERM* (QUOTE COUNT)
                                            (QUOTE X]
                (ADD-AXIOM1 (PACK (LIST DEST-NAME (QUOTE -LESSEQP)))
                       (QUOTE (REWRITE))
                       (FCONS-TERM* (QUOTE NOT)
                              (FCONS-TERM* (QUOTE LESSP)
                                     (FCONS-TERM* (QUOTE COUNT)
                                            (QUOTE X))
                                     (FCONS-TERM* (QUOTE COUNT)
                                            (FCONS-TERM* DEST-NAME (QUOTE X]
        [SETQ RENAMED-SHELL-EXPR (CONS-TERM SHELL-NAME (for DEST in DESTRUCTOR-NAMES
                                                          collect (PACK (LIST DEST "-"]
        (ADD-AXIOM1 (PACK (LIST SHELL-NAME "-EQUAL"))
               (QUOTE (REWRITE))
               (FCONS-TERM* (QUOTE EQUAL)
                      (FCONS-TERM* (QUOTE EQUAL)
                             SHELL-EXPR RENAMED-SHELL-EXPR)
                      (CONJOIN [for ARG1 in (FARGS SHELL-EXPR) as ARG2 in (FARGS RENAMED-SHELL-EXPR)
                                  as PAIR in DESTRUCTOR-ALIST
                                  collect (PROGN (SETQ TERM (fetch (TYPE-RESTRICTION TERM) of (CDR PAIR)))
                                                 (SETQ DV (fetch (TYPE-RESTRICTION DEFAULT)
                                                                 of (CDR PAIR)))
                                                 (COND
                                                    ((EQUAL TERM TRUE)
                                                     (FCONS-TERM* (QUOTE EQUAL)
                                                            ARG1 ARG2))
                                                    (T (FCONS-TERM* (QUOTE IF)
                                                              (SUBST-VAR ARG1 (QUOTE X)
                                                                     TERM)
                                                              (FCONS-TERM* (QUOTE IF)
                                                                     (SUBST-VAR ARG2 (QUOTE X)
                                                                            TERM)
                                                                     (FCONS-TERM* (QUOTE EQUAL)
                                                                            ARG1 ARG2)
                                                                     (FCONS-TERM* (QUOTE EQUAL)
                                                                            ARG1 DV))
                                                              (FCONS-TERM* (QUOTE IF)
                                                                     (SUBST-VAR ARG2 (QUOTE X)
                                                                            TERM)
                                                                     (FCONS-TERM* (QUOTE EQUAL)
                                                                            DV ARG2)
                                                                     TRUE]
                                  NIL)))
        [SETQ DEST-EXPRS-X (for DEST-NAME in DESTRUCTOR-NAMES collect (FCONS-TERM* DEST-NAME (QUOTE X]
        [ADD-AXIOM1 [PACK (CONS SHELL-NAME (for DEST-NAME in DESTRUCTOR-NAMES join (LIST "-" DEST-NAME]
               (QUOTE (REWRITE))
               (FCONS-TERM* (QUOTE EQUAL)
                      (CONS-TERM SHELL-NAME DEST-EXPRS-X)
                      (FCONS-TERM* (QUOTE IF)
                             [COND
                                [BTM-FN-SYMB (FCONS-TERM* (QUOTE AND)
                                                    (FCONS-TERM* RECOGNIZER (QUOTE X))
                                                    (FCONS-TERM* (QUOTE NOT)
                                                           (FCONS-TERM* (QUOTE EQUAL)
                                                                  (QUOTE X)
                                                                  (CONS-TERM BTM-FN-SYMB NIL]
                                (T (FCONS-TERM* RECOGNIZER (QUOTE X]
                             (QUOTE X)
                             (CONS-TERM SHELL-NAME (for X in DESTRUCTOR-ALIST
                                                      collect (fetch (TYPE-RESTRICTION DEFAULT)
                                                                     of (CDR X]
```

```
        [ADD-AXIOM1 (PACK (NCONC1 (CDR (for DEST-NAME in DESTRUCTOR-NAMES join (LIST "-" DEST-NAME)))
                                  "-ELIM"))
                (QUOTE (ELIM))
                (FCONS-TERM* (QUOTE IMPLIES)
                        [COND
                           [BTM-FN-SYMB (FCONS-TERM* (QUOTE AND)
                                                     (FCONS-TERM* RECOGNIZER (QUOTE X))
                                                     (FCONS-TERM* (QUOTE NOT)
                                                                  (FCONS-TERM* (QUOTE EQUAL)
                                                                               (QUOTE X)
                                                                               (CONS-TERM BTM-FN-SYMB NIL]
                           (T (FCONS-TERM* RECOGNIZER (QUOTE X]
                        (FCONS-TERM* (QUOTE EQUAL)
                                (CONS-TERM SHELL-NAME DEST-EXPRS-X)
                                (QUOTE X]
        (ADD-AXIOM1 (PACK (LIST (QUOTE COUNT-)
                                SHELL-NAME))
                (QUOTE (REWRITE))
                (FCONS-TERM* (QUOTE EQUAL)
                        (FCONS-TERM* (QUOTE COUNT)
                                SHELL-EXPR)
                        (FCONS-TERM* (QUOTE ADD1)
                                (PLUSJOIN (for X in (FARGS SHELL-EXPR) as PAIR in DESTRUCTOR-ALIST
                                                collect (PROGN (SETQ TERM (fetch (TYPE-RESTRICTION TERM)
                                                                                 of (CDR PAIR)))
                                                               (SETQ DV (fetch (TYPE-RESTRICTION DEFAULT)
                                                                               of (CDR PAIR)))
                                                               (COND
                                                                  ((EQUAL TERM TRUE)
                                                                   (FCONS-TERM* (QUOTE COUNT)
                                                                                X))
                                                                  (T (FCONS-TERM* (QUOTE IF)
                                                                                  (SUBST-VAR X (QUOTE X)
                                                                                             TERM)
                                                                                  (FCONS-TERM* (QUOTE COUNT)
                                                                                               X)
                                                                                  ZERO]
    SHELL-NAME])
```

(ADD-SUB-FACT

```
  [LAMBDA (ATM PROP VAL TUPLE INIT)                    (* kbr: "19-Oct-85 16:31")
```

(* Here is the spec for ADD-SUB-FACT. It takes 5 args ATM PROP VAL TUPLE and INIT but only a few of these make sense in combination. To store a new fact you call ADD-SUB-FACT using ATM PROP and VAL.
If PROP is a variable declared below we either CONS VAL to the front of PROPs top level value or set PROP to VAL depending on whether PROP is ADDITIVE or SINGLE. SET is used in both cases.
If PROP is DCELL it means PUTD1 ATM to VAL. Otherwise, PROP had better be an additive or single property declared below and if so the appropriate ADDITIVE or SINGLE PUT1 is done.
If you want to delete a previously added fact you call ADD-SUB-FACT with all args but TUPLE NIL.
TUPLE should be the undo tuple produced by the adding of the fact in question.
Before you begin to add or sub facts you must first initialize the library file.
You do that by calling ADD-SUB-FACT with INIT T and all other args NIL.
The initialization sets LIB-PROPS to the list of properties declared below in the reverse order of declaration -- making the first property declared the one of highest priority. Because the list of declarations is used to generated LIB-PROPS you must include in it all of the properties used by the event level abstraction itself, even those these properties aren't technically yours. These properties are IDATE SATELLITES MAIN-EVENT EVENT and LOCAL-UNDO-TUPLES.
They should all be declared with type HIDDEN rather than ADDITIVE or SINGLE.
The code will cause an error if you leave out any built-in prop or use HIDDEN on any nonbuilt-in one -- the whole purpose of your knowing about these properties and the token HIDDEN is just to allow you to specify where in the list of priorities they should be kept. The other thing that initialization does is set all variables declared below to NIL.
The HIDDEN variable CHRONOLOGY should be declared explicitly.
We force you to do that so you'll always remember we've claimed that variable name.
No property or variable name may contain lower case letters or be NIL.
If this convention is violated the code produced for ADD-SUB-FACT is garbage because we generate the code with SUBST's that hit lower case names and we sometimes generate SELECTQs with NIL first elements of clauses.
For ADDITIVE data you must supply a form, which may involve VAL as a free var, for computing from VAL some datum to be stored in the undo tuple. This datum must be sufficient for distinguishing that VAL from all others in that ADDITIVE pot. In particular, to find the VAL in question the undoing mechanism scans the pot and evaluates the form again for each entry, with VAL bound to the entry, and removes from the pot the first entry for which that form computes an EQUAL datum.
The form in question must not contain any free variables other than VAL and must not cause any side-effects.
*)

```
    (ADD-SUB-FACT-BODY (TYPE-PRESCRIPTION-LST ADDITIVE PROPERTY (CAR VAL))
          (LEMMAS ADDITIVE PROPERTY (fetch (REWRITE-RULE NAME) of VAL))
          (LINEAR-LEMMAS ADDITIVE PROPERTY (fetch (LINEAR-LEMMA NAME) of VAL))
          (QUICK-BLOCK-INFO SINGLE PROPERTY)
          (SDEFN SINGLE PROPERTY)
          (LISP-CODE SINGLE PROPERTY)
          (TYPE-RESTRICTIONS SINGLE PROPERTY)
          (INDUCTION-MACHINE SINGLE PROPERTY)
          (LEVEL-NO SINGLE PROPERTY)
          (JUSTIFICATIONS SINGLE PROPERTY)
          (IDATE HIDDEN PROPERTY)
          (ELIMINATE-DESTRUCTORS-SEQ SINGLE PROPERTY)
          (ELIMINATE-DESTRUCTORS-DESTS SINGLE PROPERTY)
          (CONTROLLER-POCKETS SINGLE PROPERTY)
```

```
          (SATELLITES HIDDEN PROPERTY)
          (MAIN-EVENT HIDDEN PROPERTY)
          (IMMEDIATE-DEPENDENTS0 ADDITIVE PROPERTY VAL)
          (EVENT HIDDEN PROPERTY)
          (LOCAL-UNDO-TUPLES HIDDEN PROPERTY)
          (NONCONSTRUCTIVE-AXIOM-NAMES ADDITIVE VARIABLE VAL)
          (*1*BTM-OBJECTS ADDITIVE VARIABLE VAL)
          (RECOGNIZER-ALIST ADDITIVE VARIABLE VAL)
          (SHELL-ALIST ADDITIVE VARIABLE VAL)
          (SINGLETON-TYPE-SETS ADDITIVE VARIABLE VAL)
          (GENERALIZE-LEMMAS ADDITIVE VARIABLE (fetch (GENERALIZE-LEMMA NAME) of VAL))
          (SHELL-POCKETS ADDITIVE VARIABLE VAL)
          (DISABLED-LEMMAS ADDITIVE VARIABLE VAL)
          (CHRONOLOGY HIDDEN VARIABLE])
```

(ADD-TERM-TO-POT-LST

```
  [LAMBDA (TERM POT-LST FLG ALL-NEW-FLG)                    (* kbr: "19-Oct-85 16:31")
    (PROG NIL
          [COND
             [(EQ CURRENT-LIT CURRENT-ATM)
              (COND
                 ((AND (EQ FLG NIL)
                       (EQUAL TERM CURRENT-LIT))
                  (RETURN POT-LST]
             (T (COND
                   ((AND FLG (EQUAL TERM CURRENT-ATM))
                    (RETURN POT-LST]
          (RPLACA ADD-TERM-TO-POT-LST-TEMP TERM)
          (RETURN (ADD-TERMS-TO-POT-LST ADD-TERM-TO-POT-LST-TEMP POT-LST FLG ALL-NEW-FLG])
```

(ADD-TERMS-TO-POT-LST

```
  [LAMBDA (TERM-LST POT-LST FLG ALL-NEW-FLG)                (* kbr: "20-Oct-85 17:30")
```

(* Only called with POT-LST EQ to SIMPLIFY-CLAUSE-POT-LST.
Either returns (QUOTE CONTRADICTION,) in which case there is a proof of F from TYPE-ALIST, the assumption of the members of TERM-LST true or false according as FLG is T or NIL, LINEAR-ASSUMPTIONS, and a subset S of the polys in POT-LST such that if ITIMES IEQP (LIST (QUOTE MARK)) is a MEMB of the LEMMAS of a member of S then ITIMES is in LEMMAS-USED-BY-LINEAR, or returns a new pot lst such that for each poly p in the new pot lst there is a proof of p from TYPE-ALIST, the assumption of the members of TERM-LST true or false according as FLG is T or NIL, and a subset S of the polys in the input POT-LST such that if ITIMES IEQP (LIST
(QUOTE MARK)) is a MEMB of the lemmas of a member of S, then ITIMES is in the LEMMAS field of p.
In no case is the lemma stack or linearize assumptions stack visibly affected by this call.
Not necessary for soundness, but true, are the facts that the lemmas
(ignoring typeset lemmas, of course) that are used in the proofs are included in the LEMMAS fields.
Furthermore, the LITERALS fields contain the literals that were passed in TERM-LST to ADD-TERMS-TO-POT-LST and used to construct, with LINEARIZE, the original polynomials. If ALL-NEW-FLG is T then every addend in the pot list is treated as new for the consideration of lemmas to be added. Otherwise, we add lemmas for the addends that are introduced by this call. *)

```
    (PROG (POLY-LST SPLIT-LST LST BASIC-POT-LST UNIFY-SUBST POT-LST1 POT-LST2)
          (COND
             (NO-BUILT-IN-ARITH-FLG (RETURN NIL)))
          [for TERM in TERM-LST
             do (SETQ LST (LINEARIZE TERM FLG))
                (COND
                   ((NULL LST))
                   ((NULL (CDR LST))
                    (SETQ POLY-LST (APPEND (CAR LST)
                                           POLY-LST)))
                   ((NULL (CDDR LST))
                    (SETQ SPLIT-LST (CONS LST SPLIT-LST)))
                   (T (ERROR1 (PQUOTE (PROGN LINEARIZE RETURNED A LIST WITH MORE THAN 2 ELEMENTS !))
                              NIL
                              (QUOTE HARD]
          (SETQ BASIC-POT-LST (ADD-EQUATIONS-TO-POT-LST POLY-LST POT-LST ALL-NEW-FLG))
          [for PAIR in SPLIT-LST bind (MARK _ (LIST (QUOTE MARK))) while (NEQ BASIC-POT-LST (QUOTE CONTRADICTION))
             do
```

(* We will add both branches separately and hope at least one gives a contradiction.
Suppose the first branch does not but the second does. Then we will use the first branch's pot list in the future.
But we must add to the assumptions and lemmas of the first branch those of the second.
To recognize the polys in the first branch's pot lst that descend from the polys in the first branch we will mark them by putting a unique CONS in the lemmas field. *)

```
                (for POLY in (CAR PAIR) do (replace (POLY LEMMAS) of POLY with (LIST MARK)))
                (SETQ POT-LST1 (ADD-EQUATIONS-TO-POT-LST (CAR PAIR)
                                                         BASIC-POT-LST ALL-NEW-FLG))
                (COND
                   ((EQ POT-LST1 (QUOTE CONTRADICTION))
                    [for POLY in (CADR PAIR) do (replace (POLY LEMMAS) of POLY with (REMQ MARK
                                                                                          LEMMAS-USED-BY-LINEAR))
                                                (replace (POLY ASSUMPTIONS) of POLY
                                                   with (UNION-EQUAL LINEAR-ASSUMPTIONS (fetch (POLY ASSUMPTIONS)
                                                                                                of POLY]
                    (SETQ BASIC-POT-LST (ADD-EQUATIONS-TO-POT-LST (CADR PAIR)
```

```
                                                BASIC-POT-LST ALL-NEW-FLG)))
                        (T (SETQ POT-LST2 (ADD-EQUATIONS-TO-POT-LST (CADR PAIR)
                                                    BASIC-POT-LST ALL-NEW-FLG))
                           (COND
                              ((EQ POT-LST2 (QUOTE CONTRADICTION))
                               [for POT in POT-LST1 do [for POLY in (fetch (LINEAR-POT POSITIVES) of POT)
                                                          when (MEMB MARK (fetch (POLY LEMMAS) of POLY))
                                                          do (replace (POLY ASSUMPTIONS) of POLY
                                                                with (UNION-EQUAL LINEAR-ASSUMPTIONS
                                                                             (fetch (POLY ASSUMPTIONS) of POLY)))
                                                             (replace (POLY LEMMAS) of POLY
                                                                with (UNIONQ LEMMAS-USED-BY-LINEAR
                                                                             (REMQ MARK (fetch (POLY LEMMAS)
                                                                                           of POLY]
                                                       (for POLY in (fetch (LINEAR-POT NEGATIVES) of POT)
                                                          when (MEMB MARK (fetch (POLY LEMMAS) of POLY))
                                                          do (replace (POLY ASSUMPTIONS) of POLY
                                                                with (UNION-EQUAL LINEAR-ASSUMPTIONS
                                                                             (fetch (POLY ASSUMPTIONS) of POLY)))
                                                             (replace (POLY LEMMAS) of POLY
                                                                with (UNIONQ LEMMAS-USED-BY-LINEAR
                                                                             (REMQ MARK (fetch (POLY LEMMAS)
                                                                                           of POLY]
                                 (SETQ BASIC-POT-LST POT-LST1]
         (RETURN BASIC-POT-LST])
```

(ADD-TO-SET-EQ

```
[LAMBDA (X LST)                                                    (* kbr: "19-Oct-85 16:31")
  (COND
     ((MEMB X LST)
      LST)
     (T (CONS X LST])
```

(ADD-TYPE-SET-LEMMAS

```
[LAMBDA (SHELL-NAME BTM-FN-SYMB RECOGNIZER DESTRUCTOR-ALIST)   (* kbr: "19-Oct-85 16:31")
  (LET (CURRENT-TYPE-NO)
       (SETQ CURRENT-TYPE-NO (NEXT-AVAILABLE-TYPE-NO))
       (ADD-FACT NIL (QUOTE SHELL-ALIST)
              (CONS SHELL-NAME CURRENT-TYPE-NO))
       [ADD-FACT NIL (QUOTE SHELL-POCKETS)
              (CONS SHELL-NAME (for X in DESTRUCTOR-ALIST collect (CAR X]
       [ADD-FACT SHELL-NAME (QUOTE TYPE-PRESCRIPTION-LST)
              (CONS SHELL-NAME (CONS (LOGBIT CURRENT-TYPE-NO)
                                     (for X in DESTRUCTOR-ALIST collect NIL]
       [AND DESTRUCTOR-ALIST (ADD-FACT SHELL-NAME (QUOTE TYPE-RESTRICTIONS)
                                   (for X in DESTRUCTOR-ALIST collect (CDR X]
       [COND
          ((AND (NULL DESTRUCTOR-ALIST)
                (NULL BTM-FN-SYMB))
           (ADD-FACT NIL (QUOTE SINGLETON-TYPE-SETS)
                  (LOGBIT CURRENT-TYPE-NO]
       (AND BTM-FN-SYMB (ADD-FACT NIL (QUOTE *1*BTM-OBJECTS)
                                  BTM-FN-SYMB))
       [AND BTM-FN-SYMB (ADD-FACT BTM-FN-SYMB (QUOTE TYPE-PRESCRIPTION-LST)
                                  (CONS SHELL-NAME (CONS (LOGBIT CURRENT-TYPE-NO)
                                                         NIL]
       (ADD-FACT NIL (QUOTE RECOGNIZER-ALIST)
              (CONS RECOGNIZER (LOGBIT CURRENT-TYPE-NO)))
       [ADD-FACT RECOGNIZER (QUOTE TYPE-PRESCRIPTION-LST)
              (CONS SHELL-NAME (CONS TYPE-SET-BOOLEAN (QUOTE (NIL]
       [for PAIR in DESTRUCTOR-ALIST do (ADD-FACT (CAR PAIR)
                                                  (QUOTE TYPE-PRESCRIPTION-LST)
                                                  (CONS SHELL-NAME (CONS (fetch (TYPE-RESTRICTION TYPE-SET)
                                                                            of (CDR PAIR))
                                                                         (QUOTE (NIL]
       NIL])
```

(ALL-ARGLISTS

```
[LAMBDA (FNNAME TERM)                                              (* kbr: "19-Oct-85 16:31")

        (* Returns the set of arglists of all subterms of TERM with function symbol FNNAME.
        *)

   (COND
      ((VARIABLEP TERM)
       NIL)
      ((FQUOTEP TERM)
       (COND
          ((OR (ASSOC FNNAME SHELL-ALIST)
               (MEMB FNNAME *1*BTM-OBJECTS))
           (ERROR1 (PQUOTE (PROGN ALL-ARGLISTS DOES NOT KNOW HOW TO GO INTO QUOTED CONSTANTS FOR BOTTOM
                                  OBJECTS
                                    AND SHELL CONSTRUCTORS %.))
                   NIL
```

```
                (QUOTE HARD)))
           (T NIL)))
      [(EQ (FFN-SYMB TERM)
           FNNAME)
       (ADD-TO-SET (FARGS TERM)
              (for ARG in (FARGS TERM) bind LOOP-ANS do (SETQ LOOP-ANS (UNION-EQUAL (ALL-ARGLISTS FNNAME ARG)
                                                                                    LOOP-ANS))
                   finally (RETURN LOOP-ANS]
      (T (for ARG in (FARGS TERM) bind LOOP-ANS do (SETQ LOOP-ANS (UNION-EQUAL (ALL-ARGLISTS FNNAME ARG)
                                                                                LOOP-ANS))
            finally (RETURN LOOP-ANS])
```

(ALL-FNNAMES

```
[LAMBDA (TERM)                                                  (* kbr: "19-Oct-85 16:31")
  (LET (ANS)
       (ALL-FNNAMES1 TERM)
       ANS])
```

(ALL-FNNAMES-LST

```
[LAMBDA (LST)                                                   (* kbr: "19-Oct-85 16:31")
  (LET (ANS)
       (for X in LST do (ALL-FNNAMES1 X))
       ANS])
```

(ALL-FNNAMES1

```
[LAMBDA (TERM)                                                  (* kbr: "19-Oct-85 16:31")
  (COND
     ((VARIABLEP TERM)
      NIL)
     ((FQUOTEP TERM)
      (ALL-FNNAMES1-EVG (CADR TERM)))
     (T [COND
           ((AND (NEQ (QUOTE IF)
                      (FFN-SYMB TERM))
                 (NEQ (QUOTE EQUAL)
                      (FFN-SYMB TERM)))
            (SETQ ANS (ADD-TO-SET (FFN-SYMB TERM)
                                  ANS]
        (for ARG in (FARGS TERM) do (ALL-FNNAMES1 ARG])
```

(ALL-FNNAMES1-EVG

```
[LAMBDA (EVG)                                                   (* kbr: "19-Oct-85 16:31")
  (COND
     [(NLISTP EVG)
      (SETQ ANS (UNIONQ ANS (COND
                               ((EQ EVG *1*T)
                                (QUOTE (TRUE)))
                               ((EQ EVG *1*F)
                                (QUOTE (FALSE)))
                               [(FIXP EVG)
                                (COND
                                   ((LESSP EVG 0)
                                    (QUOTE (MINUS ADD1 ZERO)))
                                   ((EQUAL EVG 0)
                                    (QUOTE (ZERO)))
                                   (T (QUOTE (ADD1 ZERO]
                               (T (QUOTE (PACK CONS ADD1 ZERO]
     ((EQ (CAR EVG)
          *1*SHELL-QUOTE-MARK)
      (SETQ ANS (ADD-TO-SET (CADR EVG)
                            ANS))
      (for X in (CDDR EVG) do (ALL-FNNAMES1-EVG X)))
     (T (SETQ ANS (ADD-TO-SET (QUOTE CONS)
                              ANS))
        (ALL-FNNAMES1-EVG (CAR EVG))
        (ALL-FNNAMES1-EVG (CDR EVG])
```

(ALL-INSERTIONS

```
[LAMBDA (X FINAL-SEG INIT-SEG)                                  (* kbr: "19-Oct-85 16:31")

          (* Inserts X into FINAL-SEG in all possible ways IDIFFERENCE assuming INIT-SEG is NIL at the top most call.
          *)

  (COND
     [(NULL FINAL-SEG)
      (LIST (APPEND INIT-SEG (LIST X]
     (T (CONS (APPEND INIT-SEG (LIST X)
                      FINAL-SEG)
              (ALL-INSERTIONS X (CDR FINAL-SEG)
                     (NCONC1 INIT-SEG (CAR FINAL-SEG])
```

(ALL-PATHS

```
[LAMBDA (FORM)                                          (* kbr: "19-Oct-85 16:31")

        (* This function is used only by OPTIMIZE-COMMON-SUBTERMS.
        It is assumed that FORM is as described in the documentation of OPTIMIZE-COMMON-SUBTERMS.
        In particular, *2*IF and QUOTE are the only symbols used as function symbols in FORM that are not spread LAMBDAs.
        A real-path through FORM is defined to be a list of all of the subterms of FORM that are MEMBers of
        COMMONSUBTERMS and that are evaluated in the evaluation of FORM under some assignment of values to the variables
        in FORM. The terms are listed in reverse order of evaluation completion, with FORM coming first.
        ALL-PATHS returns a list L of pairs. Each pair consists of a flag doted with a list of subterms of FORM that are members of
        COMMONSUBTERMS. For each real-path P through FORM, there exists a member
        (FLG) of L such that L1 is PATH-EQ to P and (a) if FLG is NIL, then any evaluation of FORM whose real-path is P returns
        NIL and (b) if FLG is T, then any such evaluation returns something other than NIL.
        If FLG is ?, nothing is asserted. Not every member of L need correspond to real-path.
        For example, even if FOO always returns T, (ALL-PATHS (*2*IF
        (FOO X) (G X) (H X))) will return a list of length two. In the documentation of OPTIMIZE-COMMON-SUBTERMS, we define
        the concepts FIRST, SECOND, and ISOLATED on a path. From the foregoing specification of the output of ALL-PATHS, we
        may conclude that if a MEMBer of COMMONSUBTERMS is SECOND on every path in
        (ALL-PATHS FORM) on which it occurs, then it is SECOND on any real-path through FORM on which it occurs.
        Furthermore, we may conclude that if a MEMBer of COMMON-SUBTERMS is ever FIRST on any real-path through FORM,
        then it is FIRST on some path in (ALL-PATHS FORM)%. These two observations are the key to the soundness of
        OPTIMIZE-COMMON-SUBTERMS. (A) If a term is ever FIRST on any path of ALL-PATHS, then the appropriate *2*variable
        is set when it is executed (if it has not already been set.) (B) If a term is SECOND on each path of
        (ALL-PATHS FORM), then we assume that the appropriate *2*variable has been set and we use it.
        If a term is FIRST on each path of ALL-PATHS on which it occurs, then it is first on each real-path.
        Thus there is no loss of efficiency in simply SETting the appropriate *2*variable.
        *)

    (LET (TEMP)
         (COND
            ([OR (EQ FORM NIL)
                 (EQUAL FORM (QUOTE (QUOTE NIL]
             (LIST (CONS NIL NIL)))
            ((OR (EQ FORM T)
                 (AND (LISTP FORM)
                      (EQ (CAR FORM)
                          (QUOTE QUOTE)))
                 (FIXP FORM))
             (LIST (CONS T NIL)))
            ((NLISTP FORM)
             (LIST (CONS (QUOTE ?)
                         NIL)))
            ((NEQ (FFN-SYMB FORM)
                  (QUOTE *2*IF))
             (for PICK in [ALL-PICKS (for ARG in (REVERSE (FARGS FORM)) collect (CDR-ALL (ALL-PATHS ARG]
                bind LOOP-ANS do (SETQ LOOP-ANS (PATH-ADD-TO-SET [CONS (QUOTE ?)
                                                                       (COND
                                                                          ((MEMB FORM COMMONSUBTERMS)
                                                                           (CONS FORM
                                                                                 (APPLY (FUNCTION APPEND)
                                                                                        PICK)))
                                                                          (T (APPLY (FUNCTION APPEND)
                                                                                    PICK]
                                                                 LOOP-ANS))
                finally (RETURN LOOP-ANS)))
            (T (PATH-UNION (for PICK in [ALL-PICKS (LIST (ALL-PATHS (CADDR FORM))
                                                         (for X in (SETQ TEMP (ALL-PATHS (CADR FORM)))
                                                            unless (EQ (CAR X)
                                                                       NIL)
                                                            collect (CDR X]
                           bind LOOP-ANS do (SETQ LOOP-ANS
                                                  (PATH-ADD-TO-SET [CONS (CAR (CAR PICK))
                                                                         (COND
                                                                            [(MEMB FORM COMMONSUBTERMS)
                                                                             (CONS FORM (APPEND
                                                                                          (CDR (CAR PICK))
                                                                                          (CADR PICK]
                                                                            (T (APPEND (CDR (CAR PICK))
                                                                                       (CADR PICK]
                                                                   LOOP-ANS))
                           finally (RETURN LOOP-ANS))
                    (for PICK in [ALL-PICKS (LIST (ALL-PATHS (CADDDR FORM))
                                                  (for X in TEMP unless (EQ T (CAR X)) collect (CDR X]
                       bind LOOP-ANS do (SETQ LOOP-ANS (PATH-ADD-TO-SET
                                                         [CONS (CAR (CAR PICK))
                                                               (COND
                                                                  [(MEMB FORM COMMONSUBTERMS)
                                                                   (CONS FORM (APPEND (CDR (CAR PICK))
                                                                                      (CADR PICK]
                                                                  (T (APPEND (CDR (CAR PICK))
                                                                             (CADR PICK]
                                                         LOOP-ANS))
                       finally (RETURN LOOP-ANS])
```

(ALL-PERMUTATIONS

```
[LAMBDA (L)                                             (* kbr: "19-Oct-85 19:58")
```

```
                                                                    (* Returns the list of all permutations of list L.
                                                                    *)
    (COND
       ((NULL L)
        (LIST NIL))
       (T (for PERM in (ALL-PERMUTATIONS (CDR L)) join (ALL-INSERTIONS (CAR L)
                                                                PERM NIL])
```

(ALL-PICKS

```
  [LAMBDA (POCKET-LIST)                                             (* kbr: "19-Oct-85 20:03")

          (* POCKET-LIST is a list of pockets and this fn returns all of the possible ways you can pick one thing from each pocket.
          *)

    (COND
       ((NULL POCKET-LIST)
        (LIST NIL))
       (T (for PICK in (ALL-PICKS (CDR POCKET-LIST)) join (for CHOICE in (CAR POCKET-LIST)
                                                             collect (CONS CHOICE PICK])
```

(ALL-SUBSEQUENCES

```
  [LAMBDA (L MAX)                                                   (* kbr: "19-Oct-85 16:31")

          (* Returns all subsets of L which have length less than or equal to MAX, preserving the order of the elements in L.
          *)

    (LET (TEMP)
         (COND
            ((NULL L)
             (LIST NIL))
            (T (SETQ TEMP (ALL-SUBSEQUENCES (CDR L)
                                    MAX))
               (APPEND TEMP (for X in TEMP unless (EQLENGTH X MAX) collect (CONS (CAR L)
                                                                                 X])
```

(ALL-VARS

```
  [LAMBDA (TERM)                                                    (* kbr: " 6-Jul-86 09:29")

          (* Free variables in TERM. Collects vars in TERM in reverse print order of first occurrences.
          This ordering is exploited in LOOP-STOPPER. *)

    (LET (ANS)
         (ALL-VARS1 TERM)
         ANS])
```

(ALL-VARS-BAG

```
  [LAMBDA (TERM)                                                    (* kbr: "19-Oct-85 16:31")
    (LET (ANS)
         (ALL-VARS-BAG1 TERM)
         ANS])
```

(ALL-VARS-BAG1

```
  [LAMBDA (TERM)                                                    (* kbr: "19-Oct-85 16:31")
    (COND
       ((VARIABLEP TERM)
        (SETQ ANS (CONS TERM ANS)))
       ((FQUOTEP TERM)
        NIL)
       (T (for ARG in (FARGS TERM) do (ALL-VARS-BAG1 ARG])
```

(ALL-VARS-LST

```
  [LAMBDA (LST)                                                     (* kbr: " 6-Jul-86 09:31")
                                                                    (* Free variables occuring in a LST of terms.
                                                                    *)
    (for TERM in LST bind LOOP-ANS do (SETQ LOOP-ANS (UNIONQ (ALL-VARS TERM)
                                                             LOOP-ANS))
       finally (RETURN LOOP-ANS])
```

(ALL-VARS1

```
  [LAMBDA (TERM)                                                    (* kbr: " 6-Jul-86 09:31")

          (* Called by ALL-VARS. Add free variables in TERM to the growing answer ANS bound by ALL-VARS.
          *)

    (COND
       ((VARIABLEP TERM)
        (SETQ ANS (ADD-TO-SET TERM ANS)))
       ((FQUOTEP TERM)
        NIL)
       (T (for ARG in (FARGS TERM) do (ALL-VARS1 ARG])
```

(ALMOST-SUBSUMES

```
[LAMBDA (CL1 CL2)                                          (* kbr: "19-Oct-85 16:31")
  (COND
     ((NULL CL1)
      (SETQ ALMOST-SUBSUMES-LITERAL ALMOST-SUBSUMES-CONSTANT)
      T)
     ((MEMBER (CAR CL1)
              CL2)
      (ALMOST-SUBSUMES (CDR CL1)
              CL2))
     ((MEMB-NEGATIVE (CAR CL1)
              CL2)
      (COND
         ((SUBSETP (CDR CL1)
                   CL2)
          (SETQ ALMOST-SUBSUMES-LITERAL (CAR CL1))
          T)
         (T NIL)))
     (T NIL])
```

(ALMOST-SUBSUMES-LOOP

```
[LAMBDA (LST)                                              (* kbr: "19-Oct-85 16:31")
  (LET (HITFLG ANS DEADLST)
       (SETQ HITFLG T)
       (while HITFLG do (SETQ HITFLG NIL)
                        (SETQ ANS NIL)
                        (SETQ DEADLST NIL)
                        [for CL1 in LST do (COND
                                             [(for CL2 in LST when (AND (NEQ CL1 CL2)
                                                                        (NOT (MEMB CL2 DEADLST)))
                                                 thereis (COND
                                                           ((ALMOST-SUBSUMES CL2 CL1)
                                                            (SETQ DEADLST (CONS CL1 DEADLST))
                                                            (COND
                                                               ((EQ ALMOST-SUBSUMES-LITERAL
                                                                    ALMOST-SUBSUMES-CONSTANT)
                                                                T)
                                                               (T (SETQ HITFLG T)
                                                                  (SETQ ANS (CONS (REMOVE-NEGATIVE
                                                                                   ALMOST-SUBSUMES-LITERAL
                                                                                   CL1)
                                                                                  ANS))
                                                                  T)))
                                                           (T NIL]
                                             (T (SETQ ANS (CONS CL1 ANS]
                        (SETQ LST ANS))
       ANS])
```

(ALMOST-VALUEP

```
[LAMBDA (TERM)                                             (* kbr: "19-Oct-85 16:31")
  (AND (NVARIABLEP TERM)
       (ALMOST-VALUEP1 TERM])
```

(ALMOST-VALUEP1

```
[LAMBDA (TERM)                                             (* kbr: "19-Oct-85 16:31")
  (COND
     ((VARIABLEP TERM)
      T)
     ((FQUOTEP TERM)
      T)
     ((SHELLP TERM)
      (for ARG in (FARGS TERM) always (ALMOST-VALUEP1 ARG)))
     (T NIL])
```

(APPLY-HINTS

```
[LAMBDA (HINTS TERM)                                       (* kbr: "19-Oct-85 16:31")
  (SETQ TERM (APPLY-USE-HINT (CDR (ASSOC (QUOTE USE)
                                         HINTS))
                             (APPLY-INDUCT-HINT (CADR (ASSOC (QUOTE INDUCT)
                                                             HINTS))
                                                TERM)))
  [for X in HINT-VARIABLE-ALIST when (ASSOC (CAR X)
                                            HINTS)
     do (SET (CADR X)
             (COND
                ((CADDR X)
                 (for Y in (CDR (ASSOC (CAR X)
                                       HINTS))
                    collect (TRANSLATE Y)))
                (T (CDR (ASSOC (CAR X)
                               HINTS]
```

```
    TERM])
```

(APPLY-INDUCT-HINT

```
  [LAMBDA (HINT TERM)                                                    (* kbr: "19-Oct-85 16:31")
    (PROG (FORMALS)
          (COND
             ((NULL HINT)
              (RETURN TERM)))
          (SETQ HINT (TRANSLATE HINT))
          [SETQ FORMALS (CADR (GETPROP (FFN-SYMB HINT)
                                       (QUOTE SDEFN]
          (RETURN
           (CONJOIN (for CL
                       in [IND-FORMULA [for TA in (GETPROP (FN-SYMB HINT)
                                                           (QUOTE INDUCTION-MACHINE))
                                              collect (create TESTS-AND-ALISTS
                                                              TESTS _ (SUB-PAIR-VAR-LST FORMALS (FARGS HINT)
                                                                                        (fetch (TESTS-AND-CASES TESTS) of TA))
                                                              ALISTS _
                                                              (for ARGLIST in (fetch (TESTS-AND-CASES CASES) of TA)
                                                                 collect (for ARG in ARGLIST as ACTUAL
                                                                            in (FARGS HINT)
                                                                            collect (CONS ACTUAL (SUB-PAIR-VAR FORMALS
                                                                                                               (FARGS HINT)
                                                                                                               ARG]
                                       (LIST HINT)
                                       (LIST (LIST (TRANSLATE TERM]
                             collect (DISJOIN CL NIL))
                    NIL])
```

(APPLY-USE-HINT

```
  [LAMBDA (HINT TERM)                                                    (* kbr: "20-Oct-85 19:41")
    (COND
       ((NULL HINT)
        TERM)
       (T (DUMB-IMPLICATE-LITS (CONJOIN
                                [for PAIR in HINT bind EVENT
                                   collect (PROGN (SETQ EVENT (GETPROP (CAR PAIR)
                                                                       (QUOTE EVENT)))
                                                  (SUBLIS-VAR [for X in (CDR PAIR)
                                                                 collect (CONS (TRANSLATE (CAR X))
                                                                               (TRANSLATE (CADR X]
                                                        (TRANSLATE (SELECTQ (CAR EVENT)
                                                                            (DEFN (LIST (QUOTE EQUAL)
                                                                                        (CONS (CADR EVENT)
                                                                                              (CADDR EVENT))
                                                                                        (CADDDR EVENT)))
                                                                            (REFLECT (SETQ TEMP-TEMP
                                                                                           (GETPROP (CADR EVENT)
                                                                                                    (QUOTE SDEFN)))
                                                                                     (LIST (QUOTE EQUAL)
                                                                                           (CONS (CADR EVENT)
                                                                                                 (CADR TEMP-TEMP))
                                                                                           (CADDR TEMP-TEMP)))
                                                                            (CADDDR EVENT]
                                NIL)
                TERM])
```

(ARG1-IN-ARG2-UNIFY-SUBST

```
  [LAMBDA (ARG1 ARG2)                                                    (* kbr: "19-Oct-85 16:31")
    (COND
       ((OR (VARIABLEP ARG2)
            (FQUOTEP ARG2))
        NIL)
       ((ONE-WAY-UNIFY ARG2 ARG1)
        T)
       (T (for ARG in (FARGS ARG2) thereis (ARG1-IN-ARG2-UNIFY-SUBST ARG1 ARG])
```

(ARGN0

```
  [LAMBDA (TERM N)                                                       (* kbr: "19-Oct-85 16:31")
    (COND
       ((NEQ (CAR TERM)
             (QUOTE QUOTE))
        (BM-NTH N TERM))
       [(LITATOM (CADR TERM))
        (LIST (QUOTE QUOTE)
              (DTACK-0-ON-END (CHCON (CADR TERM]
       [(FIXP (CADR TERM))
        (COND
           [(LESSP (CADR TERM)
                   0)
            (LIST (QUOTE QUOTE)
                  (MINUS (CADR TERM]
```

```
          (T (LIST (QUOTE QUOTE)
                   (SUB1 (CADR TERM]
      [(EQ (CAR (CADR TERM))
           *1*SHELL-QUOTE-MARK)
       (LIST (QUOTE QUOTE)
             (BM-NTH N (CDR (CADR TERM]
      (T (COND
            [(IEQP N 1)
             (LIST (QUOTE QUOTE)
                   (CAR (CADR TERM]
            (T (LIST (QUOTE QUOTE)
                     (CDR (CADR TERM])
```

(ARITY

```
[LAMBDA (FNNAME)                                          (* kbr: "19-Oct-85 16:31")
  (COND
     ((SETQ TEMP-TEMP (TYPE-PRESCRIPTION FNNAME))
      (LENGTH (CDR TEMP-TEMP)))
     ((SETQ TEMP-TEMP (ASSOC FNNAME ARITY-ALIST))
      (CDR TEMP-TEMP))
     (T NIL])
```

(ASSOC-OF-APPEND

```
[LAMBDA NIL                                               (* kbr: "19-Oct-85 16:31")
  (REDO-UNDONE-EVENTS [QUOTE ((DEFN APPEND (X Y)
                                    (IF (LISTP X)
                                        (CONS (CAR X)
                                              (APPEND (CDR X)
                                                      Y))
                                        Y))
                              (PROVE-LEMMA ASSOC-OF-APPEND (REWRITE)
                                     (EQUAL (APPEND (APPEND A B)
                                                    C)
                                            (APPEND A (APPEND B C]
        T
        (QUOTE Q)
        NIL NIL NIL])
```

(ASSUME-TRUE-FALSE

```
[LAMBDA (TERM)                                            (* kbr: "19-Oct-85 16:31")
  (LET (NOT-FLG TYPE-ARG1 TYPE-ARG2 TRUE-SEG FALSE-SEG PAIR ARG1 ARG2 INTERSECTION SWAPPED-TERM SWAP-FLG
        LOCAL-MUST-BE-TRUE LOCAL-MUST-BE-FALSE)
       (COND
          ((BM-MATCH TERM (NOT TERM))
           (SETQ NOT-FLG T)))
       [COND
          [(AND (NVARIABLEP TERM)
                (NOT (FQUOTEP TERM))
                (SETQ PAIR (ASSOC (FFN-SYMB TERM)
                                  RECOGNIZER-ALIST)))
           (SETQ TYPE-ARG1 (TYPE-SET (FARGN TERM 1)))
           (COND
              ((IEQP 0 (LOGAND TYPE-ARG1 (CDR PAIR)))
               (SETQ LOCAL-MUST-BE-FALSE T))
              ((LOGSUBSETP TYPE-ARG1 (CDR PAIR))
               (SETQ LOCAL-MUST-BE-TRUE T))
              (T [SETQ TRUE-SEG (LIST (CONS (FARGN TERM 1)
                                            (CDR PAIR]
                 (SETQ FALSE-SEG (LIST (CONS (FARGN TERM 1)
                                             (LOGAND (LOGNOT (CDR PAIR))
                                                     TYPE-ARG1]
          [(BM-MATCH TERM (EQUAL ARG1 ARG2))
           (COND
              ((EQUAL ARG1 ARG2)
               (SETQ LOCAL-MUST-BE-TRUE T))
              ((AND (SETQ TEMP-TEMP (CDR (SASSOC TERM TYPE-ALIST)))
                    (IEQP TEMP-TEMP TYPE-SET-TRUE))
               (SETQ LOCAL-MUST-BE-TRUE T))
              ((AND TEMP-TEMP (IEQP TEMP-TEMP TYPE-SET-FALSE))
               (SETQ LOCAL-MUST-BE-FALSE T))
              ((AND (SETQ TEMP-TEMP (CDR (SASSOC (SETQ SWAPPED-TERM (FCONS-TERM* (QUOTE EQUAL)
                                                                                 ARG2 ARG1))
                                                 TYPE-ALIST)))
                    (EQUAL TEMP-TEMP TYPE-SET-TRUE))
               (SETQ LOCAL-MUST-BE-TRUE T))
              ((AND TEMP-TEMP (IEQP TEMP-TEMP TYPE-SET-FALSE))
               (SETQ LOCAL-MUST-BE-FALSE T))
              (T (SETQ SWAP-FLG (TERM-ORDER ARG1 ARG2))
                 (SETQ TYPE-ARG1 (TYPE-SET ARG1))
                 (SETQ TYPE-ARG2 (TYPE-SET ARG2))
                 (SETQ INTERSECTION (LOGAND TYPE-ARG1 TYPE-ARG2))
                 (COND
                    ((IEQP 0 INTERSECTION)
                     (SETQ LOCAL-MUST-BE-FALSE T))
```

```
                    ((AND (IEQP TYPE-ARG1 TYPE-ARG2)
                          (MEMBER TYPE-ARG1 SINGLETON-TYPE-SETS))
                     (SETQ LOCAL-MUST-BE-TRUE T))
                    (T [SETQ TRUE-SEG (COND
                                        (SWAP-FLG (LIST (CONS SWAPPED-TERM TYPE-SET-TRUE)))
                                        (T (LIST (CONS TERM TYPE-SET-TRUE]
                       (OR (IEQP TYPE-ARG1 INTERSECTION)
                           (NOT SWAP-FLG)
                           (SETQ TRUE-SEG (CONS (CONS ARG1 INTERSECTION)
                                                TRUE-SEG)))
                       (OR (IEQP TYPE-ARG2 INTERSECTION)
                           SWAP-FLG
                           (SETQ TRUE-SEG (CONS (CONS ARG2 INTERSECTION)
                                                TRUE-SEG)))
                       (SETQ FALSE-SEG (LIST (CONS TERM TYPE-SET-FALSE)
                                             (CONS SWAPPED-TERM TYPE-SET-FALSE)))
                       (OR (NOT (MEMBER TYPE-ARG2 SINGLETON-TYPE-SETS))
                           (SETQ FALSE-SEG (CONS (CONS ARG1 (LOGAND (LOGNOT TYPE-ARG2)
                                                                    TYPE-ARG1))
                                                 FALSE-SEG)))
                       (OR (NOT (MEMBER TYPE-ARG1 SINGLETON-TYPE-SETS))
                           (SETQ FALSE-SEG (CONS (CONS ARG2 (LOGAND (LOGNOT TYPE-ARG1)
                                                                    TYPE-ARG2))
                                                 FALSE-SEG]
           (T (SETQ TYPE-ARG1 (TYPE-SET TERM))
              (COND
                 ((IEQP TYPE-ARG1 TYPE-SET-FALSE)
                  (SETQ LOCAL-MUST-BE-FALSE T))
                 ((IEQP 0 (LOGAND TYPE-ARG1 TYPE-SET-FALSE))
                  (SETQ LOCAL-MUST-BE-TRUE T))
                 (T [SETQ TRUE-SEG (LIST (CONS TERM (LOGAND TYPE-ARG1 (LOGNOT TYPE-SET-FALSE]
                    (SETQ FALSE-SEG (LIST (CONS TERM TYPE-SET-FALSE]
        (COND
           (NOT-FLG (swap LOCAL-MUST-BE-TRUE LOCAL-MUST-BE-FALSE)
                    (swap TRUE-SEG FALSE-SEG)))
        (SETQ TRUE-TYPE-ALIST (NCONC TRUE-SEG TYPE-ALIST))
        (SETQ FALSE-TYPE-ALIST (NCONC FALSE-SEG TYPE-ALIST))
        (SETQ MUST-BE-TRUE LOCAL-MUST-BE-TRUE)
        (SETQ MUST-BE-FALSE LOCAL-MUST-BE-FALSE)
        NIL])
```

(**ATTEMPT-TO-REWRITE-RECOGNIZER**

```
  [LAMBDA (TERM)                                          (* kbr: "19-Oct-85 16:31")
    (BM-MATCH TERM (NOT TERM))
    (AND (NVARIABLEP TERM)
         (ASSOC (FN-SYMB TERM)
                RECOGNIZER-ALIST)
         (VARIABLEP (ARGN TERM 1])

)

(RPAQQ CODE-B-DCOMS
       ((* CODE-B-D *)
        (FNS BATCH-PROVEALL BOOLEAN BOOT-STRAP0 BREAK-LEMMA BTM-OBJECT BTM-OBJECT-OF-TYPE-SET BTM-OBJECTP
             BUILD-SUM CANCEL CANCEL-POSITIVE CANCEL1 CAR-CDRP CDR-ALL CHK-ACCEPTABLE-DEFN CHK-ACCEPTABLE-DCL
             CHK-ACCEPTABLE-ELIM-LEMMA CHK-ACCEPTABLE-GENERALIZE-LEMMA CHK-ACCEPTABLE-HINTS CHK-ACCEPTABLE-LEMMA
             CHK-ACCEPTABLE-META-LEMMA CHK-ACCEPTABLE-REFLECT CHK-ACCEPTABLE-REWRITE-LEMMA CHK-ACCEPTABLE-SHELL
             CHK-ACCEPTABLE-TOGGLE CHK-ARGLIST CHK-MEANING CHK-NEW-*1*NAME CHK-NEW-NAME CLAUSIFY CLAUSIFY-INPUT
             CLAUSIFY-INPUT1 CLEAN-UP-BRANCHES CNF-DNF COMMON-SWEEP COMMUTE-EQUALITIES COMPARE-STATS
             COMPLEMENTARY-MULTIPLEP COMPLEMENTARYP COMPLEXITY COMPRESS-POLY COMPRESS-POLY1 COMPUTE-VETOES
             COMSUBT1 COMSUBTERMS CONJOIN CONJOIN-CLAUSE-SETS CONJOIN2 CONS-PLUS CONS-TERM CONSJOIN
             CONTAINS-REWRITEABLE-CALLP CONVERT-CAR-CDR CONVERT-CONS CONVERT-NOT CONVERT-QUOTE
             CONVERT-TYPE-NO-TO-RECOGNIZER-TERM BM-COUNT COUNT-IFS CREATE-REWRITE-RULE DCL0 DECODE-IDATE
             DEFN-ASSUME-TRUE-FALSE DEFN-LOGIOR DEFN-SETUP DEFN-TYPE-SET DEFN-TYPE-SET2 DEFN-WRAPUP DEFN0
             DELETE1 DELETE-TAUTOLOGIES DELETE-TOGGLES DEPEND DEPENDENT-EVENTS DEPENDENTS-OF DEPENDENTS-OF1
             DESTRUCTORS DESTRUCTORS1 DETACH DETACHED-ERROR DETACHEDP DISJOIN DISJOIN-CLAUSES DISJOIN2
             DTACK-0-ON-END DUMB-CONVERT-TYPE-SET-TO-TYPE-RESTRICTION-TERM DUMB-IMPLICATE-LITS DUMB-NEGATE-LIT
             DUMB-OCCUR DUMB-OCCUR-LST DUMP DUMP-ADD-AXIOM DUMP-ADD-SHELL DUMP-BEGIN-GROUP DUMP-DCL DUMP-DEFN
             DUMP-END-GROUP DUMP-HINTS DUMP-LEMMA-TYPES DUMP-OTHER DUMP-PROVE-LEMMA DUMP-TOGGLE)))


          (* * CODE-B-D *)

(DEFINEQ

(BATCH-PROVEALL
  [LAMBDA (FILE)                                          (* kbr: "19-Oct-85 16:31")

          (* FILE should contain a sequence of forms such as (PROVEALL ...)
          (PROVEALL ...)%. Each is executed. *)

    (RESTART-BATCH (READ-FILE FILE])

(BOOLEAN
  [LAMBDA (TERM)                                          (* kbr: "19-Oct-85 16:31")
```

```
    (LOGSUBSETP (TYPE-SET TERM)
           TYPE-SET-BOOLEAN])
```

(BOOT-STRAP0

```
  [LAMBDA NIL                                                    (* kbr: "19-Oct-85 16:31")
    (ADD-SUB-FACT NIL NIL NIL NIL T)
    (ADD-SUB-FACT NIL NIL NIL NIL T)
    (MAKUNBOUND (QUOTE LIB-FILE])
```

(BREAK-LEMMA

```
  [LAMBDA (NAME WHEN)                                            (* kbr: "26-Oct-85 14:57")
    (OR WHEN (SETQ WHEN T))
    [APPLY (FUNCTION TRACE)
           (LIST (LIST (QUOTE RELIEVE-HYPS)
                       (QUOTE BREAK)
                       (QUOTE (AND (SETQ TEMP-TEMP (ASSOC (CADR LEMMA)
                                                          BROKEN-LEMMAS))
                                   (EVAL (CDR TEMP-TEMP))
                                   (PROGN (BM-PPR (LIST (LIST (QUOTE LEMMA)
                                                              (CADR LEMMA))
                                                        (LIST (QUOTE TERM)
                                                              TERM)
                                                        (LIST (QUOTE UNIFY-SUBST)
                                                              UNIFY-SUBST))
                                                  T)
                                          T]
    (SETQ BROKEN-LEMMAS (ADD-TO-SET (CONS NAME WHEN)
                                    BROKEN-LEMMAS])
```

(BTM-OBJECT

```
  [LAMBDA (CONST)                                                (* kbr: "19-Oct-85 16:31")
```

(* If the shell for which CONST is the constructor has a bottom object return the term that is that bottom object. Else, return NIL. *)

```
    (LET (TYPE-SET ANS)
         [SETQ TYPE-SET (LSH 1 (CDR (ASSOC CONST SHELL-ALIST]
         (COND
            ((for FN in *1*BTM-OBJECTS thereis (IEQP (TYPE-SET (SETQ ANS (CONS-TERM FN NIL)))
                                                     TYPE-SET))
             ANS)
            (T NIL])
```

(BTM-OBJECT-OF-TYPE-SET

```
  [LAMBDA (TYPE-SET)                                             (* kbr: "19-Oct-85 16:31")
```

(* Returns the btm object fn symb with the specified type set, or NIL if no such btm object exists. *)

```
    (COND
       ((NULL (CDR *1*BTM-OBJECTS))
        (COND
           ((IEQP TYPE-SET TYPE-SET-NUMBERS)
            (QUOTE ZERO))
           (T NIL)))
       (T (for X in *1*BTM-OBJECTS when (IEQP TYPE-SET (CAR (TYPE-PRESCRIPTION X))) do (RETURN X])
```

(BTM-OBJECTP

```
  [LAMBDA (TERM)                                                 (* kbr: "19-Oct-85 16:31")
    (COND
       ((VARIABLEP TERM)
        NIL)
       [(FQUOTEP TERM)
        (COND
           ((NLISTP (CADR TERM))
            (EQUAL 0 (CADR TERM)))
           (T (AND (EQ *1*SHELL-QUOTE-MARK (CAR (CADR TERM)))
                   (MEMB (CADR (CADR TERM))
                         *1*BTM-OBJECTS]
       (T (MEMB (FFN-SYMB TERM)
                *1*BTM-OBJECTS])
```

(BUILD-SUM

```
  [LAMBDA (WINNING-PAIR ALIST)                                   (* kbr: "20-Oct-85 15:48")
    (COND
       ((NLISTP ALIST)
        ZERO)
       ((EQUAL WINNING-PAIR (CAR ALIST))
        (BUILD-SUM WINNING-PAIR (CDR ALIST)))
       (T (CONS-PLUS [COND
                        ([EQUAL 1 (ABS (CDR (CAR ALIST]
```

```
                        (CAR (CAR ALIST)))
                       (T (FCONS-TERM* (QUOTE TIMES)
                                       [LIST (QUOTE QUOTE)
                                             (ABS (CDR (CAR ALIST]
                                       (CAR (CAR ALIST]
                  (BUILD-SUM WINNING-PAIR (CDR ALIST])
```

(CANCEL

```
[LAMBDA (EQ1 EQ2)                                          (* kbr: "19-Oct-85 16:31")
  (LET (CO1 CO2 POLY)
       (SETQ CO1 (ABS (FIRST-COEFFICIENT EQ1)))
       (SETQ CO2 (ABS (FIRST-COEFFICIENT EQ2)))
```

(* See ADD-TERMS-TO-POT-LST for an explanation of why we UNIONQ rather than UNION-EQUAL the LITERALS and LEMMAS. *)

```
       [SETQ POLY (create POLY
                          CONSTANT _ (PLUS (TIMES CO2 (fetch (POLY CONSTANT) of EQ1))
                                           (TIMES CO1 (fetch (POLY CONSTANT) of EQ2)))
                          ALIST _ (CANCEL1 CO2 (CDR (fetch (POLY ALIST) of EQ1))
                                           CO1
                                           (CDR (fetch (POLY ALIST) of EQ2)))
                          ASSUMPTIONS _ (UNION-EQUAL (fetch (POLY ASSUMPTIONS) of EQ1)
                                                     (fetch (POLY ASSUMPTIONS) of EQ2))
                          LITERALS _ (UNIONQ (fetch (POLY LITERALS) of EQ1)
                                             (fetch (POLY LITERALS) of EQ2))
                          LEMMAS _ (UNIONQ (fetch (POLY LEMMAS) of EQ1)
                                           (fetch (POLY LEMMAS) of EQ2]
        (COND
           ((IMPOSSIBLE-POLYP POLY)
            (SETQ LINEAR-ASSUMPTIONS (fetch (POLY ASSUMPTIONS) of POLY))
            (SETQ LEMMAS-USED-BY-LINEAR (UNIONQ (fetch (POLY LEMMAS) of POLY)
                                                (fetch (POLY LITERALS) of POLY)))
            (RETFROM (QUOTE ADD-EQUATIONS)
                   (QUOTE CONTRADICTION)))
           ((TRUE-POLYP POLY)
            NIL)
           (T POLY])
```

(CANCEL-POSITIVE

```
[LAMBDA (EQUATION)                                         (* kbr: "19-Oct-85 16:31")
  (COND
     ((GREATERP (FIRST-COEFFICIENT EQUATION)
                0)
      (SETQ EQUATION (create POLY
                             CONSTANT _ (fetch (POLY CONSTANT) of EQUATION)
                             ALIST _ (CDR (fetch (POLY ALIST) of EQUATION))
                             ASSUMPTIONS _ (fetch (POLY ASSUMPTIONS) of EQUATION)
                             LITERALS _ (fetch (POLY LITERALS) of EQUATION)
                             LEMMAS _ (fetch (POLY LEMMAS) of EQUATION)))
      (COND
         ((IMPOSSIBLE-POLYP EQUATION)
          (SETQ LINEAR-ASSUMPTIONS (fetch (POLY ASSUMPTIONS) of EQUATION))
          (SETQ LEMMAS-USED-BY-LINEAR (UNIONQ (fetch (POLY LEMMAS) of EQUATION)
                                              (fetch (POLY LITERALS) of EQUATION)))
          (RETFROM (QUOTE ADD-EQUATIONS)
                 (QUOTE CONTRADICTION)))
         ((TRUE-POLYP EQUATION)
          NIL)
         (T EQUATION)))
     (T NIL])
```

(CANCEL1

```
[LAMBDA (CO1 AL1 CO2 AL2)                                  (* kbr: "20-Oct-85 15:50")
  (LET (TEMP)
       (COND
          [(NULL AL1)
           (for PAIR in AL2 collect (CONS (CAR PAIR)
                                          (TIMES (CDR PAIR)
                                                 CO2]
          [(NULL AL2)
           (for PAIR in AL1 collect (CONS (CAR PAIR)
                                          (TIMES (CDR PAIR)
                                                 CO1]
          ([NOT (TERM-ORDER (CAAR AL1)
                            (CAR (CAR AL2]
           (CONS (CONS (CAR (CAR AL1))
                       (TIMES (CDR (CAR AL1))
                              CO1))
                 (CANCEL1 CO1 (CDR AL1)
                          CO2 AL2)))
          [(EQUAL (CAR (CAR AL1))
                  (CAR (CAR AL2)))
           [SETQ TEMP (PLUS (TIMES CO1 (CDR (CAR AL1)))
```

```
                              (TIMES CO2 (CDR (CAR AL2]
            (COND
               ((EQUAL TEMP 0)
                (CANCEL1 CO1 (CDR AL1)
                        CO2
                        (CDR AL2)))
               (T (CONS (CONS (CAR (CAR AL1))
                              TEMP)
                        (CANCEL1 CO1 (CDR AL1)
                                CO2
                                (CDR AL2]
            (T (CONS (CONS (CAR (CAR AL2))
                           (TIMES (CDR (CAR AL2))
                                  CO2))
                     (CANCEL1 CO1 AL1 CO2 (CDR AL2])
```

(CAR-CDRP

```
[LAMBDA (X)                                                    (* kbr: "19-Oct-85 16:31")
  (LET ((FLAT (NCHARS X)))
       (AND (EQ (NTHCHAR X 1)
                (QUOTE C))
            (EQ (NTHCHAR X FLAT)
                (QUOTE R))
            (IGREATERP FLAT 2)
            [for I from 2 by 1 to (SUB1 FLAT) always (MEMB (NTHCHAR X I)
                                                           (QUOTE (A D]
            (CDR (DREVERSE (CDR (UNPACK X])
```

(CDR-ALL

```
[LAMBDA (X)                                                    (* kbr: "19-Oct-85 16:31")
  (for X1 in X collect (CDR X1])
```

(CHK-ACCEPTABLE-DEFN

```
[LAMBDA (NAME ARGS BODY RELATION-MEASURE-LST)                  (* kbr: "26-Oct-85 15:30")
  (LET ((ARITY-ALIST (CONS (CONS NAME (LENGTH-TO-ATOM ARGS))
                           ARITY-ALIST)))
       (CHK-NEW-NAME NAME NIL)
       (CHK-NEW-*1*NAME NAME)
       (CHK-ARGLIST NAME ARGS)
       [COND
          ((IGREATERP (LENGTH ARGS)
                  32)
           (ERROR1 (PQUOTE (PROGN TOO MANY ARGS ! BECAUSE OF OUR USE OF 32-BIT WORDS TO ENCODE SETS OF
                                  RECURSION CONTROLLERS WE CANNOT ACCEPT FUNCTIONS , SUCH AS (!PPR NAME NIL)
                                  , WITH MORE THAN 32 ARGUMENTS %.))
                   (BINDINGS (QUOTE NAME)
                             NAME)
                   (QUOTE SOFT]
       (SETQ BODY (TRANSLATE BODY))
       [COND
          ((NOT IN-BOOT-STRAP-FLG)
           (CHK-MEANING NAME (ALL-FNNAMES BODY]
       (FREE-VAR-CHK NAME ARGS BODY)
       [for X in RELATION-MEASURE-LST
          do (COND
                ((NOT (AND (LISTP X)
                           (MEMB (CAR X)
                                 WELL-ORDERING-RELATIONS)
                           (LISTP (CDR X))
                           (NULL (CDDR X))
                           (SUBSETP (ALL-VARS (TRANSLATE (CADR X)))
                                    ARGS)))
                 (ERROR1 (PQUOTE (PROGN EACH MEMBER OF THE FOURTH ARGUMENT TO DEFN MUST BE OF THE FORM
                                        (!PPR (QUOTE (REL TERM))
                                              NIL)
                                        , WHERE REL IS THE NAME OF A WELL-FOUNDED RELATION
                                          AND TERM IS A TERM ALL OF WHOSE VARIABLES ARE AMONG THE FORMALS OF
                                        THE FUNCTION BEING DEFINED %.))
                         NIL
                         (QUOTE SOFT]
       NIL])
```

(CHK-ACCEPTABLE-DCL

```
[LAMBDA (NAME ARGS)                                            (* kbr: "19-Oct-85 16:31")
  (CHK-ARGLIST NAME ARGS)
  (CHK-NEW-NAME NAME NIL)
  (COND
     ((IGREATERP (LENGTH ARGS)
             32)
      (ERROR1 (PQUOTE (PROGN TOO MANY ARGS ! BECAUSE OF OUR USE OF 32-BIT WORDS TO ENCODE SETS OF RECURSION
                             CONTROLLERS WE CANNOT ACCEPT FUNCTIONS , SUCH AS (!PPR NAME NIL)
                             , WITH MORE THAN 32 ARGUMENTS %.))
              (BINDINGS (QUOTE NAME)
```

```
                    NAME)
               (QUOTE SOFT])
```

(CHK-ACCEPTABLE-ELIM-LEMMA

```
 [LAMBDA (NAME TYPE TERM)                                    (* kbr: "20-Oct-85 15:51")
   TYPE
   (LET (LST ALLVARS LHS RHS DESTS)
        (SETQ LST (UNPRETTYIFY TERM))
        [COND
           ([NOT (AND LST (NULL (CDR LST))
                      (BM-MATCH (CDR (CAR LST))
                             (EQUAL LHS RHS))
                      (VARIABLEP RHS)
                      (NVARIABLEP LHS)
                      (for ARG in (SARGS LHS) thereis (NVARIABLEP ARG]
            (ERROR1 (PQUOTE (PROGN (!PPR NAME NIL)
                                   IS AN UNACCEPTABLE ELIM LEMMA BECAUSE ITS CONCLUSION IS NOT AN EQUALITY OF
                                   THE FORM IQUOTIENT (EQUAL TERM VAR/)
                                   WHERE TERM CONTAINS SOME NON-VARIABLE ARGUMENTS
                                     AND VAR IS A VARIABLE %.))
                    (BINDINGS (QUOTE NAME)
                           NAME)
                    (QUOTE SOFT]
        (SETQ ALLVARS (ALL-VARS TERM))
        [COND
           ([NOT (SETQ DESTS (DESTRUCTORS (LIST LHS]
            (ERROR1 (PQUOTE (PROGN (!PPR NAME NIL)
                                   IS AN UNACCEPTABLE ELIM LEMMA BECAUSE THE LEFT HAND SIDE OF THE CONCLUSION
                                   DOES NOT CONTAIN ANY TERMS OF THE FORM IQUOTIENT (FN VAR1 VAR2 ... VARN/)
                                   WHERE FN IS A RECURSIVE FUNCTION AND THE VARI ARE ALL DISTINCT VARIABLES %.
                                   ))
                    (BINDINGS (QUOTE NAME)
                           NAME)
                    (QUOTE SOFT)))
           ([NOT (NO-DUPLICATESP (for X in DESTS collect (FN-SYMB X]
            (ERROR1 (PQUOTE (PROGN (!PPR NAME NIL)
                                   IS AN UNACCEPTABLE ELIM LEMMA BECAUSE THE LEFT HAND SIDE OF THE CONCLUSION
                                   CONTAINS TWO OR MORE DESTRUCTOR TERMS WITH THE SAME FUNCTION SYMBOL %.))
                   NIL
                    (QUOTE SOFT)))
           ([NOT (for X in DESTS always (SUBSETP ALLVARS (SARGS X]
            (ERROR1 (PQUOTE (PROGN (!PPR NAME NIL)
                                   IS NOT AN ACCEPTABLE ELIM LEMMA BECAUSE SOME OF THE DESTRUCTOR NESTS DO NOT
                                   MENTION ALL OF THE VARIABLES IN THE LEMMA %.))
                    (BINDINGS (QUOTE NAME)
                           NAME)
                    (QUOTE SOFT)))
           ((OCCUR RHS (SUB-PAIR-EXPR DESTS (for X in DESTS collect (TRUE X))
                             LHS))
            (ERROR1 (PQUOTE (PROGN (!PPR NAME NIL)
                                   IS AN UNACCEPTABLE ELIM LEMMA BECAUSE THE RIGHT-HAND SIDE OF THE CONCLUSION
                                   , (!PPR RHS NIL)
                                   , OCCURS IN THE LEFT-HAND SIDE IN PLACES OTHER THAN THE DESTRUCTOR
                                   (PLURAL? DESTS TERMS TERM)
                                   (!PPR-LIST DESTS)
                                   %.))
                    (BINDINGS (QUOTE NAME)
                           NAME
                           (QUOTE RHS)
                           RHS
                           (QUOTE DESTS)
                           DESTS)
                    (QUOTE SOFT)))
           (T (for X in DESTS when (GETPROP (CAR X)
                                            (QUOTE ELIMINATE-DESTRUCTORS-DESTS))
                 do (ERROR1 (PQUOTE (PROGN WE DO NOT KNOW HOW TO HANDLE MULTIPLE ELIMINATION LEMMAS FOR THE
                                           SAME FUNCTION SYMBOL, E.G., (!PPR (CAR X)
                                                                             NIL)
                                           %.))
                            (BINDINGS (QUOTE X)
                                   X)
                            (QUOTE SOFT]
        NIL])
```

(CHK-ACCEPTABLE-GENERALIZE-LEMMA

```
 [LAMBDA (NAME TYPE TERM)                                    (* kbr: "19-Oct-85 16:31")
   NAME TYPE TERM T])
```

(CHK-ACCEPTABLE-HINTS

```
 [LAMBDA (HINTS)                                             (* kbr: "26-Oct-85 17:49")
   (LET
    (EVENT)
    (for X in HINTS
       do (COND
```

```
((NLISTP X)
 (ERROR1 (PQUOTE (PROGN EACH ELEMENT OF THE HINTS ARGUMENT TO PROVE-LEMMA MUST BE A PAIR BUT
                        (!PPR X NIL)
                        IS NOT %.))
         (BINDINGS (QUOTE X)
                   X)
         (QUOTE SOFT)))
(T (SELECTQ (CAR X)
       (USE [for PAIR in (CDR X)
              do (OR [AND (LISTP PAIR)
                          (LITATOM (CAR PAIR))
                          (SETQ EVENT (GETPROP (CAR PAIR)
                                               (QUOTE EVENT)))
                          (MEMB (CAR EVENT)
                                (QUOTE (ADD-AXIOM PROVE-LEMMA DEFN REFLECT)))
                          (NULL (CDR (LAST PAIR)))
                          (for X in (CDR PAIR) always (AND (VARIABLEP (TRANSLATE (CAR X)))
                                                           (PROGN (TRANSLATE (CADR X))
                                                                  T]
                     (ERROR1 (PQUOTE (PROGN THE USE HINT MUST HAVE THE FORM (!PPR H NIL)
                                            WHERE EACH EVENTI IS THE NAME OF AN ADD-AXIOM ,
                                            PROVE-LEMMA , DEFN ,
                                              OR REFLECT EVENT , EACH VI IS A VARIABLE NAME ,
                                              AND EACH TI IS A TERM %. THE ENTRY
                                                  (!PPR PAIR NIL)
                                                  IS THUS UNACCEPTABLE %.))
                             (BINDINGS (QUOTE H)
                                       [QUOTE (USE (EVENT1 (V1 T1)
                                                           ...
                                                           (VN TN))
                                                   ...
                                                   (EVENTK (VK TK)
                                                           ...
                                                           (VM TM]
                                       (QUOTE PAIR)
                                       PAIR)
                                       (QUOTE SOFT])
       (EXPAND [for X in (CDR X) bind Y
                  do (SETQ Y (TRANSLATE X))
                     (OR (AND (NVARIABLEP Y)
                              (NOT (FQUOTEP Y))
                              (GETPROP (FFN-SYMB Y)
                                       (QUOTE SDEFN)))
                         (ERROR1 (PQUOTE (PROGN EVERY ELEMENT OF AN EXPAND HINT MUST BE AN
                                                APPLICATION OF A DEFINED FUNCTION TO SOME
                                                ARGUMENTS
                                                  AND (!PPR Y NIL)
                                                      IS NOT %.))
                                 (BINDINGS (QUOTE Y)
                                           Y)
                                 (QUOTE SOFT])
       (DISABLE [for X in (CDR X)
                  do (OR (LITATOM X)
                         (ERROR1 (PQUOTE (PROGN EVERY ELEMENT OF A DISABLE HINT MUST BE A
                                                LITERAL NLISTP AND (!PPR X NIL)
                                                                   IS NOT %.))
                                 (BINDINGS (QUOTE X)
                                           X)
                                 (QUOTE SOFT])
       (INDUCT (OR (NULL (CADR X))
                   (AND (SETQ HINT (TRANSLATE (CADR X)))
                        (NVARIABLEP HINT)
                        (NOT (FQUOTEP HINT))
                        (GETPROP (FFN-SYMB HINT)
                                 (QUOTE INDUCTION-MACHINE))
                        (GETPROP (FFN-SYMB HINT)
                                 (QUOTE SDEFN))
                        (for X in (FARGS HINT) always (VARIABLEP X))
                        (NO-DUPLICATESP (FARGS HINT)))
                   (ERROR1 (PQUOTE (PROGN THE INDUCT HINT MUST HAVE EITHER THE FORM
                                          (!PPR G NIL) OR THE FORM (!PPR H NIL)
                                          WHERE FN IS A RECURSIVELY DEFINED FUNCTION
                                            AND THE VI ARE DISTINCT VARIABLES %. THUS,
                                                (!PPR X NIL)
                                                IS AN INAPPROPRIATE INDUCT HINT %.))
                           (BINDINGS (QUOTE G)
                                     (QUOTE (INDUCT NIL))
                                     (QUOTE H)
                                     (QUOTE (INDUCT (FN V1 ... VN)))
                                     (QUOTE X)
                                     X)
                           (QUOTE SOFT))))
       (COND
          [(ASSOC (CAR X)
                  HINT-VARIABLE-ALIST)
           (COND
              ((CADDR (ASSOC (CAR X)
```

```
                                    HINT-VARIABLE-ALIST))
                              (for Y in (CDR X) do (TRANSLATE Y]
                         (T (ERROR1 (PQUOTE (PROGN EACH ENTRY IN THE HINTS ARGUMENT OF PROVE-LEMMA MUST BE A
                                                   LIST BEGINNING WITH ONE OF THE ATOMS USE , EXPAND , DISABLE
                                                   , INDUCT , OR TIME. THE PROPOSED HINT (!PPR X NIL)
                                                                IS THUS INAPPROPRIATE %.))
                                    (BINDINGS (QUOTE X)
                                              X)
                                    (QUOTE SOFT])
```

(CHK-ACCEPTABLE-LEMMA

```
[LAMBDA (NAME TYPES TERM)                                          (* kbr: "26-Nov-85 15:39")
  (CHK-NEW-NAME NAME NIL)
  (SETQ TERM (TRANSLATE TERM))
  [COND
     (IN-ADD-AXIOM-FLG (CHK-MEANING NAME (ALL-FNNAMES TERM]
  (for TYPE in TYPES do (COND
                           ((MEMB (COND
                                     ((LISTP TYPE)
                                      (CAR TYPE))
                                     (T TYPE))
                                  LEMMA-TYPES)
                            (APPLY* (PACK (LIST "CHK-ACCEPTABLE-" (COND
                                                                     ((LISTP TYPE)
                                                                      (CAR TYPE))
                                                                     (T TYPE))
                                                "-LEMMA"))
                                    NAME TYPE TERM))
                           (T (ERROR1 (PQUOTE (PROGN (!PPR TYPE NIL)
                                                     IS NOT AMONG THE LEGAL TYPES, VIZ. , (!LIST LEMMA-TYPES)
                                                     %.))
                                      (BINDINGS (QUOTE TYPE)
                                                TYPE
                                                (QUOTE LEMMA-TYPES)
                                                LEMMA-TYPES)
                                      (QUOTE SOFT])
```

(CHK-ACCEPTABLE-META-LEMMA

```
[LAMBDA (NAME TYPE TERM)                                           (* kbr: "19-Oct-85 16:31")
  (LET (FN1 V1 A1 V2 A2 FN2 V3 V4)
       [COND
          ((AND (NOT IN-ADD-AXIOM-FLG)
                NONCONSTRUCTIVE-AXIOM-NAMES)
           (ERROR1 (PQUOTE (PROGN META LEMMAS MUST BE PROVED IN A CONSTRUCTIVE HISTORY %. THE CURRENT HISTORY
                                  CONTAINS THE NONCONSTRUCTIVE (PLURAL? LST AXIOMS AXIOM)
                                  (!LIST LST)
                                  %. IF THIS METALEMMA IS PROVED USING UNSOUND AXIOMS YOU MAY GETPROP WIPED
                                  OUT BY THE APPLICATION OF THE METAFUNCTION %.))
                   (BINDINGS (QUOTE LST)
                          NONCONSTRUCTIVE-AXIOM-NAMES)
                   (QUOTE WARNING)))
          ((NOT (AND [BM-MATCH TERM (IMPLIES (FORMP V1)
                                             (AND (EQUAL (MEANING V2 A1)
                                                         (MEANING (LIST FN1 V3)
                                                                  A2))
                                                  (FORMP (LIST FN2 V4]
                     (VARIABLEP V1)
                     (VARIABLEP A1)
                     (EQ V1 V2)
                     (EQ V1 V3)
                     (EQ V1 V4)
                     (EQ A1 A2)
                     (NEQ V1 A1)
                     (GETPROP FN1 (QUOTE LISP-CODE))
                     (EQ FN1 FN2)))
           (ERROR1 (PQUOTE (PROGN META LEMMAS HAVE TO HAVE THE FORM (!PPR X NIL)
                                  WHERE V AND A ARE DISTINCT VARIABLES
                                    AND FN IS AN EXPLICIT VALUE PRESERVING FUNCTION %. (!PPR NAME NIL)
                                        DOES NOT HAVE THIS FORM %.))
                   (BINDINGS (QUOTE X)
                          [QUOTE (IMPLIES (FORMP V)
                                          (AND (EQUAL (MEANING V A)
                                                      (MEANING (FN V)
                                                               A))
                                               (FORMP (FN V]
                          (QUOTE NAME)
                          NAME)
                   (QUOTE SOFT)))
          ([NOT (AND (BM-MATCH TYPE (CONS (QUOTE META)
                                          FNS))
                     (for FN in FNS always (AND (LITATOM FN)
                                                (GETPROP FN (QUOTE TYPE-PRESCRIPTION-LST]
           (ERROR1 (PQUOTE (PROGN META LEMMAS MUST BE STORED UNDER ONE OR MORE FUNCTIONS NAMED BY THE USER
                                  IN A LEMMA TYPE OF THE FORM (!PPR X NIL)
                                  WHERE THE FNI ARE FUNCTION NAMES %. (!PPR TYPE NIL)
```

```
                                    IS NOT OF THIS FORM %.))
                    (BINDINGS (QUOTE X)
                           (QUOTE (META FN1 FN2 ... FNN))
                           (QUOTE TYPE)
                           TYPE)
                    (QUOTE SOFT]
          T])
```

(CHK-ACCEPTABLE-REFLECT

```
  [LAMBDA (NAME SATISFACTION-LEMMA-NAME RELATION-MEASURE-LST)    (* kbr: "26-Oct-85 17:50")
    (LET (FN ARGS BODY)
         (CHK-NEW-NAME NAME NIL)
         (CHK-NEW-*1*NAME NAME)
         [COND
            ((OR (NOT (BM-MATCH (FORMULA-OF SATISFACTION-LEMMA-NAME)
                                (EQUAL (CONS FN ARGS)
                                       BODY)))
                 (MEMB FN *1*BTM-OBJECTS)
                 (ASSOC FN SHELL-ALIST))
             (ERROR1 (PQUOTE (PROGN THE SECOND ARGUMENT OF REFLECT MUST BE THE NAME OF A LEMMA OF THE FORM
                                    (!PPR (QUOTE (EQUAL (FN ARG1 ... ARGN)
                                                        BODY))
                                          NIL)
                                    WHERE FN IS NOT A SHELL CONSTRUCTOR OR BOTTOM OBJECT FUNCTION SYMBOL %.
                                                                          (!PPR LEMMA NIL)
                                                                          IS NOT OF THIS FORM %.))
                     (BINDINGS (QUOTE LEMMA)
                            SATISFACTION-LEMMA-NAME)
                     (QUOTE SOFT]
         (CHK-ARGLIST NAME ARGS)
         (SETQ BODY (TRANSLATE BODY))
         [COND
            ((NOT IN-BOOT-STRAP-FLG)
             (CHK-MEANING NAME (ALL-FNNAMES BODY]
         (FREE-VAR-CHK NAME ARGS BODY)
         [for X in RELATION-MEASURE-LST
            do (COND
                  ((NOT (AND (LISTP X)
                             (MEMB (CAR X)
                                   WELL-ORDERING-RELATIONS)
                             (LISTP (CDR X))
                             (NULL (CDDR X))
                             (SUBSETP (ALL-VARS (TRANSLATE (CADR X)))
                                      ARGS)))
                   (ERROR1 (PQUOTE (PROGN EACH MEMBER OF THE THIRD ARGUMENT TO REFLECT MUST BE OF THE FORM
                                          (!PPR (QUOTE (REL TERM))
                                                NIL)
                                          , WHERE REL IS THE NAME OF A WELL-FOUNDED RELATION
                                            AND TERM IS A TERM ALL OF WHOSE VARIABLES ARE AMONG THE FORMALS OF
                                          THE FUNCTION BEING DEFINED %.))
                           NIL
                           (QUOTE SOFT]
          NIL])
```

(CHK-ACCEPTABLE-REWRITE-LEMMA

```
  [LAMBDA (NAME TYPE TERM)                                    (* kbr: "20-Oct-85 15:53")
    TYPE
    (for X in (UNPRETTYIFY TERM) bind (TOP-FNNAME-VAR REWRITE-RULE LHS ALL-VARS-HYPS ALL-VARS-CONCL MAX-TERMS LST
                                       HYPS CONCL)
       do (SETQ HYPS (CAR X))
          (SETQ CONCL (CDR X))
          (SETQ TOP-FNNAME-VAR (TOP-FNNAME CONCL))
          (COND
             ((ACCEPTABLE-TYPE-PRESCRIPTION-LEMMAP HYPS CONCL)
              T)
             ((NULL TOP-FNNAME-VAR)
              (ERROR1 (PQUOTE (PROGN (!PPR NAME NIL)
                                     IS AN UNACCEPTABLE REWRITE LEMMA BECAUSE IT REWRITES A VARIABLE %.))
                      (BINDINGS (QUOTE NAME)
                             NAME)
                      (QUOTE SOFT)))
             ((EQ TOP-FNNAME-VAR (QUOTE IF))
              (ERROR1 (PQUOTE (PROGN (!PPR NAME NIL)
                                     IS AN UNACCEPTABLE REWRITE LEMMA BECAUSE IT REWRITES AN IF-EXPRESSION %.))
                      (BINDINGS (QUOTE NAME)
                             NAME)
                      (QUOTE SOFT)))
             ((FQUOTEP CONCL)
              NIL)
             ([AND (NOT NO-BUILT-IN-ARITH-FLG)
                   (OR (BM-MATCH CONCL (NOT (LESSP & &)))
                       (BM-MATCH CONCL (LESSP & &]
              (SETQ LST (EXTERNAL-LINEARIZE CONCL T))
              [COND
                 ([OR [NOT (AND LST (NULL (CDR LST]
```

```
                  (NOT (AND (CAR LST)
                            (NULL (CDR (CAR LST]
                (ERROR1 (PQUOTE (PROGN LINEARIZE RETURNED A LIST OF MORE THAN ONE THING , EVEN THOUGH CALLED
                                       ON A LESSP NLISTP !))
                        NIL
                        (QUOTE HARD]
             (SETQ ALL-VARS-HYPS (ALL-VARS-LST HYPS))
             (SETQ ALL-VARS-CONCL (ALL-VARS CONCL))
             (SETQ MAX-TERMS (for PAIR in (fetch (POLY ALIST) of (CAR (CAR LST)))
                                when [AND (NVARIABLEP (CAR PAIR))
                                          (SUBSETP ALL-VARS-CONCL (UNIONQ (ALL-VARS (CAR PAIR))
                                                                          ALL-VARS-HYPS))
                                          (for PAIR2 in (fetch (POLY ALIST) of (CAR (CAR LST)))
                                             when (NEQ PAIR2 PAIR)
                                             never (AND (LESSP (FORM-COUNT (CAR PAIR))
                                                               (FORM-COUNT (CAR PAIR2)))
                                                        (SUBBAGP (ALL-VARS-BAG (CAR PAIR))
                                                                 (ALL-VARS-BAG (CAR PAIR2]
                                collect (CAR PAIR)))
             [COND
                ((NULL MAX-TERMS)
                 (ERROR1 (PQUOTE (PROGN (!PPR NAME NIL)
                                        IS AN UNACCEPTABLE REWRITE LEMMA BECAUSE THE NLISTP OF ITS CONCLUSION
                                        IS A LESSP AND IT CANNOT BE HANDLED BY OUR LINEAR ARITHMETIC PACKAGE.
                                        TO BE ACCEPTABLE, AT LEAST ONE NONVARIABLE ADDEND OF THE CONCLUSION
                                        MUST SATISFY TWO PROPERTIES. FIRST, IT MUST CONTAIN ALL THE VARIABLES
                                        OF THE LEMMA THAT ARE NOT IN THE HYPOTHESES. SECOND, IT MUST NOT BE
                                        THE CASE THAT UNDER EVERY SUBSTITUTION, THE TERM IS SMALLER THAN
                                        ANOTHER ADDEND OF THE CONCLUSION. %.))
                         (BINDINGS (QUOTE NAME)
                                   NAME)
                         (QUOTE SOFT]
             (for X in MAX-TERMS when (NON-RECURSIVE-DEFNP (FFN-SYMB X))
                do (ERROR1 (PQUOTE (PROGN NOTE THAT THE LINEAR LEMMA (!PPR NAME NIL)
                                          IS BEING STORED UNDER THE TERM (!PPR X NIL)
                                          , WHICH IS UNUSUAL BECAUSE (!PPR FN NIL)
                                          IS A NONRECURSIVE FUNCTION SYMBOL %.))
                           (BINDINGS (QUOTE NAME)
                                     NAME
                                     (QUOTE X)
                                     X
                                     (QUOTE FN)
                                     (FFN-SYMB X))
                           (QUOTE WARNING)))
             (for X in MAX-TERMS when (NOT (SUBSETP ALL-VARS-HYPS (ALL-VARS X)))
                do (ERROR1 (PQUOTE (PROGN WHEN THE LINEAR LEMMA (!PPR NAME NIL)
                                          IS STORED UNDER (!PPR X NIL)
                                          IT CONTAINS THE FREE (PLURAL? VARS VARIABLES VARIABLE)
                                          (!LIST VARS)
                                          WHICH WILL BE CHOSEN BY INSTANTIATING THE (PLURAL? LST HYPOTHESES
                                                                                            HYPOTHESIS)
                                          (!PPR-LIST LST)
                                          %.))
                           (BINDINGS (QUOTE NAME)
                                     NAME
                                     (QUOTE X)
                                     X
                                     (QUOTE VARS)
                                     (SET-DIFF ALL-VARS-HYPS (ALL-VARS X))
                                     (QUOTE LST)
                                     (for HYP in HYPS bind (VARS _ (SET-DIFF ALL-VARS-HYPS (ALL-VARS X)))
                                        when (INTERSECTP VARS (ALL-VARS HYP))
                                        collect (PROGN (SETQ VARS (SET-DIFF VARS (ALL-VARS HYP)))
                                                       HYP)))
                           (QUOTE WARNING)))
              T)
             (T (SETQ REWRITE-RULE (CREATE-REWRITE-RULE NAME HYPS CONCL NIL))
                (SETQ ALL-VARS-HYPS (ALL-VARS-LST HYPS))
                [SETQ ALL-VARS-CONCL (ALL-VARS (COND
                                                  ((BM-MATCH CONCL (EQUAL LHS &))
                                                   LHS)
                                                  (T CONCL]
                [COND
                   ((NON-RECURSIVE-DEFNP (TOP-FNNAME CONCL))
                    (ERROR1 (PQUOTE (PROGN NOTE THAT THE REWRITE RULE (!PPR NAME NIL)
                                           WILL BE STORED SO AS TO APPLY ONLY TO TERMS WITH THE NONRECURSIVE
                                           FUNCTION SYMBOL (!PPR FN NIL)
                                           %.))
                            (BINDINGS (QUOTE NAME)
                                      NAME
                                      (QUOTE FN)
                                      (TOP-FNNAME CONCL))
                            (QUOTE WARNING]
                [COND
                   ((NOT (SUBSETP ALL-VARS-HYPS ALL-VARS-CONCL))
                    (ERROR1 (PQUOTE (PROGN NOTE THAT (!PPR NAME NIL)
                                           CONTAINS THE FREE (PLURAL? VARS VARIABLES VARIABLE)
```

```
                                        (!LIST VARS)
                                        WHICH WILL BE CHOSEN BY INSTANTIATING THE (PLURAL? LST HYPOTHESES
                                                                                          HYPOTHESIS)
                                        (!PPR-LIST LST)
                                        %.))
                              (BINDINGS (QUOTE NAME)
                                     NAME
                                     (QUOTE VARS)
                                     (SET-DIFF ALL-VARS-HYPS ALL-VARS-CONCL)
                                     (QUOTE LST)
                                     (for HYP in HYPS bind (VARS _ (SET-DIFF ALL-VARS-HYPS ALL-VARS-CONCL))
                                        when (INTERSECTP VARS (ALL-VARS HYP))
                                        collect (PROGN (SETQ VARS (SET-DIFF VARS (ALL-VARS HYP)))
                                                       HYP)))
                              (QUOTE WARNING)))
                       ((AND (ATTEMPT-TO-REWRITE-RECOGNIZER CONCL)
                             HYPS)
                        (ERROR1 (PQUOTE (PROGN (!PPR NAME NIL)
                                        WILL SLOW DOWN THE THEOREM-PROVER BECAUSE IT WILL CAUSE BACKWARD
                                        CHAINING ON EVERY INSTANCE OF A PRIMITIVE TYPE EXPRESSION %.))
                              (BINDINGS (QUOTE NAME)
                                     NAME)
                              (QUOTE WARNING]
                    (for OLD-RULE in (GETPROP (TOP-FNNAME CONCL)
                                              (QUOTE LEMMAS))
                       unless (OR (DISABLEDP (fetch (REWRITE-RULE NAME) of OLD-RULE))
                                  (META-LEMMAP OLD-RULE))
                       do (COND
                             ((SUBSUMES-REWRITE-RULE OLD-RULE REWRITE-RULE)
                              (ERROR1 (PQUOTE (PROGN THE PREVIOUSLY ADDED LEMMA , (!PPR OLDNAME NIL)
                                                     , COULD BE APPLIED WHENEVER THE NEWLY PROPOSED
                                                     (!PPR NAME NIL)
                                                     COULD !))
                                      (BINDINGS (QUOTE NAME)
                                             NAME
                                             (QUOTE OLDNAME)
                                             (fetch (REWRITE-RULE NAME) of OLD-RULE))
                                      (QUOTE WARNING)))
                             ((SUBSUMES-REWRITE-RULE REWRITE-RULE OLD-RULE)
                              (ERROR1 (PQUOTE (PROGN THE NEWLY PROPOSED LEMMA , (!PPR NAME NIL)
                                                     , COULD BE APPLIED WHENEVER THE PREVIOUSLY ADDED LEMMA
                                                     (!PPR OLDNAME NIL)
                                                     COULD %. // //))
                                      (BINDINGS (QUOTE NAME)
                                             NAME
                                             (QUOTE OLDNAME)
                                             (fetch (REWRITE-RULE NAME) of OLD-RULE))
                                      (QUOTE WARNING])
```

(CHK-ACCEPTABLE-SHELL

```
 [LAMBDA (SHELL-NAME BTM-FN-SYMB RECOGNIZER DESTRUCTOR-TUPLES)  (* kbr: "26-Nov-85 15:43")
   (LET (DESTRUCTOR-NAMES NAMES AXIOM-NAMES AC DV TR L FLG)     (* Check that there is a type no available.
                                                                *)
        (NEXT-AVAILABLE-TYPE-NO)
        (for TUPLE in DESTRUCTOR-TUPLES unless (BM-MATCH TUPLE (LIST & & &))
           do (ERROR1 (PQUOTE (PROGN THE DESTRUCTOR-TUPLES ARGUMENT TO ADD-SHELL MUST BE A LIST OF TRIPLES OF
                                     THE FORM (!PPR (QUOTE (NAME (FLG RECOGNIZER ...)
                                                                 DEFAULT-FN-SYMB))
                                                    NIL)
                                     WHERE NAME IS THE NAME OF THE ACCESSOR , FLG IS EITHER ONE-OF
                                       OR NONE-OF , AND DEFAULT-FN-SYMB IS THE FUNCTION SYMBOL FOR THE
                                     DEFAULT VALUE %.))
                      (BINDINGS)
                      (QUOTE SOFT)))
        (SETQ DESTRUCTOR-NAMES (for TUPLE in DESTRUCTOR-TUPLES collect (CAR TUPLE)))
        (SETQ NAMES (CONS SHELL-NAME (CONS RECOGNIZER DESTRUCTOR-NAMES)))
        [COND
           (BTM-FN-SYMB (SETQ NAMES (CONS BTM-FN-SYMB NAMES]
        [for NAME in NAMES
           do (CHK-NEW-NAME NAME NIL)
              (CHK-NEW-*1*NAME NAME)
              (COND
                 ((EQ (CAR (LAST (UNPACK NAME)))
                      (QUOTE -))
                  (ERROR1 (PQUOTE (PROGN HYPHEN , AS IN (!PPR NAME NIL)
                                         , IS NOT ALLOWED AS THE LAST CHARACTER IN A SHELL NAME !))
                          (BINDINGS (QUOTE NAME)
                                 NAME)
                          (QUOTE SOFT]
        [COND
           ((NOT (NO-DUPLICATESP NAMES))
            (ERROR1 (PQUOTE (PROGN MULTIPLE USE OF THE SAME NAME !))
                    (BINDINGS)
                    (QUOTE SOFT]
        [for TUPLE in DESTRUCTOR-TUPLES
           do (BM-MATCH TUPLE (LIST AC TR DV))
```

```
            [COND
               ((AND (NEQ DV (QUOTE TRUE))
                     (NEQ DV (QUOTE FALSE))
                     (NOT (MEMB DV *1*BTM-OBJECTS))
                     (OR (NULL BTM-FN-SYMB)
                         (NEQ DV BTM-FN-SYMB)))
                (ERROR1 (PQUOTE (PROGN THE DEFAULT OBJECT FOR A TYPE-RESTRICTED SHELL COMPONENT MUST BE A
                                       BOTTOM OBJECT FUNCTION SYMBOL OR ELSE MUST BE TRUE OR FALSE
                                  ! (!PPR DV NIL)
                                       IS NOT SUCH AN OBJECT %.))
                        (BINDINGS (QUOTE DV)
                              DV)
                        (QUOTE SOFT]
            [COND
               ([NOT (AND (BM-MATCH TR (CONS FLG L))
                          (OR (EQ FLG (QUOTE ONE-OF))
                              (EQ FLG (QUOTE NONE-OF)))
                          (for X in L always (ASSOC X (CONS (CONS RECOGNIZER 0)
                                                            RECOGNIZER-ALIST]
                (ERROR1 (PQUOTE (PROGN THE TYPE RESTRICTION TERM FOR A SHELL COMPONENT MUST BE A LIST OF THE
                                       FORM (!PPR (QUOTE (ONE-OF ...))
                                                  NIL)
                                         OR (!PPR (QUOTE (NONE-OF ...))
                                                  NIL)
                                       WHERE ... IS A LIST OF RECOGNIZER NAMES %.))
                        NIL
                        (QUOTE SOFT]
            (COND
               ([NOT (OR [AND (EQ DV BTM-FN-SYMB)
                              (OR (AND (EQ FLG (QUOTE ONE-OF))
                                       (MEMB RECOGNIZER L))
                                  (AND (EQ FLG (QUOTE NONE-OF))
                                       (NOT (MEMB RECOGNIZER L]
                         (AND (NEQ DV BTM-FN-SYMB)
                              (EQUAL (EQUAL FLG (QUOTE ONE-OF))
                                     (LOGSUBSETP (CAR (TYPE-PRESCRIPTION DV))
                                                 (for X in L bind (LOOP-ANS _ 0) when (NEQ X RECOGNIZER)
                                                    do [SETQ LOOP-ANS (LOGOR LOOP-ANS (CDR (ASSOC X
                                                                                                  RECOGNIZER-ALIST
                                                                                                  ]
                                                 finally (RETURN LOOP-ANS]
                (ERROR1 (PQUOTE (PROGN THE DEFAULT VALUE (!PPR DV NIL)
                                       DOES NOT SATISFY THE TYPE RESTRICTION (!PPR TR NIL)
                                       SPECIFIED FOR THE (!PPR AC NIL)
                                       COMPONENT %.))
                        (BINDINGS (QUOTE TR)
                              TR
                              (QUOTE DV)
                              DV
                              (QUOTE AC)
                              AC)
                        (QUOTE SOFT]
         [COND
            (DESTRUCTOR-NAMES (for TUPLE in DESTRUCTOR-TUPLES
                                 do (BM-MATCH TUPLE (LIST AC TR DV))
                                    (SETQ AXIOM-NAMES (CONS (PACK (LIST AC "-" SHELL-NAME))
                                                            AXIOM-NAMES))
                                    (SETQ AXIOM-NAMES (CONS (PACK (LIST AC "-N" RECOGNIZER))
                                                            AXIOM-NAMES))
                                    (AND [NOT (EQUAL TR (QUOTE (NONE-OF]
                                         (SETQ AXIOM-NAMES (CONS (PACK (LIST AC "-TYPE-RESTRICTION"))
                                                                 AXIOM-NAMES)))
                                    (SETQ AXIOM-NAMES (CONS (PACK (LIST AC "-LESSP"))
                                                            AXIOM-NAMES))
                                    (SETQ AXIOM-NAMES (CONS (PACK (LIST AC "-LESSEQP"))
                                                            AXIOM-NAMES)))
                 (SETQ AXIOM-NAMES (CONS (PACK (LIST SHELL-NAME "-EQUAL"))
                                         AXIOM-NAMES))
                 (SETQ AXIOM-NAMES (CONS [PACK (CONS SHELL-NAME (for AC in DESTRUCTOR-NAMES
                                                                   join (LIST "-" AC]
                                         AXIOM-NAMES))
                 (SETQ AXIOM-NAMES (CONS (PACK (NCONC1 (CDR (for AC in DESTRUCTOR-NAMES
                                                               join (LIST "-" AC)))
                                                       "-ELIM"))
                                         AXIOM-NAMES))
                 (SETQ AXIOM-NAMES (CONS (PACK (LIST "COUNT-" SHELL-NAME))
                                         AXIOM-NAMES]
         [COND
            ((NOT (NO-DUPLICATESP (APPEND NAMES AXIOM-NAMES)))
             (ERROR1 (PQUOTE (PROGN THE ADDITION OF A SHELL INTRODUCES MANY NEW AXIOM NAMES %. THE NEW NAMES
                                    ARE CREATED FROM THE SHELL NAME , RECOGNIZER , BOTTOM OBJECT ,
                                      AND DESTRUCTOR NAMES SUPPLIED IN THE ADD-SHELL COMMAND %. THE NAMES
                                    SUPPLIED IN THIS INSTANCE OF THE ADD-SHELL COMMAND DO NOT LEAD TO DISTINCT
                                    AXIOM NAMES %. THE AXIOM NAMES GENERATED ARE : (!LIST AXIOM-NAMES)
                                    %.))
                     (BINDINGS (QUOTE AXIOM-NAMES)
                           AXIOM-NAMES)
```

```
                (QUOTE SOFT]
         (for X in AXIOM-NAMES do (CHK-NEW-NAME X NIL))
         T])
```

(CHK-ACCEPTABLE-TOGGLE

```
  [LAMBDA (NAME OLDNAME FLG)                                  (* kbr: "19-Oct-85 16:31")
    (CHK-NEW-NAME NAME NIL)
    (MAIN-EVENT-OF OLDNAME)
    (OR (EQ FLG T)
        (EQ FLG NIL)
        (ERROR1 (PQUOTE (PROGN THE THIRD ARGUMENT OF TOGGLE MUST BE T OR NIL AND (!PPR FLG NIL)
                                                                                  IS NOT %.))
                (BINDINGS (QUOTE FLG)
                       FLG)
                (QUOTE SOFT])
```

(CHK-ARGLIST

```
  [LAMBDA (NAME ARGS)                                         (* kbr: "19-Oct-85 16:31")
    (COND
       ((OR (NOT (NO-DUPLICATESP ARGS))
            [for ARG in ARGS thereis (OR (ILLEGAL-NAME ARG)
                                         (MEMB ARG (QUOTE (T F NIL]
            (CDR (LAST ARGS)))                                (* T and F are merely confusing, not illegal.
                                                              *)
        (ERROR1 (PQUOTE (PROGN THE ARGUMENT LIST TO (!PPR NAME NIL)
                               , I.E., (!PPR ARGS NIL)
                               , IS NOT A LIST OF DISTINCT VARIABLES NAMES %.))
                (BINDINGS (QUOTE ARGS)
                       ARGS
                       (QUOTE NAME)
                       NAME)
                (QUOTE SOFT])
```

(CHK-MEANING

```
  [LAMBDA (NAME LST)                                          (* kbr: "19-Oct-85 16:31")
    (LET (FNS)
         (SETQ FNS (INTERSECTION LST META-NAMES))
         [COND
            (FNS (ERROR1 (PQUOTE (PROGN USE OF THE (PLURAL? FNS FUNCTIONS FUNCTION)
                                        (!LIST FNS)
                                        IN AN AXIOM OR DEFINITION MAY RENDER THE THEORY INCONSISTENT %.))
                         (BINDINGS (QUOTE FNS)
                                FNS)
                         (QUOTE WARNING]
         NIL])
```

(CHK-NEW-*1*NAME

```
  [LAMBDA (NAME)                                              (* kbr: "22-Oct-85 15:57")
    (COND
       ([OR [NOT (LITATOM (PACK (LIST STRING-WEIRD NAME]
            (AND (NOT IN-BOOT-STRAP-FLG)
                 (OR (GETD (PACK (LIST STRING-WEIRD NAME)))
                     (HAS-LIB-PROPS (PACK (LIST STRING-WEIRD NAME]
        (ERROR1 (PQUOTE (PROGN THE NLISTP (!PPR FN NIL)
                               , WHICH IS DERIVED FROM (!PPR NAME NIL) AND USED FOR INTERNAL PURPOSES , IS NOT
                               A LITERAL ATOM, HAS A LISP FUNCTION DEFINITION OR LIB-PROP PROPERTIES %. YOU
                                                                                SHOULD CHANGE THE NAME OF
                               YOUR FUNCTION TO AVOID CLASHES OF THIS SORT %.))
                (BINDINGS (QUOTE NAME)
                       NAME
                       (QUOTE FN)
                       (PACK (LIST STRING-WEIRD NAME)))
                (QUOTE SOFT])
```

(CHK-NEW-NAME

```
  [LAMBDA (NAME QUIET-FLG)                                    (* kbr: "24-Oct-85 18:11")

          (* Checks that NAME has the correct syntax for use as a symbol in the theory
          (and hence as an event name)%. Further checks that the name has no properties and is not one of the symbols about which
          there are syntactic conventions (e.g., LIST, CADR, NIL, QUOTE)%.
          Thus there are no axioms about NAME. *)

    (COND
       [(ILLEGAL-NAME NAME)
        (COND
           (QUIET-FLG NIL)
           (T (ERROR1 (PQUOTE (PROGN (!PPR NAME NIL)
                                     IS AN ILLEGAL OBJECT TO USE FOR A NAME !))
                      (BINDINGS (QUOTE NAME)
                             NAME)
                      (QUOTE SOFT]
       [(PROPERTYLESS-SYMBOLP NAME)
```

```
        (COND
           (QUIET-FLG NIL)
           (T (ERROR1 (PQUOTE (PROGN THE NAME (!PPR NAME NIL)
                                     IS A RESERVED SYMBOL AND CANNOT BE USED AS A USER NAME %.))
                      (BINDINGS (QUOTE NAME)
                            NAME)
                      (QUOTE SOFT]
       [(HAS-LIB-PROPS NAME)
        (COND
           (QUIET-FLG NIL)
           (T (ERROR1 (PQUOTE (PROGN NAME CURRENTLY IN USE : (!PPR NAME NIL)
                                     %.))
                      (BINDINGS (QUOTE NAME)
                            NAME)
                      (COND
                         (IN-BOOT-STRAP-FLG (QUOTE WARNING))
                         (T (QUOTE SOFT]
       (T T])
```

(CLAUSIFY

```
  [LAMBDA (TERM)                                                    (* kbr: "19-Oct-85 16:31")
    (COND
       ((EQUAL TERM TRUE)
        NIL)
       ((EQUAL TERM FALSE)
        (LIST NIL))
       ((FNNAMEP-IF TERM)
        (CLEAN-UP-BRANCHES (STRIP-BRANCHES TERM)))
       (T (LIST (LIST TERM])
```

(CLAUSIFY-INPUT

```
  [LAMBDA (TERM)                                                    (* kbr: "19-Oct-85 16:31")
```

(* In addition to clausifying TERM, we expand ANDs in the hyps and ORs in the concl, adding entries to ABBREVIATIONS-USED. *)

```
    (for TERM1 in (CLAUSIFY-INPUT1 TERM FALSE) collect (CLAUSIFY-INPUT1 (DUMB-NEGATE-LIT TERM1)
                                                                        TRUE])
```

(CLAUSIFY-INPUT1

```
  [LAMBDA (TERM BOOL)                                               (* kbr: "19-Oct-85 16:31")
```

(* If BOOL is TRUE, returns a list of terms whose disjunction is equivalent to TERM.
IF BOOL is FALSE, returns a list of terms whose disjunction is equivalent to the negation of TERM.
Opens up some nonrec fns and applies some unconditional rewrite rules --
according to BOOL -- and side-effects ABBREVIATIONS-USED.
*)

```
    (LET (C1 C2 C3)
         (COND
            ((EQUAL TERM (BM-NEGATE BOOL))
             NIL)
            [[BM-MATCH TERM (COND
                               ((C1 C2 C3]
             (COND
                [(EQUAL BOOL TRUE)
                 (COND
                    ((EQUAL C3 TRUE)
                     (DISJOIN-CLAUSES (CLAUSIFY-INPUT1 C1 FALSE)
                            (CLAUSIFY-INPUT1 C2 TRUE)))
                    ((EQUAL C2 TRUE)
                     (DISJOIN-CLAUSES (CLAUSIFY-INPUT1 C1 TRUE)
                            (CLAUSIFY-INPUT1 C3 TRUE)))
                    (T (LIST TERM]
                (T (COND
                      ((EQUAL C3 FALSE)
                       (DISJOIN-CLAUSES (CLAUSIFY-INPUT1 C1 FALSE)
                              (CLAUSIFY-INPUT1 C2 FALSE)))
                      ((EQUAL C2 FALSE)
                       (DISJOIN-CLAUSES (CLAUSIFY-INPUT1 C1 TRUE)
                              (CLAUSIFY-INPUT1 C3 FALSE)))
                      (T (LIST (DUMB-NEGATE-LIT TERM]
            ((SETQ C1 (EXPAND-AND-ORS TERM BOOL))
             (CLAUSIFY-INPUT1 C1 BOOL))
            ((EQUAL BOOL FALSE)
             (LIST (DUMB-NEGATE-LIT TERM)))
            (T (LIST TERM])
```

(CLEAN-UP-BRANCHES

```
  [LAMBDA (LST)                                                     (* kbr: "19-Oct-85 19:59")
    (LET (PARTITIONS)
         (SETQ PARTITIONS (PARTITION-CLAUSES LST))
         (SETQ TEMP-TEMP (for POCKET in PARTITIONS join (ALMOST-SUBSUMES-LOOP POCKET)))
```

```
         (COND
            ((NULL (CDR PARTITIONS))
             TEMP-TEMP)
            (T (ALMOST-SUBSUMES-LOOP TEMP-TEMP])
```

(CNF-DNF

```
  [LAMBDA (TERM FLG)                                                    (* kbr: "19-Oct-85 16:31")

          (* If FLG is (QUOTE C), returns a list of lists, say: ((p11 p12 ...)
          (p21 p22 ...) ... (pn1 pn2 ...)) such that TERM is not equal to F iff
          (AND (OR p11 p12 ...) (OR p21 p22 ...) ... (OR pn1 pn2 ...)) is not equal to F.
          The latter term is the TERM. If FLG is (QUOTE D) computes the disjunctive normal form.
          *)

    (LET (P Q NF-Q)
         (COND
            ([OR (AND (EQ FLG (QUOTE C))
                      (BM-MATCH TERM (AND P Q)))
                 (AND (EQ FLG (QUOTE D))
                      (BM-MATCH TERM (OR P Q]
             (APPEND (CNF-DNF P FLG)
                     (CNF-DNF Q FLG)))
            ([OR (AND (EQ FLG (QUOTE C))
                      (BM-MATCH TERM (OR P Q)))
                 (AND (EQ FLG (QUOTE D))
                      (BM-MATCH TERM (AND P Q]
             (SETQ NF-Q (CNF-DNF Q FLG))
             (for L1 in (CNF-DNF P FLG) bind LOOP-ANS do (SETQ LOOP-ANS (UNION-EQUAL (for L2 in NF-Q
                                                                                       collect (UNION-EQUAL L1 L2)
                                                                                       )
                                                                                 LOOP-ANS))
                 finally (RETURN LOOP-ANS)))
            [(BM-MATCH TERM (NOT P))
             (for L1 in (CNF-DNF P (SELECTQ FLG
                                            (D (QUOTE C))
                                            (QUOTE D)))
                 collect (for TERM in L1 collect (DUMB-NEGATE-LIT TERM]
            ((BM-MATCH TERM (IMPLIES P Q))
             (CNF-DNF (FCONS-TERM* (QUOTE OR)
                                   (DUMB-NEGATE-LIT P)
                                   Q)
                      FLG))
            (T (LIST (LIST TERM])
```

(COMMON-SWEEP

```
  [LAMBDA (FORM)                                                        (* kbr: "19-Oct-85 16:31")
    (LET (VAR DECISION)
         (COND
            ((OR (NLISTP FORM)
                 (EQ (CAR FORM)
                     (QUOTE QUOTE)))
             FORM)
            [(SETQ DECISION (ASSOC FORM DECISIONS))
             (SETQ VAR (CDR (SASSOC FORM VAR-ALIST)))
             (SUBLIS [LIST (CONS (QUOTE VAR)
                                 VAR)
                           (CONS (QUOTE FORM)
                                 (CONS (CAR FORM)
                                       (for ARG in (CDR FORM) collect (COMMON-SWEEP ARG]
                     (SELECTQ (CDR DECISION)
                          (TEST-AND-SET (QUOTE (*2*IF (NEQ VAR (QUOTE *1*X))
                                                      VAR
                                                      (SETQ VAR FORM))))
                          (SET (QUOTE (SETQ VAR FORM)))
                          (TEST (QUOTE (*2*IF (NEQ VAR (QUOTE *1*X))
                                              VAR FORM)))
                          (VAR (QUOTE VAR))
                          (ERROR (LIST (QUOTE COMMON-SWEEP)
                                       (CDR DECISION]
            (T (CONS (CAR FORM)
                     (for ARG in (CDR FORM) collect (COMMON-SWEEP ARG])
```

(COMMUTE-EQUALITIES

```
  [LAMBDA (TERM)                                                        (* kbr: "19-Oct-85 16:31")
    (COND
       ((VARIABLEP TERM)
        TERM)
       ((FQUOTEP TERM)
        TERM)
       ((EQ (FFN-SYMB TERM)
            (QUOTE EQUAL))
        (FCONS-TERM* (QUOTE EQUAL)
               (FARGN TERM 2)
               (FARGN TERM 1)))
```

```
      (T (CONS-TERM (CAR TERM)
                    (for ARG in (FARGS TERM) collect (COMMUTE-EQUALITIES ARG])
```

(COMPARE-STATS

```
  [LAMBDA (ALIST-NEW ALIST-OLD TOTALS-NEW TOTALS-OLD LST)        (* kbr: "24-Oct-85 16:39")

          (* LST is a 4 tuple of integers indicating which events are considered interesting.
          The first two numbers deal with the cons counts, the second two with the cpu times.
          The first and third integers are percentages by which the appropriate measures in NEW should differ from those of OLD.
          The second and fourth are the absolute value of the differences between NEW and OLD.
          An event must be both relatively and absolutely interesting to be printed out.
          *)

    [OR LST (SETQ LST (QUOTE (10 1000 10 30]
    (COND
       ([NOT (AND (IEQP (LENGTH ALIST-NEW)
                        (LENGTH ALIST-OLD))
                  (for PAIR-NEW in ALIST-NEW as PAIR-OLD in ALIST-OLD always (EQUAL (CAR PAIR-NEW)
                                                                                    (CAR PAIR-OLD]
        [SETQ ALIST-NEW (SORT (for PAIR-NEW in ALIST-NEW when (for PAIR-OLD in ALIST-OLD
                                                                  thereis (EQUAL (CADAR PAIR-NEW)
                                                                                 (CADAR PAIR-OLD)))
                                  collect PAIR-NEW)
                              (FUNCTION (LAMBDA (X Y)
                                                (ALPHORDER (CADAR X)
                                                           (CADAR Y]
        [SETQ ALIST-OLD (SORT (for PAIR-OLD in ALIST-OLD when (for PAIR-NEW in ALIST-NEW
                                                                  thereis (EQUAL (CADAR PAIR-NEW)
                                                                                 (CADAR PAIR-OLD)))
                                  collect PAIR-OLD)
                              (FUNCTION (LAMBDA (X Y)
                                                (ALPHORDER (CADAR X)
                                                           (CADAR Y]
        (SETQ TOTALS-NEW (SUM-STATS-ALIST ALIST-NEW))
        (SETQ TOTALS-OLD (SUM-STATS-ALIST ALIST-OLD))
        (PRINEVAL (PQUOTE (PROGN THE TWO FILES DO NOT CONTAIN THE SAME SEQUENCE OF EVENT NAMES %. THE
                                 COMPARISON WILL BE ON THE INTERSECTION OF THE TWO SEQUENCES , WHICH HAS LENGTH
                                 (!PPR N NIL)
                                 %. // //))
                  (BINDINGS (QUOTE N)
                            (LENGTH ALIST-NEW))
                  0 T)))
    (IPRINC "OLD =" T)
    (IPRINC "NEW =" T)
    (ITERPRI T)
    (IPRINC "CONSES CPU GC IO ELAPSED" T)
    (ITERPRI T)
    (ITERPRI T)
    (for X in (LIST TOTALS-NEW TOTALS-OLD) as Y in (QUOTE (NEW OLD))
       do (IPRINC Y T)
          (for PAIR in X as Z in (QUOTE ((FIX 12)
                                         (FLOAT 9 1)
                                         (FLOAT 9 1)
                                         (FLOAT 9 1)
                                         (FIX 12)))
             do (PRIN2 (CADR PAIR)
                       T)
                (SPACES 3 T))
          (ITERPRI T))
    (ITERPRI T)
    (IPRINC "N/O" T)
    (for X in TOTALS-NEW as Y in TOTALS-OLD as Z in (QUOTE ((FLOAT 12 1)
                                                            (FLOAT 9 1)
                                                            (FLOAT 9 1)
                                                            (FLOAT 9 1)
                                                            (FLOAT 12 1)))
       do (PRIN2 (QUOTIENT (CADR X)
                           (CADR Y))
                 T)
          (SPACES 3 T))
    (ITERPRI T)
    (PRINEVAL (PQUOTE (PROGN // // EVENTS WHOSE CONS COUNTS ARE MORE THAN (!PPR X NIL)
                             PERCENT AND (!PPR N NIL)
                                         CONSES DIFFERENT, PRINTED AS IQUOTIENT (N//O NEW OLD IQUOTIENT)
                               AND ORDERED BY N//O : //))
              (BINDINGS (QUOTE X)
                        (CAR LST)
                        (QUOTE N)
                        (CADR LST))
              0 T)
    (PPRIND [DREVERSE (SORT [for X in ALIST-NEW as Y in ALIST-OLD
                               when [AND (OR (GREATERP (QUOTIENT (CADR X)
                                                                 (CADR Y))
                                                       (QUOTIENT (PLUS 100 (CAR LST))
                                                                 100))
                                             (LESSP (QUOTIENT (CADR X)
```

```
                                                    (CADR Y))
                                                (QUOTIENT (DIFFERENCE 100 (CAR LST))
                                                       100)))
                                        (OR (GREATERP (CADR X)
                                                (PLUS (CADR Y)
                                                       (CADR LST)))
                                            (LESSP (PLUS (CADR X)
                                                       (CADR LST))
                                                (CADR Y]
                                    collect (LIST (QUOTIENT (CADR X)
                                                         (CADR Y))
                                                  (CONS (CADR (CAR X))
                                                        (CDR X))
                                                  (CONS (CADR (CAR Y))
                                                        (CDR Y]
                                (FUNCTION (LAMBDA (X Y)
                                                  (LESSP (CAR X)
                                                         (CAR Y]
        0 0 NIL T)
    (ITERPRI T)
    (PRINEVAL (PQUOTE (PROGN // // EVENTS WHOSE CPU TIMES ARE MORE THAN (!PPR X NIL)
                              PERCENT AND (!PPR N NIL)
                                       SECONDS DIFFERENT, PRINTED AS IQUOTIENT (N//O NEW OLD IQUOTIENT)
                                 AND ORDERED BY N//O : //))
           (BINDINGS (QUOTE X)
                  (CADDR LST)
                  (QUOTE N)
                  (CADDDR LST))
           0 T)
    (PPRIND [DREVERSE (SORT [for X in ALIST-NEW as Y in ALIST-OLD
                               when [AND (OR (GREATERP (QUOTIENT (CADDR X)
                                                                 (CADDR Y))
                                                       (QUOTIENT (PLUS 100 (CADDR LST))
                                                              100))
                                             (LESSP (QUOTIENT (CADDR X)
                                                              (CADDR Y))
                                                       (QUOTIENT (DIFFERENCE 100 (CADDR LST))
                                                              100)))
                                         (OR (GREATERP (CADDR X)
                                                       (PLUS (CADDR Y)
                                                              (CADDDR LST)))
                                             (LESSP (PLUS (CADDR X)
                                                              (CADDDR LST))
                                                       (CADDR Y]
                               collect (LIST (QUOTIENT (CADDR X)
                                                       (CADDR Y))
                                             (CONS (CADR (CAR X))
                                                   (CDR X))
                                             (CONS (CADR (CAR Y))
                                                   (CDR Y]
                            (FUNCTION (LAMBDA (X Y)
                                              (LESSP (CAR X)
                                                     (CAR Y]
           0 0 NIL T)
    (ITERPRI T])
```

(COMPLEMENTARY-MULTIPLEP

```
[LAMBDA (WINNING-PAIR POLY1 POLY2)                                    (* kbr: "19-Oct-85 16:31")
```

(* Return T iff multiplying POLY1 by some negative integer produces POLY2.
WINNING-PAIR is a member of POLY1 with coefficient IPLUS or -1.0 *)

```
    (PROG (FACTOR)
          (COND
             ([NULL (SETQ TEMP-TEMP (SASSOC (CAR WINNING-PAIR)
                                            (fetch (POLY ALIST) of POLY2]
              (RETURN NIL)))
          [SETQ FACTOR (COND
                          ((EQUAL (CDR WINNING-PAIR)
                                  1)
                           (CDR TEMP-TEMP))
                          (T (MINUS (CDR TEMP-TEMP]
          (COND
             ((NOT (LESSP FACTOR 0))
              (RETURN NIL)))
          (RETURN (AND (EQUAL (fetch (POLY CONSTANT) of POLY2)
                              (TIMES FACTOR (fetch (POLY CONSTANT) of POLY1)))
                       (IEQP (LENGTH (fetch (POLY ALIST) of POLY2))
                             (LENGTH (fetch (POLY ALIST) of POLY1)))
                       (for PAIR1 in (fetch (POLY ALIST) of POLY1) as PAIR2 in (fetch (POLY ALIST) of POLY2)
                          always (AND (EQUAL (CAR PAIR1)
                                             (CAR PAIR2))
                                      (EQUAL (CDR PAIR2)
                                             (TIMES FACTOR (CDR PAIR1])
```

(COMPLEMENTARYP

```
[LAMBDA (LIT1 LIT2)                                    (* kbr: "20-Oct-85 13:43")
                                                       (* Is LIT2 the syntactic NOT of LIT1? *)

  (OR (AND (NVARIABLEP LIT1)
           (NOT (FQUOTEP LIT1))
           (EQ (FFN-SYMB LIT1)
               (QUOTE NOT))
           (EQUAL (FARGN LIT1 1)
                  LIT2))
      (AND (NVARIABLEP LIT2)
           (NOT (FQUOTEP LIT2))
           (EQ (FFN-SYMB LIT2)
               (QUOTE NOT))
           (EQUAL (FARGN LIT2 1)
                  LIT1])
```

(COMPLEXITY

```
[LAMBDA (TERM)                                         (* kbr: "24-Oct-85 15:49")
  (COND
     ((VARIABLEP TERM)
      0)
     ((FQUOTEP TERM)

        (* The level number of all function symbols in evgs is 0, so even if we recursed into them with FN-SYMBs and ARGS we'd
        compute 0.0 *)

      0)
     (T (IPLUS (GET-LEVEL-NO (FFN-SYMB TERM))
               (PROG (MAX)
                     (SETQ MAX 0)
                     (for ARG in (FARGS TERM) do (SETQ MAX (IMAX (COMPLEXITY ARG)
                                                                 MAX)))
                     (RETURN MAX])
```

(COMPRESS-POLY

```
[LAMBDA (POLY)                                         (* kbr: "19-Oct-85 16:31")
  [COND
     ((IMPOSSIBLE-POLYP POLY)
      (replace (POLY ALIST) of POLY with NIL))
     ((TRUE-POLYP POLY)
      (replace (POLY ALIST) of POLY with NIL))
     (T (replace (POLY ALIST) of POLY with (COMPRESS-POLY1 (fetch (POLY ALIST) of POLY]
  POLY])
```

(COMPRESS-POLY1

```
[LAMBDA (ALIST)                                        (* kbr: "20-Oct-85 15:44")
                                                       (* Return ALIST with buckets whose CDRs are 0 removed.
                                                       *)

  (COND
     ((NLISTP ALIST)
      NIL)
     ((EQUAL (CDR (CAR ALIST))
             0)
      (COMPRESS-POLY1 (CDR ALIST)))
     (T (RPLACD ALIST (COMPRESS-POLY1 (CDR ALIST])
```

(COMPUTE-VETOES

```
[LAMBDA (CANDLST)                                      (* kbr: "19-Oct-85 16:31")

        (* This function weeds out behind the notion competing with it for instantiation of its variables.
        What we actually do is throw out any candidate whose changing induction variables --
        that is the induction variables as computed by INDUCT-VARS intersected with the changed vars of candidate --
        intersect the changed or unchanged variables of another candidate.
        The reason we do not care about the first candidates unchanging vars is as follows.
        The reason you want a candidate clean is so that the terms riding on that cand will reoccur in both the hypothesis and
        conclusion of an induction. There are two ways to assure (or at least make likely) this, change the variables in the terms as
        specified or leave them constant. Thus, if the first cands changing vars are clean but its unchanging vars intersect another
        cand it means that the first cand is keeping those other terms constant which is fine.
        (Note that the first cand would be clean here. The second might be clean or dirty depending on whether its changed vars or
        unchanged vars intersected the first cands vars.) The reason we check only the induction vars and not all of the changed
        vars is if cand1's changed vars include some induction vars and some accumulators and the accumulators are claimed by
        another cand2 we believe that cand1 is still clean. The motivating example was
        (IMPLIES (MEMBER A C) (MEMBER A (UNION: B C))) where the induction on C is dirty because the induction on B and C
        claims C, but the induction on B and C is clean because the B does not occur in the C induction.
        We do not even bother to check the C from the (B C) induction because since it is necessarily an accumulator it is probably
        being constructed and thus, if it occurs in somebody elses ind vars it is probably being eaten so it will be ok.
        In formulating this heuristic we did not consider the possibility that the accums of one candidate occur as constants in the
        other. Oh well. JULY 20, 1978.0 We have added an additional heuristic, to be applied if the above one eliminates all cands.
        We consider a cand flawed if it changes anyone elses constants.
        The motivating example was GREATEST-FACTOR-LESSP -- which was previously proved only by virtue of a very ugly use
        of the no-op fn ID to make a certain induction flawed. *)

  (OR (for CAND1 in CANDLST bind CHANGING-INDVARS
```

```
          unless [PROGN (SETQ CHANGING-INDVARS (INTERSECTION (fetch (CANDIDATE CHANGED-VARS) of CAND1)
                                                             (INDUCT-VARS CAND1)))
                        (for CAND2 in CANDLST when (NEQ CAND1 CAND2)
                           thereis (OR (INTERSECTP CHANGING-INDVARS (fetch (CANDIDATE CHANGED-VARS) of CAND2))
                                       (INTERSECTP CHANGING-INDVARS (fetch (CANDIDATE UNCHANGEABLE-VARS)
                                                                           of CAND2]
          collect CAND1)
       (for CAND1 in CANDLST bind CHANGING-VARS unless [PROGN (SETQ CHANGING-VARS (fetch (CANDIDATE CHANGED-VARS)
                                                                                        of CAND1))
                                                              (for CAND2 in CANDLST when (NEQ CAND1 CAND2)
                                                                 thereis (INTERSECTP CHANGING-VARS
                                                                                     (fetch (CANDIDATE UNCHANGEABLE-VARS)
                                                                                        of CAND2]
          collect CAND1)
       CANDLST])
```

(COMSUBT1

```
[LAMBDA (T1)                                                    (* kbr: "19-Oct-85 16:31")
```

(* We add to GENRLTLIST every common subterm t of T1 and T2 such that t has property p, and no subterm of t has property p. Property (p ITIMES) is ITIMES is not a variable and the function symbol of ITIMES is not a btm object, constructor, or destructor. We return T iff T1 is a common subterm of T2, but neither T1 nor any subterm of T1 has property p. *)

```
   (PROG (FAILED)
         [COND
            ((OR (VARIABLEP T1)
                 (FQUOTEP T1))
             (RETURN (OCCUR T1 T2]
```

(* After the following FOR, FAILED is set to T iff COMSUBT1 returned NIL on at least one of the arguments of T1. GENRLTLIST now contains all of proper subterms of T1 that occur in T2, have property p, and have no subterms with property p, by inductive hypothesis. *)

```
         (for ARG in (FARGS T1) when (NOT (COMSUBT1 ARG)) do (SETQ FAILED T))
         (COND
            (FAILED
```

(* One of T1's arguments returned NIL. So either the argument is not a subterm of T2, in which case neither is T1, or the argument or one of its subterms has property p, in which case one of T1's subterms also has property p.
So we return NIL and do not add T1 to GENRLTLIST. *)

```
                    (RETURN NIL))
            ((NOT (OCCUR T1 T2))
```

(* If T1 does not occur in T2, then its not a common subterm -- regardless of what properties its args have --
and so we return NIL and do not add T1 to GENRLTLIST. *)

```
             (RETURN NIL))
            ([AND (NOT (SHELLP T1))
                  (NOT (AND (SETQ TEMP-TEMP (GETPROP (FFN-SYMB T1)
                                                     (QUOTE ELIMINATE-DESTRUCTORS-SEQ)))
                            (NOT (DISABLEDP (fetch (REWRITE-RULE NAME) of TEMP-TEMP]
```

(* The test above checks that T1 has property p. We know that T1 occurs in T2.
We also know that every argument of T1 recursively returned T and so no argument nor any subterm has property p.
Therefore we add T1 to GENRLTLIST. We return NIL because T1 has property p.
*)

```
             (SETQ GENRLTLIST (ADD-TO-SET T1 GENRLTLIST))
             (RETURN NIL))
            (T
```

(* T1 does not have property p. It is a subterm of T2, and no subterm of it has property p.
*)

```
               (RETURN T])
```

(COMSUBTERMS

```
[LAMBDA (T1 T2)                                                 (* kbr: "19-Oct-85 16:31")
```

(* We add to GENRLTLIST every common subterm t of T1 and T2 such that t has property p, and no subterm of t has property p. Property (p ITIMES) is ITIMES is not a variable and the function symbol of ITIMES is not a btm object, constructor, or destructor. *)

```
   (COND
      ((GREATERP (COUNT T1)
                 (COUNT T2))
       (swap T1 T2)))
   (COMSUBT1 T1])
```

(CONJOIN

```
[LAMBDA (LST IF-FLG)                                            (* kbr: "19-Oct-85 16:31")
   (COND
```

```
       ((NULL LST)
        TRUE)
       (T (CONJOIN2 (CAR LST)
                    (CONJOIN (CDR LST)
                             IF-FLG)
                    IF-FLG])
```

(CONJOIN-CLAUSE-SETS

```
  [LAMBDA (LST1 LST2)                                          (* kbr: "19-Oct-85 16:31")
    (LET (ANS)
         (for CL in LST1 when (AND (NOT (EQUAL CL TRUE-CLAUSE))
                                   (NOT (MEMBER CL ANS)))
            do (SETQ ANS (CONS CL ANS)))
         (for CL in LST2 when (AND (NOT (EQUAL CL TRUE-CLAUSE))
                                   (NOT (MEMBER CL ANS)))
            do (SETQ ANS (CONS CL ANS)))
        ANS])
```

(CONJOIN2

```
  [LAMBDA (P Q IF-FLG)                                         (* kbr: "19-Oct-85 16:31")
    (COND
       ((FALSE-NONFALSEP P)
        (COND
           (DEFINITELY-FALSE FALSE)
           ((FALSE-NONFALSEP Q)
            (COND
               (DEFINITELY-FALSE FALSE)
               (T TRUE)))
           ((NOT (BOOLEAN Q))
            (FCONS-TERM* (QUOTE IF)
                  Q TRUE FALSE))
           (T Q)))
       [(FALSE-NONFALSEP Q)
        (COND
           (DEFINITELY-FALSE FALSE)
           ((BOOLEAN P)
            P)
           (T (FCONS-TERM* (QUOTE IF)
                     P TRUE FALSE]
       (IF-FLG (FCONS-TERM* (QUOTE IF)
                     P
                     (COND
                        ((BOOLEAN Q)
                         Q)
                        (T (FCONS-TERM* (QUOTE IF)
                                  Q TRUE FALSE)))
                     FALSE))
       (T (FCONS-TERM* (QUOTE AND)
                 P Q])
```

(CONS-PLUS

```
  [LAMBDA (X Y)                                                (* kbr: "19-Oct-85 16:31")
    (COND
       ((EQUAL X ZERO)
        Y)
       ((EQUAL Y ZERO)
        X)
       (T (FCONS-TERM* (QUOTE PLUS)
                 X Y])
```

(CONS-TERM

```
  [LAMBDA (FN ARGS)                                            (* kbr: "19-Oct-85 16:31")
```

(* After great deliberation, we have decided to guarantee throughout the theorem-prover that every explicit value term should be represented as an evg. Unless the function symbol of a term being constructed is known not to be a constructor or bottom object, the term should be constructed using CONS-TERM rather than with FCONS-TERM or FCONS-TERM*. *)

```
    (COND
       [(AND (for ARG in ARGS always (QUOTEP ARG))
             (OR (MEMB FN *1*BTM-OBJECTS)
                 (ASSOC FN SHELL-ALIST)))
```

(* We wish to apply the LISP-CODE for this shell constructor or btm object to the guts of each arg and QUOTE the result. To avoid having to cons up the list of guts, we will consider the common cases separately. *)

```
        (COND
           [(NULL ARGS)
            (LIST (QUOTE QUOTE)
                  (APPLY* (GETPROP FN (QUOTE LISP-CODE]
           [(NULL (CDR ARGS))
            (LIST (QUOTE QUOTE)
```

```
                (APPLY* (GETPROP FN (QUOTE LISP-CODE))
                        (CADR (CAR ARGS]
          [(NULL (CDDR ARGS))
           (LIST (QUOTE QUOTE)
                 (APPLY* (GETPROP FN (QUOTE LISP-CODE))
                         (CADR (CAR ARGS))
                         (CADR (CADR ARGS]
          [(NULL (CDDDR ARGS))
           (LIST (QUOTE QUOTE)
                 (APPLY* (GETPROP FN (QUOTE LISP-CODE))
                         (CADR (CAR ARGS))
                         (CADR (CADR ARGS))
                         (CADR (CADDR ARGS]
          (T (LIST (QUOTE QUOTE)
                   (APPLY (GETPROP FN (QUOTE LISP-CODE))
                          (for ARG in ARGS collect (CADR ARG]
      (T (CONS FN ARGS])
```

(CONSJOIN

```
[LAMBDA (LST)                                          (* kbr: "19-Oct-85 16:31")
  (COND
     ((NLISTP (CDR LST))
      (CAR LST))
     (T (CONS-TERM (QUOTE CONS)
                   (LIST (CAR LST)
                         (CONSJOIN (CDR LST])
```

(CONTAINS-REWRITEABLE-CALLP

```
[LAMBDA (NAME TERM)                                    (* kbr: "19-Oct-85 16:31")
```

(* This function scans the nonQUOTE part of TERM and determines whether it contains a call of NAME not on TERMS-TO-BE-IGNORED-BY-REWRITE. *)

```
  (COND
     ((VARIABLEP TERM)
      NIL)
     ((FQUOTEP TERM)
      NIL)
     ((AND (EQ (FFN-SYMB TERM)
               NAME)
           (NOT (MEMBER TERM TERMS-TO-BE-IGNORED-BY-REWRITE)))
      T)
     (T (for X in (FARGS TERM) thereis (CONTAINS-REWRITEABLE-CALLP NAME X])
```

(CONVERT-CAR-CDR

```
[LAMBDA (X)                                            (* kbr: "19-Oct-85 16:31")
  (LET (ANS (GUTS X))
       (while (OR (BM-MATCH GUTS (CAR &))
                  (BM-MATCH GUTS (CDR &)))
          do (SETQ ANS (CONS (NTHCHAR (CAR GUTS)
                                      2)
                             ANS))
             (SETQ GUTS (CADR GUTS)))
       (COND
          ((IGREATERP (LENGTH ANS)
                      1)
           (LIST [PACK (CONS (QUOTE C)
                             (DREVERSE (CONS (QUOTE R)
                                             ANS]
                 GUTS))
          (T X])
```

(CONVERT-CONS

```
[LAMBDA (X)                                            (* kbr: "19-Oct-85 16:31")
  (COND
     ((SETQ TEMP-TEMP (LISTABLE X))
      (APPEND (QUOTE (LIST))
              TEMP-TEMP))
     (T X])
```

(CONVERT-NOT

```
[LAMBDA (X)                                            (* kbr: "19-Oct-85 16:31")
  (LET (U V)
       (COND
          ((BM-MATCH X (NOT (LESSP U V)))
           (LIST (QUOTE LEQ)
                 V U))
          (T X])
```

(CONVERT-QUOTE

```
[LAMBDA (X)                                            (* kbr: "19-Oct-85 16:31")
```

```
    (SETQ PPR-MACRO-LST NIL)
    (COND
       ((EQ (CADR X)
            *1*T)
        T)
       ((EQ (CADR X)
            *1*F)
        (QUOTE F))
       ((FIXP (CADR X))
        (CADR X))
       ((EQ (CADR X)
            NIL)
        NIL)
       [(AND (LISTP (CADR X))
             (EQ (CAR (CADR X))
                 *1*SHELL-QUOTE-MARK))
        (CONS (CADR (CADR X))
              (for ARG in (CDDR (CADR X)) collect (CONVERT-QUOTE (LIST (QUOTE QUOTE)
                                                                       ARG]
       (T X])
```

(CONVERT-TYPE-NO-TO-RECOGNIZER-TERM

```
  [LAMBDA (TYPE-NO ARG)                                    (* kbr: "19-Oct-85 16:31")
    (LET (TYPE-SET)
         (SETQ TYPE-SET (LOGBIT TYPE-NO))
         (COND
            ((SETQ TEMP-TEMP (for PAIR in RECOGNIZER-ALIST when (IEQP TYPE-SET (CDR PAIR))
                                do (RETURN PAIR)))
             (FCONS-TERM* (CAR TEMP-TEMP)
                    ARG))
            (T (ERROR1 (PQUOTE (PROGN CONVERT-TYPE-NO-TO-RECOGNIZER-TERM CALLED WITH A NUMBER NOT ASSIGNED AS A
                                      TYPE NO !))
                       (BINDINGS)
                       (QUOTE HARD])
```

(BM-COUNT

```
  [LAMBDA (X)                                              (* kbr: "19-Oct-85 16:31")
    (COND
       ((NLISTP X)
        0)
       (T (PLUS 1 (COUNT (CAR X))
                (COUNT (CDR X])
```

(COUNT-IFS

```
  [LAMBDA (TERM)                                           (* kbr: "19-Oct-85 16:31")
    (COND
       ((VARIABLEP TERM)
        0)
       ((FQUOTEP TERM)
        0)
       [(EQ (FFN-SYMB TERM)
            (QUOTE IF))
        (ADD1 (for ARG in (FARGS TERM) sum (COUNT-IFS ARG]
       (T (for ARG in (FARGS TERM) sum (COUNT-IFS ARG])
```

(CREATE-REWRITE-RULE

```
  [LAMBDA (NAME HYPS CONCL LOOP-STOPPER-ARG)               (* kbr: "19-Oct-85 16:31")
    (create REWRITE-RULE
            NAME _ NAME
            HYPS _ (PREPROCESS-HYPS HYPS)
            CONCL _ CONCL
            LOOP-STOPPER _ (OR LOOP-STOPPER-ARG (LOOP-STOPPER CONCL])
```

(DCL0

```
  [LAMBDA (NAME ARGS)                                      (* kbr: "19-Oct-85 16:31")
    [ADD-FACT NAME (QUOTE TYPE-PRESCRIPTION-LST)
              (CONS NAME (CONS TYPE-SET-UNKNOWN (for X in ARGS collect NIL]
    (ADD-FACT NAME (QUOTE LEVEL-NO)
              0])
```

(DECODE-IDATE

```
  [LAMBDA (N)                                              (* kbr: "19-Oct-85 16:31")
    (POWER-REP N 100.0])
```

(DEFN-ASSUME-TRUE-FALSE

```
  [LAMBDA (TERM)                                           (* kbr: "24-Oct-85 15:34")
    (LET (TYPE-ARG1 TYPE-ARG2 TRUE-SEG FALSE-SEG PAIR ARG1 ARG2 INTERSECTION LOCAL-MUST-BE-TRUE
                LOCAL-MUST-BE-FALSE)
         [COND
            [(AND (NVARIABLEP TERM)
```

```
                (NOT (FQUOTEP TERM))
                (SETQ PAIR (ASSOC (FFN-SYMB TERM)
                                  RECOGNIZER-ALIST)))
           (SETQ TYPE-ARG1 (DEFN-TYPE-SET (FARGN TERM 1)))
           (COND
              ([AND (NULL (CDR TYPE-ARG1))
                    (IEQP 0 (LOGAND (CAR TYPE-ARG1)
                                    (CDR PAIR]
               (SETQ LOCAL-MUST-BE-FALSE T))
              ((AND (NULL (CDR TYPE-ARG1))
                    (LOGSUBSETP (CAR TYPE-ARG1)
                                (CDR PAIR)))
               (SETQ LOCAL-MUST-BE-TRUE T))
              (T [SETQ TRUE-SEG (LIST (CONS (FARGN TERM 1)
                                            (CONS (CDR PAIR)
                                                  NIL]
                 (SETQ FALSE-SEG (LIST (CONS (FARGN TERM 1)
                                             (CONS (LOGAND (CAR TYPE-ARG1)
                                                           (LOGNOT (CDR PAIR)))
                                                   (CDR TYPE-ARG1]
          [(BM-MATCH TERM (EQUAL ARG1 ARG2))
           (SETQ TYPE-ARG1 (DEFN-TYPE-SET ARG1))
           (SETQ TYPE-ARG2 (DEFN-TYPE-SET ARG2))
           (SETQ INTERSECTION (LOGAND (CAR TYPE-ARG1)
                                      (CAR TYPE-ARG2)))
           (COND
              ((AND (IEQP 0 INTERSECTION)
                    (NULL (CDR TYPE-ARG1))
                    (NULL (CDR TYPE-ARG2)))
               (SETQ LOCAL-MUST-BE-FALSE T))
              ((AND (NULL (CDR TYPE-ARG1))
                    (NULL (CDR TYPE-ARG2))
                    (IEQP (CAR TYPE-ARG1)
                          (CAR TYPE-ARG2))
                    (MEMBER (CAR TYPE-ARG1)
                            SINGLETON-TYPE-SETS))
               (SETQ LOCAL-MUST-BE-TRUE T))
              ((AND (EQUAL TYPE-ARG1 TYPE-ARG2)
                    (IEQP 0 (CAR TYPE-ARG1))
                    (IEQP (LENGTH (CDR TYPE-ARG1))
                          1))
               (SETQ LOCAL-MUST-BE-TRUE T))
              (T [SETQ TRUE-SEG (LIST (CONS TERM (CONS TYPE-SET-TRUE NIL]
                 [COND
                    ((NOT (IEQP (CAR TYPE-ARG1)
                                INTERSECTION))
                     (SETQ TRUE-SEG (CONS (CONS ARG1 (CONS INTERSECTION (CDR TYPE-ARG1)))
                                          TRUE-SEG]
                 [COND
                    ((NOT (IEQP (CAR TYPE-ARG2)
                                INTERSECTION))
                     (SETQ TRUE-SEG (CONS (CONS ARG2 (CONS INTERSECTION (CDR TYPE-ARG2)))
                                          TRUE-SEG]
                 [SETQ FALSE-SEG (LIST (CONS TERM (CONS TYPE-SET-FALSE NIL]
                 [COND
                    ((AND (MEMBER (CAR TYPE-ARG2)
                                  SINGLETON-TYPE-SETS)
                          (NULL (CDR TYPE-ARG2)))
                     (SETQ FALSE-SEG (CONS (CONS ARG1 (CONS (LOGAND (CAR TYPE-ARG1)
                                                                    (LOGNOT (CAR TYPE-ARG2)))
                                                            (CDR TYPE-ARG1)))
                                           FALSE-SEG]
                 [COND
                    ((AND (MEMBER (CAR TYPE-ARG1)
                                  SINGLETON-TYPE-SETS)
                          (NULL (CDR TYPE-ARG1)))
                     (SETQ FALSE-SEG (CONS (CONS ARG2 (CONS (LOGAND (CAR TYPE-ARG2)
                                                                    (LOGNOT (CAR TYPE-ARG1)))
                                                            (CDR TYPE-ARG2)))
                                           FALSE-SEG]
                 [COND
                    ((AND (IEQP 0 (CAR TYPE-ARG2))
                          (IEQP (LENGTH (CDR TYPE-ARG2))
                                1)
                          (MEMB (CADR TYPE-ARG2)
                                (CDR TYPE-ARG1)))
                     (SETQ FALSE-SEG (CONS [CONS ARG1 (CONS (CAR TYPE-ARG1)
                                                            (REMOVE (CADR TYPE-ARG2)
                                                                    (CDR TYPE-ARG1]
                                           FALSE-SEG]
                 (COND
                    ((AND (IEQP 0 (CAR TYPE-ARG1))
                          (IEQP (LENGTH (CDR TYPE-ARG1))
                                1)
                          (MEMB (CADR TYPE-ARG1)
                                (CDR TYPE-ARG2)))
                     (SETQ FALSE-SEG (CONS [CONS ARG2 (CONS (CAR TYPE-ARG2)
```

```
                                                                (REMOVE (CADR TYPE-ARG1)
                                                                        (CDR TYPE-ARG2]
                                                FALSE-SEG]
            (T (SETQ TYPE-ARG1 (DEFN-TYPE-SET TERM))
               (COND
                  ((AND (IEQP (CAR TYPE-ARG1)
                              TYPE-SET-FALSE)
                        (NULL (CDR TYPE-ARG1)))
                   (SETQ LOCAL-MUST-BE-FALSE T))
                  ((AND (NULL (CDR TYPE-ARG1))
                        (IEQP 0 (LOGAND (CAR TYPE-ARG1)
                                        TYPE-SET-FALSE)))
                   (SETQ LOCAL-MUST-BE-TRUE T))
                  (T [SETQ TRUE-SEG (LIST (CONS TERM (CONS (LOGAND (CAR TYPE-ARG1)
                                                                   (LOGNOT TYPE-SET-FALSE))
                                                           (CDR TYPE-ARG1]
                     (SETQ FALSE-SEG (LIST (CONS TERM (CONS TYPE-SET-FALSE NIL]
          (SETQ TRUE-TYPE-ALIST (NCONC TRUE-SEG TYPE-ALIST))
          (SETQ FALSE-TYPE-ALIST (NCONC FALSE-SEG TYPE-ALIST))
          (SETQ MUST-BE-TRUE LOCAL-MUST-BE-TRUE)
          (SETQ MUST-BE-FALSE LOCAL-MUST-BE-FALSE)
          NIL])
```

(DEFN-LOGIOR

```
  [LAMBDA (X Y)                                                 (* kbr: "19-Oct-85 16:31")
    (CONS (LOGOR (CAR X)
                 (CAR Y))
          (UNIONQ (CDR X)
                  (CDR Y])
```

(DEFN-SETUP

```
  [LAMBDA (EVENT)                                               (* kbr: "19-Oct-85 16:31")
    (SETQ ORIGEVENT EVENT)
    (SETQ LAST-PROCESS (QUOTE SETUP))
    [COND
       ((NOT (MEMBER ORIGEVENT FAILED-THMS))
        (SETQ FAILED-THMS (CONS ORIGEVENT FAILED-THMS]
    (SETQ EXPAND-LST HINTED-EXPANSIONS)
    (SETQ TERMS-TO-BE-IGNORED-BY-REWRITE NIL)
    (SETQ INDUCTION-HYP-TERMS NIL)
    (SETQ INDUCTION-CONCL-TERMS NIL)
    (SETQ STACK NIL)
    (SETQ FNSTACK NIL)
    (SETQ TYPE-ALIST NIL)
    (SETQ LITS-THAT-MAY-BE-ASSUMED-FALSE NIL)
    (SETQ CURRENT-LIT 0)
    (SETQ CURRENT-ATM 0)
    (SETQ ANCESTORS NIL)
    (INIT-LEMMA-STACK)
    (INIT-LINEARIZE-ASSUMPTIONS-STACK)
    (SETQ LAST-PRINEVAL-CHAR NIL)
    (RANDOM-INITIALIZATION ORIGEVENT)
   EVENT])
```

(DEFN-TYPE-SET

```
  [LAMBDA (TERM)                                                (* kbr: "24-Oct-85 15:44")
    (COND
       ((SETQ TEMP-TEMP (SASSOC TERM TYPE-ALIST))
        (CDR TEMP-TEMP))
       ((VARIABLEP TERM)
        (ERROR1 (PQUOTE (PROGN DEFN-TYPE-SET HAS FOUND AN UNBOUND VARIABLE IN THE TERM (!PPR TERM NIL)
                               %.))
                (BINDINGS (QUOTE TERM)
                          TERM)
                (QUOTE HARD)))
       [(EQ (FN-SYMB TERM)
            (QUOTE IF))
        (DEFN-ASSUME-TRUE-FALSE (FARGN TERM 1))
        (COND
           (MUST-BE-TRUE (DEFN-TYPE-SET (FARGN TERM 2)))
           (MUST-BE-FALSE (DEFN-TYPE-SET (FARGN TERM 3)))
           (T (DEFN-LOGIOR (DEFN-TYPE-SET2 (FARGN TERM 2)
                                           TRUE-TYPE-ALIST)
                           (DEFN-TYPE-SET2 (FARGN TERM 3)
                                           FALSE-TYPE-ALIST]
       [(SETQ TEMP-TEMP (TYPE-PRESCRIPTION (FN-SYMB TERM)))
        (DEFN-LOGIOR (CONS (CAR TEMP-TEMP)
                           NIL)
                (COND
                   ((CDR TEMP-TEMP)
                    (for ARG in (SARGS TERM) as FLG in (CDR TEMP-TEMP) bind ANS first (SETQ ANS (CONS 0 NIL))
                       when FLG do (SETQ ANS (DEFN-LOGIOR (DEFN-TYPE-SET ARG)
                                                          ANS))
                       finally (RETURN ANS)))
```

```
                (T (CONS 0 NIL]
       (T (CONS TYPE-SET-UNKNOWN NIL])
```

(DEFN-TYPE-SET2

```
  [LAMBDA (TERM TYPE-ALIST)                                    (* kbr: "19-Oct-85 16:31")
    (LET (FALSE-TYPE-ALIST)
         (DEFN-TYPE-SET TERM])
```

(DEFN-WRAPUP

```
  [LAMBDA (WON-FLG)                                            (* kbr: "19-Oct-85 16:31")
    (SETQ WON-FLG (COND
                     (WON-FLG (QUOTE DEFN-OK))
                     (T NIL)))
    [COND
       ((NEQ LEMMA-STACK ORIG-LEMMA-STACK)
        (ITERPRI T)
        (ERROR1 (PQUOTE (PROGN DEFN-WRAPUP FOUND A NON-TRIVIAL LEMMA-STACK !))
               (BINDINGS)
               (QUOTE WARNING]
    [COND
       ((NEQ LINEARIZE-ASSUMPTIONS-STACK ORIG-LINEARIZE-ASSUMPTIONS-STACK)
        (ITERPRI T)
        (ERROR1 (PQUOTE (PROGN DEFN-WRAPUP FOUND A NON-TRIVIAL LINEARIZE-ASSUMPTIONS-STACK !))
               (BINDINGS)
               (QUOTE WARNING]
    [COND
       (WON-FLG (SETQ FAILED-THMS (REMOVE ORIGEVENT FAILED-THMS))
              (SETQ PROVED-THMS (CONS ORIGEVENT PROVED-THMS]
    (IO (QUOTE FINISHED)
        NIL NIL NIL (LIST WON-FLG])
```

(DEFN0

```
  [LAMBDA (NAME ARGS BODY RELATION-MEASURE-LST FLG)            (* kbr: " 4-Jul-86 18:16")
    (LET (TRANSLATED-BODY CONTROL-VARS (ARITY-ALIST (CONS (CONS NAME (LENGTH ARGS))
                                                          ARITY-ALIST))
                (META-NAMES (CONS NAME META-NAMES)))
```

(* The list of comments on this function do not necessarily describe the code below.
They have been left around in reverse chronology order to remind us of the various combinations of preprocessing we have tried. If we ever get blown out of the water while normalizing IFs in a large defn, read the following comment before abandoning normalization. 18 August 1982.0 Here we go again! At the time of this writing the preprocessing of defns is as follows, we compute the induction and type info on the translated body and store under sdefn the translated body.
This seems to slow down the system a lot and we are going to change it so that we store under sdefn the result of expanding boot strap nonrec fns and normalizing IFs. As nearly as we can tell from the comments below, we have not previously tried this. According to the record, we have tried expanding all nonrec fns, and we have tried expanding boot strap fns and doing a little normalization. The data that suggests this will speed things up is as follows.
Consider the first call of SIMPLIFY-CLAUSE in the proof of PRIME-LIST-TIMES-LIST.
The first three literals are trivial but the fourth call of SIMPLIFY-CLAUSE1 is on
(NOT (PRIME1 C (SUB1 C)))%. With SDEFNs not expanded and normalized --
i.e., under the processing as it was immediately before the current change --
there are 2478 calls of REWRITE and 273 calls of RELIEVE-HYPS for this literal.
With all defns preprocessed as described here those counts drop to 1218 and 174.0 On a sample of four theorems, PRIME-LIST-TIMES-LIST, PRIME-LIST-PRIME-FACTORS, FALSIFY1-FALSIFIES, and ORDERED-SORT, the use of normalized and expanded sdefns saves us 16\ of the conses over the use of untouched sdefns, reducing the cons counts for those theorems from 880K to 745K. It seems unlikely that this preprocessing will blow us out of the water on large defns.
For the EV used in UNSOLV and for the 386L M with subroutine call this new preprocessing only marginally increases the size of the sdefn. It would be interesting to see a function that blows us out of the water.
When one is found perhaps the right thing to do is to so preprocess small defns and leave big ones alone.
17 December 1981.0 Henceforth we will assume that the very body the user supplies
(modulo translation) is the body that the theorem-prover uses to establish that there is one and only one function satisfying the definition equation by determining that the given body provides a method for computing just that function.
This prohibits our such as (f ITIMES) IEQP (if (f ITIMES) a a) to
(f ITIMES) IEQP a. 18 November 1981.0 We are sick of having to disable nonrec fns in order to get large fns processed, e.g., the interpreter for our 386L class. Thus, we have decided to adopt the policy of not touching the user's typein except to TRANSLATE it. The induction and type analysis as well as the final SDEFN are based on the translated typein.
Before settling with the preprocessing used below we tried several different combinations and did provealls.
The main issue was whether we should normalize sdefns. Unfortunately, the incorporation of META0-LEMMAS was also being experimented with, and so we do not have a precise breakdown of who is responsible for what.
However, below we give the total stats for three separate provealls.
The first, called 1PROVEALL, contained exactly the code below --
except that the ADD-DCELL was given the SDEFN with all the fn names replaced by 1fns instead of a fancy TRANSLATE-TO-INTERLISP call. Here are the 1PROVEALL stats.
Elapsed time IEQP 9532.957, CPU time IEQP 4513.88, GC time IEQP 1423.261, IO time IEQP 499.894, CONSes consumed IEQP 6331517.0 We then incorporated META0-LEMMAS.
Simultaneously, we tried running the RUN fns through DEFN and found that we exploded.
The expansion of nonrec fns and the normalization of IFs before the induction analysis transformed functions of COUNT 300 to functions of COUNT exceeding 18K. We therefore decided to expand only BOOT-STRAP fns --
and not NORMALIZE-IFS for the purposes of induction analysis.
After the induction and type analyses were done, we put down an SDEFN with some trivial IF simplification performed --
e.g., IF X Y Y => Y and IF bool T F => bool -- but not a NORMALIZE-IFs version.
We then ran a proveall with CANCEL around as a META0-LEMMA.
The result was about 20\ slower than the 1PROVEALL and used 15\ more CONSes.
At first this was attributed to CANCEL. However, we then ran two simultaneous provealls, one with META0-LEMMAS set to NIL and one with it set to ((1CANCEL))%. The result was that the version with CANCEL available used slightly fewer CONSes than the other one -- 7303311 to 7312505 That was surprising because the implementation of META0-LEMMAS

uses no CONSes if no META0-LEMMAS are available, so the entire 15\ more CONSes had to be attributed to the difference in the defn processing. This simultaneous run was interesting for two other reasons.
The times -- while still 20\ worse than 1PROVEALL -- were one half of one percent different, with CANCEL being the slower. That means having CANCEL around does not cost much at all --
and the figures are significant despite the slop in the operating system's timing due to thrashing because the two jobs really were running simultaneously. The second interesting fact is that CANCEL can be expected to save us a few CONSes rather than cost us. We therefore decided to return the DEFN0 processing to its original state.
Only we did it in two steps. First, we put NORMALIZE-IFs into the pre-induction processing and into the final SDEFN processing. Here are the stats on the resulting proveall, which was called PROVEALL-WITH-NORM-AND-CANCEL but not saved. Elapsed time IEQP 14594.01, CPU time IEQP 5024.387, GC time IEQP 1519.932, IO time IEQP 593.625, CONSes consumed IEQP 6762620.0 While an improvement, we were still 6\ worse than 1PROVEALL on CONSes.
But the only difference between 1PROVEALL and PROVEALL-WITH-NORM-AND-CANCEL --
if you discount CANCEL which we rightly believed was paying for itself --
was that in the former induction analyses and type prescriptions were being computed from fully expanded bodies while in the latter they were computed from only BOOT-STRAP-expanded bodies.
We did not believe that would make a difference of over 400,000 CONSes, but had nothing else to believe.
So we went to the current state, where we do the induction and type analyses on the fully expanded and normalized bodies -- bodies that blow us out of the water on some of the RUN fns.
Here are the stats for PROVEALL-PROOFS.79101, which was the proveall for that version.
Elapsed time IEQP 21589.84, CPU time IEQP 4870.231, GC time IEQP 1512.813, IO time IEQP 554.292, CONSes consumed= 6356282.0 Note that we are within 25K of the number of CONSes used by 1PROVEALL.
But to TRANSLATE-TO-INTERLISP all of the defns in question costs 45K.
So -- as expected -- CANCEL actually saved us a few CONSes by shortening proofs.
It takes only 18 seconds to TRANSLATE-TO-INTERLISP the defns, so a similar argument does not explain why the latter proveall is 360 seconds slower than 1PROVEALL. But since the elapsed time is over twice as long, we believe it is fair to chalk that time up to the usual slop involved in measuring cpu time on a time sharing system.
We now explain the formal justification of the processing we do on the body before testing it for admissibility.
We do not work with the body that is typed in by the user but with an equivalent body' produced by normalization and the expansion of nonrecursive function calls in body. We now prove that if
(under no assumptions about NAME except that it is a function symbol of the correct arity)
(a) body is equivalent to body' and (b) (name) IEQP body' is accepted under our principle of definition, then there exists exactly one function satisfying the original equation (name) IEQP body.
First observe that since the definition (name) IEQP body' is accepted by our principle of definition, there exists a function satisfying that equation. But the accepted equation is equivalent to the equation
(name) IEQP body by the hypothesis that body is equivalent to body'.
We prove that there is only one such function by induction. Assume that the definition
(name) IEQP body has been accepted under the principle of definition.
Suppose that f is a new name and that (f) IEQP bodyf, where bodyf results from replacing every use of name as a function symbol in body with f. It follows that (f) IEQP bodyf', where bodyf' results from replacing every use of name as a function symbol in body' with f. We can now easily prove that (f) IEQP (name) by induction according to the definition of name.
Q.E.D. One might be tempted to think that if the defn with body' is accepted under the principle of definition then so would be the defn with body and that the use of body' was merely to make the implementation of the defn principle more powerful. This is not the case. For example (R X) IEQP (IF (R X) T T) is not accepted by the definitional principle, but we would accept the body'-version (R X) IEQP T, and by our proof, that function uniquely satisfies the equation the user typed in.
One might be further tempted to think that if we changed normalize so that
(IF X Y Y) IEQP Y was not applied, then the two versions were inter-acceptable under the defn principle.
This is not the case either. The function (F X) IEQP (IF (IF (X.ne.0)
(F X-1) F) (F X-1) T) is not accepted under the principle of defn.
Consider its normalized body. *)

```
(DEFN-SETUP (LIST (QUOTE DEFN)
                  NAME ARGS BODY RELATION-MEASURE-LST))
(SETQ TRANSLATED-BODY (TRANSLATE BODY))
[SETQ RELATION-MEASURE-LST (for TEMP in RELATION-MEASURE-LST collect (LIST (CAR TEMP)
                                                                          (TRANSLATE (CADR TEMP]
(PUT-INDUCTION-INFO NAME ARGS TRANSLATED-BODY RELATION-MEASURE-LST NIL)
(ADD-FACT NAME (QUOTE SDEFN)
       (LIST (QUOTE LAMBDA)
             ARGS
              (NORMALIZE-IFS (EXPAND-BOOT-STRAP-NON-REC-FNS TRANSLATED-BODY)
                    NIL NIL)))
(PUT-TYPE-PRESCRIPTION NAME)
(PUT-LEVEL-NO NAME)

 (* CONTROLLER-POCKETS of NAME is a list of bit encodings.
 Each bit encoding summarizes a SUBSET of some JUSTIFICATION for NAME to terminate.
 *)

[AND (GETPROP NAME (QUOTE JUSTIFICATIONS))
     (ADD-FACT NAME (QUOTE CONTROLLER-POCKETS)
            (SCRUNCH (for TEMP in (GETPROP NAME (QUOTE JUSTIFICATIONS))
                        collect (PROGN (SETQ CONTROL-VARS (fetch (JUSTIFICATION SUBSET) of TEMP))
                                       (for FORMAL in ARGS as I from 0 bind (LOOP-ANS _ 0)
                                          when (MEMB FORMAL CONTROL-VARS)
                                          do (SETQ LOOP-ANS (LOGOR LOOP-ANS (LSH 1 I)))
                                          finally (RETURN LOOP-ANS]
[COND
   [FLG (ADD-FACT NAME (QUOTE LISP-CODE)
               (PACK (LIST STRING-WEIRD NAME]
   ([for FN in (ALL-FNNAMES TRANSLATED-BODY) always (OR (EQ FN NAME)
                                                        (GETPROP FN (QUOTE LISP-CODE]
    (ADD-DCELL NAME (PACK (LIST STRING-WEIRD NAME))
           (LIST (QUOTE LAMBDA)
                 [SETQ TEMP-TEMP (for ARG in ARGS collect (PACK (LIST STRING-WEIRD3 ARG]
                 (TRANSLATE-TO-LISP (SUB-PAIR-VAR ARGS TEMP-TEMP TRANSLATED-BODY]
[COND
   ((NOT (TOTAL-FUNCTIONP NAME))
```

```
            (ERROR1 (PQUOTE (PROGN THE RECURSION IN (!PPR NAME NIL)
                                   IS UNJUSTIFIED %.))
                    (BINDINGS (QUOTE NAME)
                              NAME)
                    (QUOTE WARNING]
        NIL])
```

(DELETE1

```
[LAMBDA (X L)                                                    (* kbr: "19-Oct-85 16:31")
  (COND
     ((NLISTP L)
      NIL)
     ((EQUAL X (CAR L))
      (CDR L))
     (T (CONS (CAR L)
              (DELETE1 X (CDR L])
```

(DELETE-TAUTOLOGIES

```
[LAMBDA (CLAUSE-SET)                                             (* kbr: "19-Oct-85 16:31")
  (for CL in CLAUSE-SET unless [for TAIL on CL thereis (OR (AND (FALSE-NONFALSEP (CAR TAIL))
                                                               (NOT DEFINITELY-FALSE))
                                                          (MEMBER (NEGATE-LIT (CAR TAIL))
                                                                  (CDR TAIL]
     collect CL])
```

(DELETE-TOGGLES

```
[LAMBDA (XXX)                                                    (* kbr: "19-Oct-85 16:31")
  (for X in XXX bind N collect (COND
                                  ((BM-MATCH X (TOGGLE & N (QUOTE T)))
                                   (LIST (QUOTE DISABLE)
                                         N))
                                  ((OR (BM-MATCH X (TOGGLE & N (QUOTE NIL)))
                                       (BM-MATCH X (TOGGLE & N)))
                                   (LIST (QUOTE ENABLE)
                                         N))
                                  (T X])
```

(DEPEND

```
[LAMBDA (DEPENDENT SUPPORTERS)                                   (* kbr: "19-Oct-85 16:31")
  [COND
     ((NOT (GETPROP DEPENDENT (QUOTE EVENT)))
      (ERROR1 (PQUOTE (PROGN DEPEND SHOULD NOT BE CALLED ON A NONEVENT SUCH AS (!PPR DEPENDENT NIL)
                             %.))
              (BINDINGS (QUOTE DEPENDENT)
                        DEPENDENT)
              (QUOTE HARD]
  [SETQ SUPPORTERS (REMOVE (QUOTE GROUND-ZERO)
                           (for X in SUPPORTERS bind LOOP-ANS do (SETQ LOOP-ANS (ADD-TO-SET (MAIN-EVENT-OF X)
                                                                                            LOOP-ANS))
                              finally (RETURN LOOP-ANS]
  [COND
     ((MEMB DEPENDENT SUPPORTERS)
      (ERROR1 (PQUOTE (PROGN ATTEMPT TO MAKE (!PPR DEPENDENT NIL)
                             DEPEND UPON ITSELF !))
              (BINDINGS (QUOTE DEPENDENT)
                        DEPENDENT)
              (QUOTE HARD]
  (for X in SUPPORTERS do (ADD-FACT X (QUOTE IMMEDIATE-DEPENDENTS0)
                                 DEPENDENT])
```

(DEPENDENT-EVENTS

```
[LAMBDA (EVENT)                                                  (* kbr: "19-Oct-85 16:31")
  (for X in (DEPENDENTS-OF EVENT) collect (GETPROP X (QUOTE EVENT])
```

(DEPENDENTS-OF

```
[LAMBDA (NAME)                                                   (* kbr: "19-Oct-85 16:31")
  (COND
     ((EQ NAME (QUOTE GROUND-ZERO))
      (REVERSE CHRONOLOGY))
     ((NOT (GETPROP NAME (QUOTE EVENT)))
      (ERROR1 (PQUOTE (PROGN DEPENDENTS-OF MUST BE GIVEN AN EVENT AND (!PPR NAME NIL)
                                                                      IS NOT ONE %.))
              (BINDINGS (QUOTE NAME)
                        NAME)
              (QUOTE HARD)))
     (T (SORT (DEPENDENTS-OF1 NAME)
              (FUNCTION (LAMBDA (X Y)
                          (EVENT1-OCCURRED-BEFORE-EVENT2 X Y CHRONOLOGY])
```

(DEPENDENTS-OF1

```
  [LAMBDA (NAME)                                                  (* kbr: "19-Oct-85 19:59")
    (COND
       ((EQ NAME (QUOTE GROUND-ZERO))

          (* We never expect this fn to be called on GROUND-ZERO because its silly, but we make it behave correctly anyway.
          *)

        (COPYLIST CHRONOLOGY))
       (T (CONS NAME (SCRUNCH (for X in (IMMEDIATE-DEPENDENTS-OF NAME) join (DEPENDENTS-OF1 X])
```

(DESTRUCTORS

```
  [LAMBDA (CL)                                                    (* kbr: "19-Oct-85 16:31")

          (* This function returns the set of subterms of CL such that every member is the application of a function to one or more
          distinct variables. *)

    (LET (ANS)
         (for LIT in CL do (DESTRUCTORS1 LIT))
        ANS])
```

(DESTRUCTORS1

```
  [LAMBDA (TERM)                                                  (* kbr: "19-Oct-85 16:31")
    (COND
       ((OR (VARIABLEP TERM)
            (FQUOTEP TERM))
        NIL)
       (T (for ARG in (FARGS TERM) do (DESTRUCTORS1 ARG))
          (COND
             ((AND (FARGS TERM)
                   (for ARG in (FARGS TERM) always (VARIABLEP ARG))
                   (NO-DUPLICATESP (FARGS TERM)))
              (SETQ ANS (ADD-TO-SET TERM ANS])
```

(DETACH

```
  [LAMBDA NIL                                                     (* kbr: "19-Oct-85 16:31")
    (ERROR1 (PQUOTE (PROGN DETACH IS NOT YET IMPLEMENTED))
           NIL
           (QUOTE HARD])
```

(DETACHED-ERROR

```
  [LAMBDA NIL                                                     (* kbr: "19-Oct-85 16:31")
    (CLOSE? TTY-FILE)
    (CLOSE? PROVE-FILE)
    (SETQ PROVE-FILE NIL)
    (SETQ TTY-FILE NIL)
    (CL:BREAK (QUOTE DETACHED-ERROR])
```

(DETACHEDP

```
  [LAMBDA NIL                                                     (* kbr: "19-Oct-85 16:31")
    NIL])
```

(DISJOIN

```
  [LAMBDA (LST IF-FLG)                                            (* kbr: "19-Oct-85 16:31")
    (COND
       ((NULL LST)
        FALSE)
       (T (DISJOIN2 (CAR LST)
                    (DISJOIN (CDR LST)
                             IF-FLG)
                    IF-FLG])
```

(DISJOIN-CLAUSES

```
  [LAMBDA (CL1 CL2)                                               (* kbr: "20-Oct-85 16:25")
                                                                  (* The OR of clause CL1 and clause CL2.
                                                                  *)
    (COND
       ((OR (EQUAL CL1 TRUE-CLAUSE)
            (EQUAL CL2 TRUE-CLAUSE))
        TRUE-CLAUSE)
       ((for LIT1 in CL1 thereis (for LIT2 in CL2 thereis (COMPLEMENTARYP LIT1 LIT2)))
        TRUE-CLAUSE)
       (T (APPEND CL1 (SET-DIFF CL2 CL1])
```

(DISJOIN2

```
  [LAMBDA (P Q IF-FLG)                                            (* kbr: "19-Oct-85 16:31")
    (COND
       ((FALSE-NONFALSEP P)
        (COND
           (DEFINITELY-FALSE (COND
```

```
                                ((FALSE-NONFALSEP Q)
                                 (COND
                                    (DEFINITELY-FALSE FALSE)
                                    (T TRUE)))
                                ((NOT (BOOLEAN Q))
                                 (FCONS-TERM* (QUOTE IF)
                                        Q TRUE FALSE))
                                (T Q)))
            (T TRUE)))
       ((FALSE-NONFALSEP Q)
        (COND
           [DEFINITELY-FALSE (COND
                                ((BOOLEAN P)
                                 P)
                                (T (FCONS-TERM* (QUOTE IF)
                                          P TRUE FALSE]
            (T TRUE)))
       [IF-FLG (FCONS-TERM* (QUOTE IF)
                       P TRUE (COND
                                ((BOOLEAN Q)
                                 Q)
                                (T (FCONS-TERM* (QUOTE IF)
                                          Q TRUE FALSE]
       (T (FCONS-TERM* (QUOTE OR)
                 P Q])
```

(DTACK-0-ON-END

```
[LAMBDA (X)                                                    (* kbr: "19-Oct-85 16:31")
  (RPLACD (LAST X)
          0)
  X])
```

(DUMB-CONVERT-TYPE-SET-TO-TYPE-RESTRICTION-TERM

```
[LAMBDA (TYPE-SET ARG)                                         (* kbr: "19-Oct-85 16:31")

        (* WARNING: This function does not return a legal term. In particular, it might return
        (AND a b c ...)%. It should be used only for io purposes. *)

  (LET (LST)
       (COND
          ((IEQP TYPE-SET TYPE-SET-UNKNOWN)
           TRUE)
          ((IEQP TYPE-SET 0)
           FALSE)
          [(IEQP 0 (LSH TYPE-SET -31))
           (SETQ LST (for I from 0 to 30 when (NOT (IEQP (LOGAND TYPE-SET (LOGBIT I))
                                                         0))
                        collect (CONVERT-TYPE-NO-TO-RECOGNIZER-TERM I ARG)))
           (COND
              ((NULL LST)
               FALSE)
              ((NULL (CDR LST))
               (CAR LST))
              (T (CONS (QUOTE OR)
                       LST]
          (T [SETQ LST (for I from 0 to 30 when (IEQP 0 (LOGAND TYPE-SET (LOGBIT I)))
                          collect (DUMB-NEGATE-LIT (CONVERT-TYPE-NO-TO-RECOGNIZER-TERM I ARG]
             (COND
                ((NULL LST)
                 TRUE)
                ((NULL (CDR LST))
                 (CAR LST))
                (T (CONS (QUOTE AND)
                         LST])
```

(DUMB-IMPLICATE-LITS

```
[LAMBDA (L1 L2)                                                (* kbr: "19-Oct-85 16:31")

        (* Like DUMB-NEGATE-LIT, this function may be called when TYPE-ALIST is not valid.
        Hence this function should not be modified to use TYPE-SET. *)

  (COND
     ((QUOTEP L1)
      (COND
         ((EQUAL L1 FALSE)
          TRUE)
         (T L2)))
     (T (FCONS-TERM* (QUOTE IF)
               L1 L2 TRUE])
```

(DUMB-NEGATE-LIT

```
[LAMBDA (TERM)                                                 (* kbr: "20-Oct-85 16:27")
```

(* Syntactic NOT of TERM. Like DUMB-IMPLICATE-LITS, this function may be called when TYPE-ALIST is not valid. Hence this function should not be modified to use TYPE-SET. *)

```
    (COND
       ((VARIABLEP TERM)
        (FCONS-TERM* (QUOTE NOT)
               TERM))
       ((FQUOTEP TERM)
        (COND
           ((EQUAL TERM FALSE)
            TRUE)
           (T FALSE)))
       ((EQ (FN-SYMB TERM)
            (QUOTE NOT))
        (FARGN TERM 1))
       (T (FCONS-TERM* (QUOTE NOT)
                 TERM])
```

(DUMB-OCCUR

```
  [LAMBDA (X Y)                                          (* kbr: " 4-Jul-86 17:01")
                                                         (* Does X syntactically occur in Y? *)
    (COND
       ((EQUAL X Y)
        T)
       ((VARIABLEP Y)
        NIL)
       ((FQUOTEP Y)
        NIL)
       (T (for ARG in (FARGS Y) thereis (DUMB-OCCUR X ARG])
```

(DUMB-OCCUR-LST

```
  [LAMBDA (X LST)                                        (* kbr: "19-Oct-85 16:31")
    (for TERM in LST thereis (DUMB-OCCUR X TERM])
```

(DUMP

```
  [LAMBDA (LST FILE INDENT WIDTH INDEX-FLG SCRIBE-FLG)           (* kbr: "20-Oct-85 19:39")
    (LET (PAIRS)
         (OR INDENT (SETQ INDENT 5))
         (OR WIDTH (SETQ WIDTH 68))
         (SETQ FILE (OPENSTREAM FILE (QUOTE OUTPUT)))
         (LINEL FILE WIDTH)
         [SETQ PAIRS (for L in LST as I from 1 collect (PROGN [COND
                                                                 ((LITATOM L)
                                                                  (SETQ L (GETPROP L (QUOTE EVENT]
                                                               (SELECTQ (CAR L)
                                                                    (DEFN (DUMP-DEFN (BM-NTH 1 L)
                                                                                     (BM-NTH 2 L)
                                                                                     (BM-NTH 3 L)
                                                                                     (BM-NTH 4 L)
                                                                                     (AND INDEX-FLG I)))
                                                                    (PROVE-LEMMA (DUMP-PROVE-LEMMA (BM-NTH 1 L)
                                                                                                   (BM-NTH 2 L)
                                                                                                   (BM-NTH 3 L)
                                                                                                   (BM-NTH 4 L)
                                                                                                   (AND INDEX-FLG I)))
                                                                    (ADD-AXIOM (DUMP-ADD-AXIOM (BM-NTH 1 L)
                                                                                               (BM-NTH 2 L)
                                                                                               (BM-NTH 3 L)
                                                                                               (AND INDEX-FLG I)))
                                                                    (ADD-SHELL (DUMP-ADD-SHELL (BM-NTH 1 L)
                                                                                               (BM-NTH 2 L)
                                                                                               (BM-NTH 3 L)
                                                                                               (BM-NTH 4 L)
                                                                                               (AND INDEX-FLG I)))
                                                                    (DCL (DUMP-DCL (BM-NTH 1 L)
                                                                                   (BM-NTH 2 L)
                                                                                   (AND INDEX-FLG I)))
                                                                    (TOGGLE (DUMP-TOGGLE (BM-NTH 1 L)
                                                                                         (BM-NTH 2 L)
                                                                                         (BM-NTH 3 L)
                                                                                         (AND INDEX-FLG I)))
                                                                    (DISABLE (DUMP-TOGGLE NIL (BM-NTH 1 L)
                                                                                          NIL
                                                                                          (AND INDEX-FLG I)))
                                                                    (ENABLE (DUMP-TOGGLE NIL (BM-NTH 1 L)
                                                                                         T
                                                                                         (AND INDEX-FLG I)))
                                                                    (DUMP-OTHER L (AND INDEX-FLG I)))
                                                               (CONS (BM-NTH 1 L)
                                                                     I]
         NIL])
```

(DUMP-ADD-AXIOM

```
[LAMBDA (NAME TYPES THM INDEX)                                    (* kbr: "19-Oct-85 16:31")
  (DUMP-BEGIN-GROUP FILE)
  (COND
     (INDEX (IPRINC INDEX FILE)
            (IPRINC "." FILE)
            (ISPACES (IDIFFERENCE INDENT (IPOSITION FILE NIL NIL))
                   FILE))
     (T (ISPACES INDENT FILE)))
  (IPRINC "AXIOM." FILE)
  (IPRINC NAME FILE)
  (COND
     (TYPES (SPACES 1 FILE)
            (DUMP-LEMMA-TYPES TYPES)))
  (IPRINT (QUOTE :)
        FILE)
  (SPACES INDENT FILE)
  (PPRINDENT THM INDENT 0 FILE)
  (ITERPRI FILE)
  (DUMP-END-GROUP FILE])
```

(DUMP-ADD-SHELL

```
[LAMBDA (CONSTRUCTOR BTM RECOG ACCESSORS INDEX)                   (* kbr: "19-Oct-85 16:31")
  (DUMP-BEGIN-GROUP FILE)
  (COND
     (INDEX (IPRINC INDEX FILE)
            (IPRINC "." FILE)
            (ISPACES (IDIFFERENCE INDENT (IPOSITION FILE NIL NIL))
                   FILE))
     (T (ISPACES INDENT FILE)))
  (PRINEVAL (PQUOTE (PROGN SHELL DEFINITION %. // ADD THE SHELL (!PPR CONSTRUCTOR NIL)
                           OF
                           (@ N)
                           (PLURAL? ACCESSORS ARGUMENTS ARGUMENT)
                           WITH // (COND
                                      (BTM BOTTOM OBJECT (!PPR BTM (PQUOTE ,)
                                                                   NIL)
                                           //))
                           RECOGNIZER
                           (!PPR RECOG NIL)
                           , // (PLURAL? ACCESSORS ACCESSORS ACCESSOR)
                           (!PPR-LIST NAMES)
                           , // (COND
                                    (FLG TYPE (PLURAL? ACCESSORS RESTRICTIONS RESTRICTION)
                                         (!PPR-LIST RESTRICTIONS)
                                         , //))
                             AND DEFAULT (PLURAL? ACCESSORS VALUES VALUE)
                                 (!PPR-LIST DEFAULTS NIL)
                                 %.))
        (BINDINGS (QUOTE RECOG)
                  RECOG
                  (QUOTE BTM)
                  BTM
                  (QUOTE ACCESSORS)
                  ACCESSORS
                  (QUOTE CONSTRUCTOR)
                  CONSTRUCTOR
                  (QUOTE N)
                  (LENGTH ACCESSORS)
                  (QUOTE NAMES)
                  (for X in ACCESSORS collect (CAR X))
                  (QUOTE FLG)
                  [for X in ACCESSORS thereis (AND (NEQ (CADR X)
                                                        T)
                                                   (NOT (EQUAL (CADR X)
                                                               TRUE]
                  (QUOTE RESTRICTIONS)
                  (for X in ACCESSORS collect (CADR X))
                  (QUOTE DEFAULTS)
                  (for X in ACCESSORS collect (CADDR X)))
        INDENT FILE)
  (ITERPRI FILE)
  (DUMP-END-GROUP FILE])
```

(DUMP-BEGIN-GROUP

```
[LAMBDA (FILE)                                                    (* kbr: "19-Oct-85 16:31")
  (COND
     (SCRIBE-FLG (PRIN1 (QUOTE @BEGIN (GROUP))
                        FILE)
            (ITERPRI FILE)
            (PRIN1 (QUOTE @BEGIN (VERBATIM))
                   FILE)
            (ITERPRI FILE])
```

(DUMP-DCL

```
  [LAMBDA (FN ARGS INDEX)                                          (* kbr: "19-Oct-85 16:31")
    (DUMP-BEGIN-GROUP FILE)
    (COND
       (INDEX (IPRINC INDEX FILE)
              (IPRINC "." FILE)
              (ISPACES (IDIFFERENCE INDENT (IPOSITION FILE NIL NIL))
                       FILE))
       (T (ISPACES INDENT FILE)))
    (IPRINC "UNDEFINED FUNCTION." FILE)
    (ITERPRI FILE)
    (ISPACES INDENT FILE)
    (IPRINT (CONS FN ARGS)
           FILE)
    (DUMP-END-GROUP FILE])
```

(DUMP-DEFN

```
  [LAMBDA (FN ARGS BODY HINT INDEX)                                (* kbr: "19-Oct-85 16:31")
    (DUMP-BEGIN-GROUP FILE)
    (COND
       (INDEX (IPRINC INDEX FILE)
              (IPRINC "." FILE)
              (ISPACES (IDIFFERENCE INDENT (IPOSITION FILE NIL NIL))
                       FILE))
       (T (ISPACES INDENT FILE)))
    (IPRINC "DEFINITION." FILE)
    (ITERPRI FILE)
    (ISPACES INDENT FILE)
    (IPRINT (CONS FN ARGS)
           FILE)
    (ISPACES (IPLUS INDENT 3)
           FILE)
    (IPRINT (QUOTE IEQP)
           FILE)
    (ISPACES INDENT FILE)
    (PPRINDENT BODY INDENT 0 FILE)
    (ITERPRI FILE)
    [COND
       (HINT (ISPACES INDENT FILE)
             (IPRINC "HINT:" FILE)
             (COND
                ((NULL (CDR HINT))
                 (IPRINC "CONSIDER THE WELL-FOUNDED RELATION" FILE)
                 (IPRINT (CAR (CAR HINT))
                        FILE)
                 (ISPACES (IPLUS 7 INDENT)
                        FILE)
                 (IPRINC "AND THE MEASURE" FILE)
                 (IPRINT (CADR (CAR HINT))
                        FILE))
                (T (PPRINDENT HINT (IPLUS 7 INDENT)
                              0 FILE)
                   (ITERPRI FILE]
    (DUMP-END-GROUP FILE])
```

(DUMP-END-GROUP

```
  [LAMBDA (FILE)                                                   (* kbr: "19-Oct-85 16:31")
    (COND
       (SCRIBE-FLG (PRIN1 (QUOTE @END (VERBATIM))
                          FILE)
              (ITERPRI FILE)
              (PRIN1 (QUOTE @END (GROUP))
                     FILE)
              (ITERPRI FILE)
              (ITERPRI FILE])
```

(DUMP-HINTS

```
  [LAMBDA (HINT)                                                   (* kbr: "19-Oct-85 20:06")
    (LET (USED DISABLED ENABLED (INDENT INDENT))
         (SETQ USED (CDR (ASSOC (QUOTE USE)
                                HINT)))
         (SETQ DISABLED (CDR (ASSOC (QUOTE DISABLE)
                                    HINT)))
         (SETQ ENABLED (for X in USED when (NOT (MEMB (CAR X)
                                                      DISABLED))
                          collect (CAR X)))
         (SETQ DISABLED (for X in DISABLED when (NOT (ASSOC X USED)) collect X))
         [SETQ HINT (for X in HINT join (SELECTQ (CAR X)
                                                 (USE [COND
                                                         ((NULL ENABLED)
                                                          (LIST (CONS (QUOTE USE)
                                                                      USED)))
                                                         (T (LIST (CONS (QUOTE USE)
                                                                        USED)
                                                                  (CONS (QUOTE ENABLE)
```

```
                                                        ENABLED])
                                        (DISABLE [COND
                                                    ((NULL DISABLED)
                                                     NIL)
                                                    (T (LIST (CONS (QUOTE DISABLE)
                                                                   DISABLED])
                                        (LIST X]
        (ISPACES INDENT FILE)
        [COND
           ([OR (LISTP (CDR HINT))
                (AND USED (LISTP (CDR USED]
            (IPRINC "HINTS:" FILE)
            (SETQ INDENT (IPLUS INDENT 8)))
           (T (IPRINC "HINT:" FILE)
              (SETQ INDENT (IPLUS INDENT 7]
        (for X in HINT do (ISPACES (IDIFFERENCE INDENT (IPOSITION FILE NIL NIL))
                                   FILE)
                          (SELECTQ (CAR X)
                               (INDUCT (IPRINC "INDUCT AS FOR" FILE)
                                       (IPRINC (CADR X)
                                              FILE)
                                       (IPRINC "." FILE)
                                       (ITERPRI FILE))
                               (USE (IPRINC "CONSIDER:" FILE)
                                    (ITERPRI FILE)
                                    (for PAIR in (CDR X) do (ISPACES (ADD1 INDENT)
                                                                     FILE)
                                                            (IPRINC (CAR PAIR)
                                                                    FILE)
                                                            (COND
                                                               ((CDR PAIR)
                                                                (IPRINC "WITH {" FILE)
                                                                [for TL on (CDR PAIR)
                                                                   do (IPRINC (CAAR TL)
                                                                              FILE)
                                                                      (IPRINC "/" FILE)
                                                                      (IPRINC (CADR (CAR TL))
                                                                              FILE)
                                                                      (COND
                                                                         ((CDR TL)
                                                                          (IPRINC "," FILE]
                                                                (IPRINC "}" FILE)))
                                                            (ITERPRI FILE)))
                               (ENABLE (IPRINC "ENABLE" FILE)
                                       (PRINEVAL (PQUOTE (!LIST X))
                                              (BINDINGS (QUOTE X)
                                                        (CDR X))
                                              (IPOSITION FILE NIL NIL)
                                              FILE)
                                       (ITERPRI FILE))
                               (DISABLE (IPRINC "DISABLE" FILE)
                                        (PRINEVAL (PQUOTE (!LIST X))
                                                (BINDINGS (QUOTE X)
                                                          (CDR X))
                                                (IPOSITION FILE NIL NIL)
                                                FILE)
                                        (ITERPRI FILE))
                               (PROGN (PPRIND X (IPOSITION FILE NIL NIL)
                                              0 PPR-MACRO-LST FILE)
                                      (ITERPRI FILE])
```

(DUMP-LEMMA-TYPES

```
  [LAMBDA (TYPES)                                          (* kbr: "26-Oct-85 17:18")
    (IPRINC "(" FILE)
    [for TAIL on TYPES do (IPRINC [COND
                                     ((EQ (CAR TAIL)
                                          (QUOTE ELIM))
                                      (QUOTE ELIMINATION))
                                     (T (L-CASE (CAR TAIL]
                                  FILE)
                          (COND
                             ((NULL (CDR TAIL))
                              NIL)
                             ((NULL (CDDR TAIL))
                              (IPRINC "AND" FILE))
                             (T (IPRINC "," FILE)
                                (ISPACES 1 FILE]
    (IPRINC ")" FILE])
```

(DUMP-OTHER

```
  [LAMBDA (X INDEX)                                        (* kbr: "19-Oct-85 16:31")
    (DUMP-BEGIN-GROUP FILE)
    (COND
       (INDEX (IPRINC INDEX FILE)
              (IPRINC "." FILE)
```

```
            (ISPACES (IDIFFERENCE INDENT (IPOSITION FILE NIL NIL))
                  FILE))
      (T (ISPACES INDENT FILE)))
   (PPRIND X (IPOSITION FILE NIL NIL)
         0 NIL FILE)
   (ITERPRI FILE)
   (DUMP-END-GROUP FILE])
```

(DUMP-PROVE-LEMMA

```
  [LAMBDA (NAME TYPES THM HINT INDEX)                    (* kbr: "19-Oct-85 16:31")
    (DUMP-BEGIN-GROUP FILE)
    (COND
       (INDEX (IPRINC INDEX FILE)
              (IPRINC "." FILE)
              (ISPACES (IDIFFERENCE INDENT (IPOSITION FILE NIL NIL))
                    FILE))
       (T (ISPACES INDENT FILE)))
    (IPRINC "THEOREM." FILE)
    (IPRINC NAME FILE)
    (COND
       (TYPES (ISPACES 1 FILE)
              (DUMP-LEMMA-TYPES TYPES)))
    (IPRINT (QUOTE :)
          FILE)
    (ISPACES INDENT FILE)
    (PPRINDENT THM INDENT 0 FILE)
    (ITERPRI FILE)
    (COND
       (HINT (DUMP-HINTS HINT)))
    (DUMP-END-GROUP FILE])
```

(DUMP-TOGGLE

```
  [LAMBDA (NAME OLDNAME FLG INDEX)                       (* kbr: "19-Oct-85 16:31")
    (DUMP-BEGIN-GROUP FILE)
    (COND
       (INDEX (IPRINC INDEX FILE)
              (IPRINC "." FILE)
              (ISPACES (IDIFFERENCE INDENT (IPOSITION FILE NIL NIL))
                    FILE))
       (T (ISPACES INDENT FILE)))
    (COND
       (FLG (IPRINC "DISABLE" FILE))
       (T (IPRINC "ENABLE" FILE)))
    (IPRINC OLDNAME FILE)
    (IPRINC "." FILE)
    (ITERPRI FILE)
    (DUMP-END-GROUP FILE])

)

(RPAQQ CODE-E-MCOMS
       ((* CODE-E-M *)
        (FNS ELIMINABLE-VAR-CANDS ELIMINABLEP ELIMINATE-DESTRUCTORS-CANDIDATEP ELIMINATE-DESTRUCTORS-CANDIDATES
             ELIMINATE-DESTRUCTORS-CANDIDATES1 ELIMINATE-DESTRUCTORS-CLAUSE ELIMINATE-DESTRUCTORS-CLAUSE1
             ELIMINATE-DESTRUCTORS-SENT ELIMINATE-IRRELEVANCE-CLAUSE ELIMINATE-IRRELEVANCE-SENT
             EQUATIONAL-PAIR-FOR ERASE-EOL ERASE-EOP ERROR1 EVENT-FORM EVENT1-OCCURRED-BEFORE-EVENT2
             EVENTS-SINCE EVG EVG-OCCUR-LEGAL-CHAR-CODE-SEQ EVG-OCCUR-NUMBER EVG-OCCUR-OTHER EXECUTE
             EXPAND-ABBREVIATIONS EXPAND-AND-ORS EXPAND-BOOT-STRAP-NON-REC-FNS EXPAND-NON-REC-FNS
             EXPAND-PPR-MACROS EXTEND-ALIST EXTERNAL-LINEARIZE EXTRACT-DEPENDENCIES-FROM-HINTS FALSE-NONFALSEP
             FAVOR-COMPLICATED-CANDIDATES FERTILIZE-CLAUSE FERTILIZE-FEASIBLE FERTILIZE-SENT FERTILIZE1
             FILTER-ARGS FIND-EQUATIONAL-POLY FIRST-COEFFICIENT FIRST-VAR FITS FIXCAR-CDR FLATTEN-ANDS-IN-LIT
             FLESH-OUT-IND-PRIN FLUSH-CAND1-DOWN-CAND2 FN-SYMB0 FNNAMEP FNNAMEP-IF FORM-COUNT FORM-COUNT-EVG
             FORM-COUNT1 FORM-INDUCTION-CLAUSE FORMP-SIMPLIFIER FORMULA-OF FREE-VAR-CHK FREE-VARSP GEN-VARS
             GENERALIZE-CLAUSE GENERALIZE-SENT GENERALIZE1 GENERALIZE2 GENRLT1 GENRLTERMS GET-CANDS
             GET-LISP-SEXPR GET-LEVEL-NO GET-STACK-NAME GET-STACK-NAME1 GET-TYPES GREATEREQP
             GUARANTEE-CITIZENSHIP GUESS-RELATION-MEASURE-LST HAS-LIB-PROPS ILLEGAL-CALL ILLEGAL-NAME
             IMMEDIATE-DEPENDENTS-OF IMPLIES? IMPOSSIBLE-POLYP IND-FORMULA INDUCT INDUCT-VARS INDUCTION-MACHINE
             INFORM-SIMPLIFY INIT-LEMMA-STACK INIT-LIB INIT-LINEARIZE-ASSUMPTIONS-STACK INTERESTING-SUBTERMS
             INTERSECTP INTRODUCE-ANDS INTRODUCE-LISTS JUMPOUTP KILL-EVENT KILL-LIB KILLPROPLIST1
             LEGAL-CHAR-CODE-SEQ LENGTH-TO-ATOM LESSEQP LEXORDER LINEARIZE LISTABLE LOGSUBSETP LOOKUP-HYP
             LOOP-STOPPER MAIN-EVENT-OF CREATE-EVENT MAKE-FLATTENED-MACHINE MAKE-NEW-NAME MAKE-REWRITE-RULES
             MAKE-TYPE-RESTRICTION MAX-FORM-COUNT MAXIMAL-ELEMENTS MEANING-SIMPLIFIER MEMB-NEGATIVE MENTIONSQ
             MENTIONSQ-LST MERGE-CAND1-INTO-CAND2 MERGE-CANDS MERGE-DESTRUCTOR-CANDIDATES MERGE-TESTS-AND-ALISTS
             MERGE-TESTS-AND-ALISTS-LSTS META-LEMMAP MULTIPLE-PIGEON-HOLE)))

          (* * CODE-E-M *)

(DEFINEQ

(ELIMINABLE-VAR-CANDS
  [LAMBDA (CL HIST)                                      (* kbr: "19-Oct-85 16:31")
    HIST
    (SET-DIFF (ALL-VARS-LST CL)
          ELIM-VARIABLE-NAMES1])
```

(ELIMINABLEP

```
[LAMBDA (SET)                                                    (* kbr: "19-Oct-85 16:31")
  (OR (for LIT in SET always (PRIMITIVEP LIT))
      (AND (IEQP (LENGTH SET)
                 1)
           (OR [AND (for ARG in (SARGS (CAR SET)) always (VARIABLEP ARG))
                    (NO-DUPLICATESP (SARGS (CAR SET]
               (AND (EQ (FN-SYMB (CAR SET))
                        (QUOTE NOT))
                    (for ARG in (SARGS (ARGN (CAR SET)
                                             1))
                       always (VARIABLEP ARG))
                    (NO-DUPLICATESP (SARGS (ARGN (CAR SET)
                                                 1])
```

(ELIMINATE-DESTRUCTORS-CANDIDATEP

```
[LAMBDA (TERM)                                                   (* kbr: "19-Oct-85 16:31")

        (* Recognizes candidates for destructor elimination. It is assumed the input term is NVARIABLEP and not QUOTEP.
        To be a candidate the term must have an enabled destructor elim lemma.
        Furthermore, the crucial argument position of the term must be occupied by a variable or must itself be a candidate for
        elimination. Finally, if occupied by a variable, that variable must occur nowhere else in the arguments.
        Note that if the crucial arg is an eliminable term then the process of eliminating it will introduce a suitable distinct var.
        The answer returned is either NIL or else is the innermost term to be eliminated --
        possibly TERM itself. *)

   (PROG (LEMMA VAR)
         (SETQ LEMMA (GETPROP (FFN-SYMB TERM)
                              (QUOTE ELIMINATE-DESTRUCTORS-SEQ)))
         (COND
            ((OR (NULL LEMMA)
                 (DISABLEDP (fetch (REWRITE-RULE NAME) of LEMMA)))
             (RETURN NIL)))                                      (* We now identify the crucial arg.
                                                                 *)
         (SETQ VAR (for ARG in (FARGS TERM) as V in [FARGS (CAR (GETPROP (FFN-SYMB TERM)
                                                                         (QUOTE ELIMINATE-DESTRUCTORS-DESTS]
                     when (EQ V (FARGN (fetch (REWRITE-RULE CONCL) of LEMMA)
                                       2))
                     do (RETURN ARG)))
         (RETURN (COND
                    ((VARIABLEP VAR)                             (* If it is a variable, we make sure it occurs nowhere else.
                                                                 *)
                     (COND
                        ((for ARG in (FARGS TERM) as V in [FARGS (CAR (GETPROP (FFN-SYMB TERM)
                                                                               (QUOTE ELIMINATE-DESTRUCTORS-DESTS
                                                                                      ]
                            unless (EQ V (FARGN (fetch (REWRITE-RULE CONCL) of LEMMA)
                                                2))
                            never (OCCUR VAR ARG))
                         TERM)
                        (T NIL)))
                    (T (ELIMINATE-DESTRUCTORS-CANDIDATEP VAR])
```

(ELIMINATE-DESTRUCTORS-CANDIDATES

```
[LAMBDA (CL)                                                     (* kbr: "19-Oct-85 16:31")

        (* Returns a list of pockets. The CAR of each pocket is an eliminable destructor term.
        The CDR of each pocket is a list of all destructor terms that will in turn be eliminated as a result of eliminating the CAR.
        *)

   (LET (ANS)
        (for LIT in CL do (ELIMINATE-DESTRUCTORS-CANDIDATES1 LIT))
        (MERGE-DESTRUCTOR-CANDIDATES ANS])
```

(ELIMINATE-DESTRUCTORS-CANDIDATES1

```
[LAMBDA (TERM)                                                   (* kbr: "19-Oct-85 16:31")

        (* This function adds some lists to ANS. Each list has two elements.
        The first is a term that can be eliminated. The second is a term containing the first which will be eliminated in the same
        round as the first is eliminated. *)

   (COND
      ((OR (VARIABLEP TERM)
           (FQUOTEP TERM))
       NIL)
      (T (for ARG in (FARGS TERM) do (ELIMINATE-DESTRUCTORS-CANDIDATES1 ARG))
         (COND
            ((SETQ TEMP-TEMP (ELIMINATE-DESTRUCTORS-CANDIDATEP TERM))
             (SETQ ANS (ADD-TO-SET (LIST TEMP-TEMP TERM)
                                   ANS])
```

```
(ELIMINATE-DESTRUCTORS-CLAUSE
  [LAMBDA (CL HIST)                                          (* kbr: "20-Oct-85 19:34")
    (LET
     (ELIMINABLES NEW-CL TO-DO CANDS REWRITE-RULE HYPS LHS RHS DESTS ALIST INST-DESTS INST-RHS INST-LHS
            INST-HYPS)

          (* TO-DO is a list that controls the elimination. The invariant maintained is that the all the clauses in PROCESS-CLAUSES
          and all the clauses in TO-DO are theorems then so is the initial CL.
          When a clause is removed from TO-DO either it is added to PROCESS-CLAUSES or else an elimination is performed on it
          and the resulting cases are all added to TO-DO for any additional elims required on the new variables introduced.
          TO-DO is a list of pockets. Each pocket contains a clause, the list of all variables in the clause not introduced by an elim,
          and some candidate destructor pockets. The candidate destructor pockets each contain in their CAR a term that might be
          eliminated and in their CDR all of the terms that could recursively be eliminated should the CAR be eliminated.
          These pockets are ordered from most desirable elim to least desirable elim.
          At the moment the ordering is determined by the sum of the level numbers of the terms in the CDRs.
          *)

     [SETQ TO-DO (LIST (LIST CL (ELIMINABLE-VAR-CANDS CL HIST)
                               (SORT-DESTRUCTOR-CANDIDATES (ELIMINATE-DESTRUCTORS-CANDIDATES CL]
     (SETQ PROCESS-CLAUSES NIL)
     (SETQ PROCESS-HIST NIL)
     [while TO-DO
        do
        (SETQ CL (CAAR TO-DO))
        (SETQ ELIMINABLES (CADAR TO-DO))
        (SETQ CANDS (CADDAR TO-DO))
        (SETQ TO-DO (CDR TO-DO))
        (COND
           ((OR (NULL ELIMINABLES)
                (NULL CANDS))
            (SETQ PROCESS-CLAUSES (CONS CL PROCESS-CLAUSES)))
           [(for CAND-TAIL on CANDS bind CAND
               thereis
               (PROGN                                        (* CAND is the candidate destructor term to be eliminated.
                                                             *)
                      (SETQ CAND (CAR (CAR CAND-TAIL)))
                      (SETQ REWRITE-RULE (GETPROP (FFN-SYMB CAND)
                                                  (QUOTE ELIMINATE-DESTRUCTORS-SEQ)))

          (* We know this rule is not disabled because ELIMINATE-DESTRUCTORS-CANDIDATES checks DISABLED-LEMMAS
          before saying a term is a candidate. *)

                      (SETQ HYPS (fetch (REWRITE-RULE HYPS) of REWRITE-RULE))
                      (SETQ LHS (FARGN (fetch (REWRITE-RULE CONCL) of REWRITE-RULE)
                                       1))
                      (SETQ RHS (FARGN (fetch (REWRITE-RULE CONCL) of REWRITE-RULE)
                                       2))
                      (SETQ DESTS (GETPROP (FFN-SYMB CAND)
                                           (QUOTE ELIMINATE-DESTRUCTORS-DESTS)))
                      (SETQ ALIST (for VAR in (FARGS (CAR DESTS)) as VAL in (FARGS CAND)
                                     collect (CONS VAR VAL)))
                      (SETQ INST-RHS (SUBLIS-VAR ALIST RHS))
                      (COND
                         ((AND (MEMB INST-RHS ELIMINABLES)
                               (for HYP in HYPS never (MEMBER (SUBLIS-VAR ALIST HYP)
                                                              CL)))
                          (SETQ INST-DESTS (SUBLIS-VAR-LST ALIST DESTS))
                          (SETQ INST-HYPS (SUBLIS-VAR-LST ALIST HYPS))
                          (SETQ INST-LHS (SUBLIS-VAR ALIST LHS))
                          (SETQ TO-DO (APPEND [for HYP in INST-HYPS unless (EQUAL TRUE-CLAUSE
                                                                                  (SETQ NEW-CL
                                                                                   (ADD-LITERAL HYP CL NIL)))
                                                 collect (LIST NEW-CL ELIMINABLES
                                                               (COND
                                                                  (PROCESS-HIST (for POCKET
                                                                                   in (CDR CAND-TAIL)
                                                                                   unless (MEMBER (CAR POCKET)
                                                                                                  INST-DESTS)
                                                                                   collect POCKET))
                                                                  (T NIL]
                                              TO-DO))
                          (SETQ NEW-CL (ELIMINATE-DESTRUCTORS-CLAUSE1 CL INST-HYPS INST-LHS INST-RHS
                                                INST-DESTS))
                          [COND
                             ((NOT (EQUAL TRUE-CLAUSE NEW-CL))
                              (SETQ TO-DO
                               (CONS [LIST NEW-CL (UNIONQ GENERALIZING-SKOS (REMOVE INST-RHS ELIMINABLES))
                                           (SORT-DESTRUCTOR-CANDIDATES
                                            (MERGE-DESTRUCTOR-CANDIDATES
                                             (UNION-EQUAL (COND
                                                             (PROCESS-HIST (for POCKET in (CDR CAND-TAIL)
                                                                              when (OCCUR-LST (CAR POCKET)
                                                                                              NEW-CL)
                                                                              collect POCKET))
                                                             (T NIL))
                                                          (for POCKET in (ELIMINATE-DESTRUCTORS-CANDIDATES NEW-CL)
                                                             when (for VAR in (FARGS (CAR POCKET))
```

```
                                                        thereis (MEMB VAR GENERALIZING-SKOS))
                                                  collect POCKET]
                                      TO-DO]
                            (SETQ PROCESS-HIST (CONS (LIST (fetch (REWRITE-RULE NAME) of REWRITE-RULE)
                                                           INST-DESTS OBVIOUS-RESTRICTIONS GENERALIZE-LEMMA-NAMES
                                                           INST-RHS (SUB-PAIR-EXPR INST-DESTS GENERALIZING-SKOS
                                                                                   INST-LHS))
                                                     PROCESS-HIST))
                           T)
                          (T NIL]
          (T (SETQ PROCESS-CLAUSES (CONS CL PROCESS-CLAUSES]
     [for PAIR in PROCESS-HIST do (SETQ ALL-LEMMAS-USED (UNION-EQUAL (CADDDR PAIR)
                                                                     (ADD-TO-SET (CAR PAIR)
                                                                                 ALL-LEMMAS-USED]
     (SETQ PROCESS-CLAUSES (SCRUNCH-CLAUSE-SET PROCESS-CLAUSES))
     (NOT (NULL PROCESS-HIST])
```

(ELIMINATE-DESTRUCTORS-CLAUSE1

```
  [LAMBDA (CL HYPS LHS RHS DESTS)                          (* kbr: "19-Oct-85 16:31")
    (LET (GEN-CL GEN-LHS CL1)
         (SETQ CL1 CL)                                     (* We preserve the order of the hyps just for the hell of it.
                                                           *)
         (for HYP in (REVERSE HYPS) do (SETQ CL1 (ADD-LITERAL (NEGATE-LIT HYP)
                                                              CL1 NIL)))
         (SETQ GEN-CL (GENERALIZE1 CL1 DESTS ELIM-VARIABLE-NAMES1))
         (SETQ GEN-LHS (SUB-PAIR-EXPR DESTS GENERALIZING-SKOS LHS))
         (SUBST-VAR-LST GEN-LHS RHS GEN-CL])
```

(ELIMINATE-DESTRUCTORS-SENT

```
  [LAMBDA (CL HIST)                                        (* kbr: "19-Oct-85 16:31")
    (EXECUTE (QUOTE ELIMINATE-DESTRUCTORS-CLAUSE)
           CL HIST (QUOTE SIMPLIFY-SENT)
           (QUOTE FERTILIZE-SENT])
```

(ELIMINATE-IRRELEVANCE-CLAUSE

```
  [LAMBDA (CL HIST)                                        (* kbr: "19-Oct-85 20:00")
    HIST
    (PROG (PARTITION ELIMINABLES)
          (COND
             ((NOT (ASSOC (QUOTE BEING-PROVED)
                          STACK))
              (RETURN NIL)))
          [SETQ PARTITION (TRANSITIVE-CLOSURE (for LIT in CL collect (CONS (ALL-VARS LIT)
                                                                           (LIST LIT)))
                                              (FUNCTION (LAMBDA (X Y)
                                                          (COND
                                                             [(INTERSECTP (CAR X)
                                                                          (CAR Y))
                                                              (CONS (UNION-EQUAL (CAR X)
                                                                                 (CAR Y))
                                                                    (UNION-EQUAL (CDR X)
                                                                                 (CDR Y]
                                                            (T NIL]
          (SETQ ELIMINABLES (for PAIR in PARTITION when (ELIMINABLEP (CDR PAIR)) join (CDR PAIR)))
          (COND
             ((NULL ELIMINABLES)
              (RETURN NIL))
             (T (SETQ PROCESS-CLAUSES (LIST (for LIT in CL unless (MEMB LIT ELIMINABLES) collect LIT)))
                (SETQ PROCESS-HIST NIL)
                (RETURN T])
```

(ELIMINATE-IRRELEVANCE-SENT

```
  [LAMBDA (CL HIST)                                        (* kbr: "19-Oct-85 16:31")
    (EXECUTE (QUOTE ELIMINATE-IRRELEVANCE-CLAUSE)
           CL HIST (QUOTE STORE-SENT)
           (QUOTE STORE-SENT])
```

(EQUATIONAL-PAIR-FOR

```
  [LAMBDA (WINNING-PAIR POLY)                              (* kbr: "19-Oct-85 16:31")
    (CONS (CAR WINNING-PAIR)
          (CONS-PLUS (LIST (QUOTE QUOTE)
                           (ABS (fetch (POLY CONSTANT) of POLY)))
                (BUILD-SUM WINNING-PAIR (fetch (POLY ALIST) of POLY])
```

(ERASE-EOL

```
  [LAMBDA NIL                                              (* kbr: "19-Oct-85 16:31")
    (CURSORPOS (QUOTE L])
```

(ERASE-EOP

```
  [LAMBDA NIL                                              (* kbr: "19-Oct-85 16:31")
```

```
    (CURSORPOS (QUOTE E])
```

(ERROR1

```
  [LAMBDA (SENTENCE ALIST HARDNESS)                          (* kbr: "20-Apr-86 16:18")
    (SETQ ALIST (CONS (CONS (QUOTE SENTENCE)
                            SENTENCE)
                      (CONS (CONS (QUOTE HARDNESS)
                                  HARDNESS)
                            ALIST)))
    [COND
       ((NULL HARDNESS)
        (SETQ HARDNESS (QUOTE HARD]
    (PRINEVAL (PQUOTE (PROGN // (COND
                                   ((EQ HARDNESS (QUOTE WARNING))
                                    WARNING)
                                   ((EQ HARDNESS (QUOTE HARD))
                                    FATAL ERROR)
                                   (T ERROR))
                             :
                             (@ SENTENCE)
                             // //))
          ALIST 0 PROVE-FILE)
    (COND
       ((NEQ TTY-FILE PROVE-FILE)
        (PRINEVAL (PQUOTE (PROGN // (COND
                                       ((EQ HARDNESS (QUOTE WARNING))
                                        WARNING)
                                       ((EQ HARDNESS (QUOTE HARD))
                                        FATAL ERROR)
                                       (T ERROR))
                                 :
                                 (@ SENTENCE)
                                 // //))
              ALIST 0 TTY-FILE)))
    (COND
       (DEBUGFLG (HELP)))
    (COND
       ((EQ HARDNESS (QUOTE WARNING))
        NIL)
       ((DETACHEDP)
        (DETACHED-ERROR)
        (ERROR1 SENTENCE (CDDR ALIST)
                HARDNESS))
       ((AND (EQ HARDNESS (QUOTE SOFT))
             IN-REDO-UNDONE-EVENTS-FLG)
        (RETFROM (QUOTE APPLY)
                 (QUOTE *****ERROR*****)))
       (T (ERROR (LIST (QUOTE ERROR1)
                       SENTENCE ALIST])
```

(EVENT-FORM

```
  [LAMBDA (X)                                                (* kbr: "19-Oct-85 16:31")
    (AND (LITATOM X)
         (OR (GETPROP X (QUOTE EVENT))
             (AND (GETPROP X (QUOTE MAIN-EVENT))
                  (GETPROP (GETPROP X (QUOTE MAIN-EVENT))
                           (QUOTE EVENT])
```

(EVENT1-OCCURRED-BEFORE-EVENT2

```
  [LAMBDA (EVENT1 EVENT2 EVENT-LST)                          (* kbr: "19-Oct-85 16:31")
    (COND
       ((MEMB EVENT1 (CDR (MEMB EVENT2 EVENT-LST)))
        T)
       (T NIL])
```

(EVENTS-SINCE

```
  [LAMBDA (EVENT)                                            (* kbr: "19-Oct-85 16:31")
    (COND
       ((MEMB EVENT CHRONOLOGY)
        (CONS (GETPROP EVENT (QUOTE EVENT))
              (DREVERSE (for E in CHRONOLOGY until (EQ E EVENT) collect (GETPROP E (QUOTE EVENT])
```

(EVG

```
  [LAMBDA (Y)                                                (* kbr: "19-Oct-85 16:31")
    (COND
       ((NLISTP Y)
        (COND
           ((FIXP Y)
            (COND
               ((GREATEREQP Y 0)
                TYPE-SET-NUMBERS)
               (T TYPE-SET-NEGATIVES)))
```

```
          ((EQ Y *1*T)
           TYPE-SET-TRUE)
          ((EQ Y *1*F)
           TYPE-SET-FALSE)
          ((ILLEGAL-NAME Y)
           NIL)
          (T TYPE-SET-LITATOMS)))
      ((EQ (CAR Y)
           *1*SHELL-QUOTE-MARK)
       (COND
          [[AND (LISTP (CDR Y))
                (EQ (CDR (LAST Y))
                    NIL)
                (IEQP (LENGTH (CDDR Y))
                      (ARITY (CADR Y)))
                [OR (MEMB (CADR Y)
                          *1*BTM-OBJECTS)
                    (AND (ASSOC (CADR Y)
                                SHELL-ALIST)
                         (for RESTRICTION in (GETPROP (CADR Y)
                                                      (QUOTE TYPE-RESTRICTIONS))
                            as ARG in (CDDR Y) always (AND (SETQ TEMP-TEMP (EVG ARG))
                                                           (LOGSUBSETP TEMP-TEMP (fetch (TYPE-RESTRICTION
                                                                                          TYPE-SET)
                                                                                  of RESTRICTION]
                (COND
                   [(EQ (CADR Y)
                        (QUOTE PACK))
                    (NOT (LEGAL-CHAR-CODE-SEQ (CADDR Y]
                   ((EQ (CADR Y)
                        (QUOTE MINUS))
                    (EQUAL (CADDR Y)
                           0))
                   (T (NOT (MEMB (CADR Y)
                                 (QUOTE (ADD1 ZERO CONS]
           (CAR (TYPE-PRESCRIPTION (CADR Y]
          (T NIL)))
      ((AND (EVG (CAR Y))
            (EVG (CDR Y)))
       TYPE-SET-CONS)
      (T NIL])
```

(EVG-OCCUR-LEGAL-CHAR-CODE-SEQ

```
[LAMBDA (L EVG)                                                    (* kbr: "19-Oct-85 16:31")
  (COND
     [(NLISTP EVG)
      (COND
         ((EQ EVG *1*T)
          NIL)
         ((EQ EVG *1*F)
          NIL)
         ((FIXP EVG)
          NIL)
         ((LESSP (NCHARS EVG)
                 (LENGTH-TO-ATOM L))
          NIL)
         (T (for TAIL on L until (NLISTP TAIL) as J from (ADD1 (IDIFFERENCE (NCHARS EVG)
                                                                           (LENGTH-TO-ATOM L)))
               always (IEQP (CAR TAIL)
                            (NTHCHARCODE EVG J]
     ((EQ (CAR EVG)
          *1*SHELL-QUOTE-MARK)
      (for ARG in (CDDR EVG) thereis (EVG-OCCUR-LEGAL-CHAR-CODE-SEQ L ARG)))
     ((EQUAL L EVG)
      T)
     (T (OR (EVG-OCCUR-LEGAL-CHAR-CODE-SEQ L (CAR EVG))
            (EVG-OCCUR-LEGAL-CHAR-CODE-SEQ L (CDR EVG])
```

(EVG-OCCUR-NUMBER

```
[LAMBDA (N EVG)                                                    (* kbr: "17-Nov-85 17:08")
  (COND
     [(NLISTP EVG)
      (COND
         ((EQ EVG *1*T)
          NIL)
         ((EQ EVG *1*F)
          NIL)
         [(FIXP EVG)
          (COND
             ((LESSP N 0)
              (EQUAL N EVG))
             (T (LESSEQP N (ABS EVG]
         ((LESSP N 0)
          NIL)
         ((GREATERP N (CHARCODE Z))
```

```
           NIL)
          ((LESSEQP N (CHARCODE -))
           T)
          (T (for I from 1 to (NCHARS EVG) thereis (LESSEQP N (NTHCHARCODE EVG I]
      ((EQ (CAR EVG)
           *1*SHELL-QUOTE-MARK)
       (for ARG in (CDDR EVG) thereis (EVG-OCCUR-NUMBER N ARG)))
      (T (OR (EVG-OCCUR-NUMBER N (CAR EVG))
             (EVG-OCCUR-NUMBER N (CDR EVG])
```

(EVG-OCCUR-OTHER

```
[LAMBDA (X EVG)                                                    (* kbr: "19-Oct-85 16:31")
```

(* X must be an evg other than a FIXP or a LEGAL-CHAR-CODE-SEQ with 0 final CDR.
*)

```
  (COND
     ((EQUAL X EVG)
      T)
     ((NLISTP EVG)
      NIL)
     ((EQ (CAR EVG)
          *1*SHELL-QUOTE-MARK)
      (for ARG in (CDDR EVG) thereis (EVG-OCCUR-OTHER X ARG)))
     (T (OR (EVG-OCCUR-OTHER X (CAR EVG))
            (EVG-OCCUR-OTHER X (CDR EVG])
```

(EXECUTE

```
[LAMBDA (PROCESS CL HIST NORMAL-EXIT NO-CHANGE-EXIT)               (* kbr: "19-Oct-85 16:31")
  (LET (NEW-HIST)
       (COND
          ((APPLY* PROCESS CL HIST)
           (SETQ NEW-HIST (ADD-PROCESS-HIST PROCESS CL HIST PROCESS-CLAUSES PROCESS-HIST))
           (for CL1 in PROCESS-CLAUSES do (APPLY* NORMAL-EXIT CL1 NEW-HIST)))
          (T (APPLY* NO-CHANGE-EXIT CL HIST])
```

(EXPAND-ABBREVIATIONS

```
[LAMBDA (TERM ALIST)                                               (* kbr: "19-Oct-85 16:31")
```

(* Apply all unconditional rewrite rules and nonrec defns that are ABBREVIATIONPs.
Adds to ABBREVIATIONS-USED the names of the lemmas and fns applied.
*)

```
  (LET (TEMP LEMMA RHS LHS)
       (COND
          ((VARIABLEP TERM)
           (COND
              ((SETQ TEMP (ASSOC TERM ALIST))
               (CDR TEMP))
              (T TERM)))
          ((FQUOTEP TERM)
           TERM)
          [(MEMB (FFN-SYMB TERM)
                 FNS-TO-BE-IGNORED-BY-REWRITE)
           (CONS-TERM (FFN-SYMB TERM)
                  (for ARG in (FARGS TERM) collect (EXPAND-ABBREVIATIONS ARG ALIST]
          [(AND (SETQ TEMP (NON-RECURSIVE-DEFNP (FFN-SYMB TERM)))
                (ABBREVIATIONP (CADR TEMP)
                       (CADDR TEMP)))
           (SETQ ABBREVIATIONS-USED (ADD-TO-SET (FFN-SYMB TERM)
                                        ABBREVIATIONS-USED))
           (EXPAND-ABBREVIATIONS (CADDR TEMP)
                  (for V in (CADR TEMP) as ARG in (FARGS TERM) collect (CONS V (EXPAND-ABBREVIATIONS ARG ALIST]
          (T [SETQ TERM (CONS-TERM (FFN-SYMB TERM)
                                   (for ARG in (FARGS TERM) collect (EXPAND-ABBREVIATIONS ARG ALIST]
             (COND
                ((FQUOTEP TERM)
                 TERM)
                ((SETQ LEMMA (for LEMMA in (GETPROP (FFN-SYMB TERM)
                                                    (QUOTE LEMMAS))
                                  when (AND (NOT (DISABLEDP (fetch (REWRITE-RULE NAME) of LEMMA)))
                                            (NOT (META-LEMMAP LEMMA))
                                            (NULL (fetch (REWRITE-RULE HYPS) of LEMMA))
                                            (NULL (fetch (REWRITE-RULE LOOP-STOPPER) of LEMMA))
                                            (BM-MATCH (fetch (REWRITE-RULE CONCL) of LEMMA)
                                                   (EQUAL LHS RHS))
                                            (ABBREVIATIONP (ALL-VARS-BAG LHS)
                                                   RHS)
                                            (ONE-WAY-UNIFY LHS TERM))
                                  do (RETURN LEMMA)))
                 (SETQ ABBREVIATIONS-USED (ADD-TO-SET (fetch (REWRITE-RULE NAME) of LEMMA)
                                                      ABBREVIATIONS-USED))
                 (EXPAND-ABBREVIATIONS RHS UNIFY-SUBST))
                (T TERM])
```

(EXPAND-AND-ORS

```
[LAMBDA (TERM BOOL)                                                    (* kbr: "19-Oct-85 16:31")

          (* Expands the top-level fn symbol of TERM provided the expansion produces an AND --
          when BOOL is FALSE -- or OR -- when BOOL is TRUE -- or returns NIL if no expansion is appropriate.
          Side-effects ABBREVIATIONS-USED. *)

    (LET (TEMP LEMMA RHS LHS C2 C3)
         (COND
            ((VARIABLEP TERM)
             NIL)
            ((FQUOTEP TERM)
             NIL)
            ([AND (SETQ TEMP (NON-RECURSIVE-DEFNP (FFN-SYMB TERM)))
                  (OR (AND [BM-MATCH (CADDR TEMP)
                                     (COND
                                        ((& C2 C3]
                                   (OR (EQUAL C2 BOOL)
                                       (EQUAL C3 BOOL)))
                              (COND
                                 ((EQUAL BOOL FALSE)
                                  (BM-MATCH (CADDR TEMP)
                                            (AND & &)))
                                 (T (BM-MATCH (CADDR TEMP)
                                              (OR & &]
                           (SETQ ABBREVIATIONS-USED (ADD-TO-SET (FFN-SYMB TERM)
                                                                ABBREVIATIONS-USED))
                           (EXPAND-ABBREVIATIONS (SUB-PAIR-VAR (CADR TEMP)
                                                               (FARGS TERM)
                                                               (CADDR TEMP))
                                  NIL))
             ((SETQ LEMMA (for LEMMA in (GETPROP (FFN-SYMB TERM)
                                                 (QUOTE LEMMAS))
                               when (AND (NOT (DISABLEDP (fetch (REWRITE-RULE NAME) of LEMMA)))
                                         (NOT (META-LEMMAP LEMMA))
                                         (NULL (fetch (REWRITE-RULE HYPS) of LEMMA))
                                         (NULL (fetch (REWRITE-RULE LOOP-STOPPER) of LEMMA))
                                         (BM-MATCH (fetch (REWRITE-RULE CONCL) of LEMMA)
                                                   (EQUAL LHS RHS))
                                         [BM-MATCH RHS (COND
                                                          ((& C2 C3]
                                         (OR (EQUAL C2 BOOL)
                                             (EQUAL C3 BOOL))
                                         (ONE-WAY-UNIFY LHS TERM))
                               do (RETURN LEMMA)))
              (SETQ ABBREVIATIONS-USED (ADD-TO-SET (fetch (REWRITE-RULE NAME) of LEMMA)
                                                   ABBREVIATIONS-USED))
              (EXPAND-ABBREVIATIONS (SUBLIS-VAR UNIFY-SUBST RHS)
                     NIL))
             (T NIL])
```

(EXPAND-BOOT-STRAP-NON-REC-FNS

```
[LAMBDA (TERM)                                                         (* kbr: "19-Oct-85 16:31")
  (COND
     ((VARIABLEP TERM)
      TERM)
     ((FQUOTEP TERM)
      TERM)
     [(MEMB (FFN-SYMB TERM)
            (QUOTE (AND OR NOT IMPLIES FIX ZEROP NLISTP)))
      (EXPAND-BOOT-STRAP-NON-REC-FNS (SUB-PAIR-VAR (CADR (GETPROP (FFN-SYMB TERM)
                                                                  (QUOTE SDEFN)))
                                                   (FARGS TERM)
                                                   (CADDR (GETPROP (FFN-SYMB TERM)
                                                                   (QUOTE SDEFN]
     (T (CONS-TERM (FFN-SYMB TERM)
                   (for ARG in (FARGS TERM) collect (EXPAND-BOOT-STRAP-NON-REC-FNS ARG])
```

(EXPAND-NON-REC-FNS

```
[LAMBDA (TERM)                                                         (* kbr: "19-Oct-85 16:31")
  (COND
     ((VARIABLEP TERM)
      TERM)
     ((FQUOTEP TERM)
      TERM)
     [(NON-RECURSIVE-DEFNP (FFN-SYMB TERM))
      (EXPAND-NON-REC-FNS (SUB-PAIR-VAR (CADR (GETPROP (FFN-SYMB TERM)
                                                       (QUOTE SDEFN)))
                                        (FARGS TERM)
                                        (CADDR (GETPROP (FFN-SYMB TERM)
                                                        (QUOTE SDEFN]
     (T (CONS-TERM (FFN-SYMB TERM)
                   (for ARG in (FARGS TERM) collect (EXPAND-NON-REC-FNS ARG])
```

(EXPAND-PPR-MACROS

```
[LAMBDA (TERM)                                                    (* kbr: "19-Oct-85 16:31")
```

(* As currently defined and used, this fn is a crock. It binds PPR-MACRO-LST apparently so that the macro defns on PPR-MACRO-LST can smash the list so that while processing the value delivered by a macro macros are not expanded. This appears to be used by CONVERT-QUOTE so that after (QUOTE evg) has been processed at the top level -- possibly changing into something like a number or NIL or TRUE but possibly being unchanged -- the recursive processing of evg does not cause macro expansion -- e.g., (QUOTE (CAR (CAR X))) is otherwise changed into (* As *) *)

```
   (LET ((PPR-MACRO-LST PPR-MACRO-LST))
        (COND
           ((NLISTP TERM)
            TERM)
           [(SETQ TEMP-TEMP (ASSOC (CAR TERM)
                                   PPR-MACRO-LST))
            (SETQ TEMP-TEMP (APPLY* (CDR TEMP-TEMP)
                                    TERM))
            (COND
               ((NLISTP TEMP-TEMP)
                TEMP-TEMP)
               ((EQ (CAR TEMP-TEMP)
                    (QUOTE QUOTE))
                TEMP-TEMP)
               (T (CONS (CAR TEMP-TEMP)
                        (for ARG in (CDR TEMP-TEMP) collect (EXPAND-PPR-MACROS ARG]
           (T (CONS (CAR TERM)
                    (for ARG in (CDR TERM) collect (EXPAND-PPR-MACROS ARG])
```

(EXTEND-ALIST

```
[LAMBDA (ALIST1 ALIST2)                                           (* kbr: "19-Oct-85 16:31")
```

(* Extend ALIST2 by adding to it every pair from ALIST1 that does not conflict with an existing pair in ALIST2. *)

```
   (APPEND ALIST2 (for X in ALIST1 unless (ASSOC (CAR X)
                                                 ALIST2)
                     collect X])
```

(EXTERNAL-LINEARIZE

```
[LAMBDA (TERM FLG)                                                (* kbr: "19-Oct-85 16:31")
   (LET (HEURISTIC-TYPE-ALIST LITS-THAT-MAY-BE-ASSUMED-FALSE)
        (LINEARIZE TERM FLG])
```

(EXTRACT-DEPENDENCIES-FROM-HINTS

```
[LAMBDA (HINTS)                                                   (* kbr: "19-Oct-85 16:31")
   (for HINT in HINTS bind LOOP-ANS do (SETQ LOOP-ANS (UNIONQ (SELECTQ (CAR HINT)
                                                                       (USE (for X in (CDR HINT)
                                                                               collect (CAR X)))
                                                                       (INDUCT [LIST (FFN-SYMB (TRANSLATE
                                                                                                (CADR HINT])
                                                                       (NIL))
                                                              LOOP-ANS))
      finally (RETURN LOOP-ANS])
```

(FALSE-NONFALSEP

```
[LAMBDA (TERM)                                                    (* kbr: "29-Jun-86 17:51")
```

(* Returns T if TERM is definitely true or false. As a side effect, DEFINITELY-FALSE is set to T to indicate which of definitely true or false. *)

```
   (LET (TEMP)
        (COND
           ((VALUEP TERM)
            (SETQ DEFINITELY-FALSE (EQUAL TERM FALSE))
            T)
           (T (SETQ TEMP (TYPE-SET TERM))
              (COND
                 ((IEQP TEMP TYPE-SET-FALSE)
                  (SETQ DEFINITELY-FALSE T)
                  T)
                 ((IEQP 0 (LOGAND TEMP TYPE-SET-FALSE))
                  (SETQ DEFINITELY-FALSE NIL)
                  T)
                 (T NIL])
```

(FAVOR-COMPLICATED-CANDIDATES

```
[LAMBDA (CANDLST)                                                 (* kbr: "19-Oct-85 16:31")
   (MAXIMAL-ELEMENTS CANDLST (FUNCTION (LAMBDA (CAND)
                                       (for TERM in (CONS (fetch (CANDIDATE INDUCTION-TERM) of CAND)
```

```
                                                    (fetch (CANDIDATE OTHER-TERMS) of CAND))
                                          count (NOT (PRIMITIVE-RECURSIVEP (FN-SYMB TERM])
```

(FERTILIZE-CLAUSE

```
[LAMBDA (CL HIST)                                                    (* kbr: "19-Oct-85 16:31")
  (PROG (LIT LHS1 RHS1 LHS2 RHS2 DONT-DELETE-LIT-FLG MASS-SUBST-FLG CROSS-FERT-FLG DIRECTION)
        (SETQ LIT (for LIT in CL when (AND (BM-MATCH LIT (NOT (EQUAL LHS1 RHS1)))
                                           (SETQ DIRECTION (FERTILIZE1 LIT CL LHS1 RHS1 HIST)))
                      do (RETURN LIT)))
        (COND
           ((NULL LIT)
            (RETURN NIL)))
        (SETQ MASS-SUBST-FLG (OR (VALUEP LHS1)
                                 (VALUEP RHS1)))
        [SETQ DONT-DELETE-LIT-FLG (OR (VALUEP LHS1)
                                      (VALUEP RHS1)
                                      (AND (NOT (AND IN-PROVE-LEMMA-FLG (ASSOC (QUOTE INDUCT)
                                                                               HINTS)))
                                           (NOT (ASSOC (QUOTE BEING-PROVED)
                                                       STACK]
        [SETQ CROSS-FERT-FLG (AND (ASSOC (QUOTE BEING-PROVED)
                                         STACK)
                                  [for LIT2 in CL thereis (AND (BM-MATCH LIT2 (EQUAL LHS2 RHS2))
                                                               (COND
                                                                  ((EQ DIRECTION (QUOTE LEFT-FOR-RIGHT))
                                                                   (OCCUR RHS1 RHS2))
                                                                  (T (OCCUR LHS1 LHS2]
                                  (for LIT2 in CL thereis (AND (BM-MATCH LIT2 (EQUAL LHS2 RHS2))
                                                               (COND
                                                                  ((EQ DIRECTION (QUOTE LEFT-FOR-RIGHT))
                                                                   (OCCUR RHS1 LHS2))
                                                                  (T (OCCUR LHS1 RHS2]
        [SETQ PROCESS-CLAUSES (LIST (for LIT2 in CL when (OR DONT-DELETE-LIT-FLG (NEQ LIT LIT2))
                                           collect (COND
                                                     ((EQ LIT LIT2)
                                                      LIT)
                                                     [[OR MASS-SUBST-FLG (NOT CROSS-FERT-FLG)
                                                          (BM-MATCH LIT2 (NOT (EQUAL & &]
                                                      (COND
                                                         ((EQ DIRECTION (QUOTE LEFT-FOR-RIGHT))
                                                          (BM-SUBST LHS1 RHS1 LIT2))
                                                         (T (BM-SUBST RHS1 LHS1 LIT2]
                                                     [(BM-MATCH LIT2 (EQUAL LHS2 RHS2))
                                                      (COND
                                                         ((EQ DIRECTION (QUOTE LEFT-FOR-RIGHT))
                                                          (FCONS-TERM* (QUOTE EQUAL)
                                                                 LHS2
                                                                 (BM-SUBST LHS1 RHS1 RHS2)))
                                                         (T (FCONS-TERM* (QUOTE EQUAL)
                                                                   (BM-SUBST RHS1 LHS1 LHS2)
                                                                   RHS2]
                                                     (T LIT2]
        (SETQ PROCESS-HIST (LIST MASS-SUBST-FLG CROSS-FERT-FLG DIRECTION LHS1 RHS1 DONT-DELETE-LIT-FLG))
        (RETURN T])
```

(FERTILIZE-FEASIBLE

```
[LAMBDA (LIT CL TERM HIST)                                           (* kbr: "19-Oct-85 16:31")
  (AND (NOT (ALMOST-VALUEP TERM))
       (OR (VARIABLEP TERM)
           (NOT (SKO-DEST-NESTP TERM NIL)))
       (for LIT2 in CL when (NEQ LIT2 LIT) thereis (OCCUR TERM LIT2))
       (NOT (for ENTRY in HIST bind (LHS RHS)
               thereis (AND (BM-MATCH ENTRY (FERTILIZE-CLAUSE & & & & LHS RHS &))
                            (EQUAL (FARGN (FARGN LIT 1)
                                          1)
                                   LHS)
                            (EQUAL (FARGN (FARGN LIT 1)
                                          2)
                                   RHS])
```

(FERTILIZE-SENT

```
[LAMBDA (CL HIST)                                                    (* kbr: "19-Oct-85 16:31")
  (EXECUTE (QUOTE FERTILIZE-CLAUSE)
           CL HIST (QUOTE SIMPLIFY-SENT)
           (QUOTE GENERALIZE-SENT])
```

(FERTILIZE1

```
[LAMBDA (LIT CL LHS RHS HIST)                                        (* kbr: "19-Oct-85 16:31")
  (COND
     [(FERTILIZE-FEASIBLE LIT CL LHS HIST)
      (COND
         [(FERTILIZE-FEASIBLE LIT CL RHS HIST)
          (COND
```

```
            ((LESSP (COMPLEXITY LHS)
                    (COMPLEXITY RHS))
             (QUOTE LEFT-FOR-RIGHT))
            (T (QUOTE RIGHT-FOR-LEFT]
         (T (QUOTE RIGHT-FOR-LEFT]
      ((FERTILIZE-FEASIBLE LIT CL RHS HIST)
       (QUOTE LEFT-FOR-RIGHT))
      (T NIL])
```

(FILTER-ARGS

```
[LAMBDA (SUBSET FORMALS ARGS)                                   (* kbr: "19-Oct-85 16:31")
  (for VAR in SUBSET collect (for TERM in ARGS as FORMAL in FORMALS when (EQ FORMAL VAR) do (RETURN TERM])
```

(FIND-EQUATIONAL-POLY

```
[LAMBDA (HIST POT)                                              (* kbr: "19-Oct-85 16:31")

        (* Look for an equation to be derived from this pot. If one is found, add to LEMMAS-USED-BY-LINEAR and
        LINEAR-ASSUMPTIONS the appropriate entries from the two polys involved.
        In addition, add an extra entry to LEMMAS-USED-BY-LINEAR to store the fact that this equation has been deduced.
        Finally, do not do any of this if HIST records that the deduced equation has been previously deduced.
        See the comment in PROCESS-EQUATIONAL-POLYS for details.
        *)

   (for POLY1 in (fetch (LINEAR-POT POSITIVES) of POT) bind (WINNING-PAIR POLY2 PAIR HYP1 HYP2)
      when (SETQ TEMP-TEMP (TRIVIAL-POLYP POLY1))
      do (SETQ WINNING-PAIR (CAR TEMP-TEMP))
         (SETQ POLY1 (CDR TEMP-TEMP))                           (* POLY1 is in lowest form now. *)
         (COND
            ((SETQ POLY2 (for POLY2 in (fetch (LINEAR-POT NEGATIVES) of POT) when (COMPLEMENTARY-MULTIPLEP
                                                                                    WINNING-PAIR POLY1 POLY2)
                                do (RETURN POLY2)))
             (SETQ PAIR (EQUATIONAL-PAIR-FOR WINNING-PAIR POLY1))
             (SETQ HYP1 (NUMBERP? (CAR PAIR)))
             (SETQ HYP2 (NUMBERP? (CDR PAIR)))
             (COND
                ([AND (NOT (EQUAL HYP1 FALSE))
                      (NOT (EQUAL HYP2 FALSE))
                      (for HIST-ENTRY in HIST
                         never (AND (EQ (CAR HIST-ENTRY)
                                        (QUOTE SIMPLIFY-CLAUSE))
                                    (for X in (CDDR HIST-ENTRY)
                                       thereis (AND (LISTP X)
                                                    (LISTP (CAR X))
                                                    (EQ (CAR (CAR X))
                                                        (QUOTE FIND-EQUATIONAL-POLY))
                                                    (OR (EQUAL PAIR (CDR (CAR X)))
                                                        (AND [EQUAL (CDR PAIR)
                                                                    (CAR (CDR (CAR X]
                                                             (EQUAL (CAR PAIR)
                                                                    (CDR (CDR (CAR X]
                 (SETQ LINEAR-ASSUMPTIONS (UNION-EQUAL (UNION-EQUAL (fetch (POLY ASSUMPTIONS) of POLY1)
                                                                    (fetch (POLY ASSUMPTIONS) of POLY2))
                                                       LINEAR-ASSUMPTIONS))
                 (OR (EQUAL TRUE HYP1)
                     (SETQ LINEAR-ASSUMPTIONS (ADD-TO-SET HYP1 LINEAR-ASSUMPTIONS)))
                 (OR (EQUAL TRUE HYP2)
                     (SETQ LINEAR-ASSUMPTIONS (ADD-TO-SET HYP2 LINEAR-ASSUMPTIONS)))
                 (SETQ LEMMAS-USED-BY-LINEAR (CONS (LIST (CONS (QUOTE FIND-EQUATIONAL-POLY)
                                                               PAIR))
                                                   (UNIONQ (UNIONQ (fetch (POLY LEMMAS) of POLY1)
                                                                   (fetch (POLY LEMMAS) of POLY2))
                                                           LEMMAS-USED-BY-LINEAR)))
                 (RETURN PAIR])
```

(FIRST-COEFFICIENT

```
[LAMBDA (EQUATION)                                              (* kbr: "20-Oct-85 15:53")
  (CDR (CAR (fetch (POLY ALIST) of EQUATION])
```

(FIRST-VAR

```
[LAMBDA (EQUATION)                                              (* kbr: "19-Oct-85 16:31")
  (CAAR (fetch (POLY ALIST) of EQUATION])
```

(FITS

```
[LAMBDA (ALIST1 ALIST2 VARS)                                    (* kbr: "19-Oct-85 16:31")
                                                                (* Return T iff the two alists agree on every var in VARS.
                                                                *)
   (for VAR in VARS always (EQUAL (COND
                                     ((SETQ TEMP-TEMP (ASSOC VAR ALIST1))
                                      (CDR TEMP-TEMP))
                                     (T VAR))
                                  (COND
                                     ((SETQ TEMP-TEMP (ASSOC VAR ALIST2))
```

```
                                    (CDR TEMP-TEMP))
                                   (T VAR])
```

(FIXCAR-CDR

```
[LAMBDA (TERM)                                                    (* kbr: "19-Oct-85 16:31")
  (LET (TEMP)
       [COND
          ((SETQ TEMP (CAR-CDRP (CAR TERM)))
           (SETQ TERM (CADR TERM))
           (for A-D in TEMP do (SETQ TERM (LIST (COND
                                                   ((EQ A-D (QUOTE A))
                                                    (QUOTE CAR))
                                                   (T (QUOTE CDR)))
                                                TERM]
       TERM])
```

(FLATTEN-ANDS-IN-LIT

```
[LAMBDA (TERM)                                                    (* kbr: "19-Oct-85 16:31")
  (LET (C1 C2 C3)
       (COND
          ((EQUAL TERM TRUE)
           NIL)
          [(BM-MATCH TERM (IF C1 C2 C3))
           (COND
              ((EQUAL C2 FALSE)
               (APPEND (FLATTEN-ANDS-IN-LIT (DUMB-NEGATE-LIT C1))
                       (FLATTEN-ANDS-IN-LIT C3)))
              ((EQUAL C3 FALSE)
               (APPEND (FLATTEN-ANDS-IN-LIT C1)
                       (FLATTEN-ANDS-IN-LIT C2)))
              (T (LIST TERM]
          ((BM-MATCH TERM (AND C1 C2))
           (APPEND (FLATTEN-ANDS-IN-LIT C1)
                   (FLATTEN-ANDS-IN-LIT C2)))
          (T (LIST TERM])
```

(FLESH-OUT-IND-PRIN

```
[LAMBDA (TERM FORMALS MACHINE JUSTIFICATION MASK QUICK-BLOCK-INFO)
                                                                  (* kbr: "19-Oct-85 16:31")
  QUICK-BLOCK-INFO

        (* Constructs a CANDIDATE record for TERM given, for the fn symbol of TERM, the
        FORMALS, the
        INDUCTION-MACHINE property, a JUSTIFICATION, a sound induction principle MASK, and the QUICK-BLOCK-INFO.
        *)

    (create CANDIDATE
        SCORE _ (QUOTIENT (FLOAT (for FLG in MASK count FLG))
                          (LENGTH FORMALS))
        CONTROLLERS _ (for A in (FARGS TERM) as V in FORMALS bind LOOP-ANS
                         when (MEMB V (fetch (JUSTIFICATION SUBSET) of JUSTIFICATION))
                         do (SETQ LOOP-ANS (UNIONQ (ALL-VARS A)
                                                   LOOP-ANS))
                         finally (RETURN LOOP-ANS))
        CHANGED-VARS _ (for ACTUAL in (SARGS TERM) as FLG in MASK when (EQ FLG (QUOTE CHANGEABLE)) collect ACTUAL
                          )
        UNCHANGEABLE-VARS _ (for ACTUAL in (SARGS TERM) as FLG in MASK bind LOOP-ANS
                               when (EQ FLG (QUOTE UNCHANGEABLE)) do (SETQ LOOP-ANS (UNIONQ (ALL-VARS ACTUAL
                                                                                            )
                                                                                    LOOP-ANS))
                               finally (RETURN LOOP-ANS))
        TESTS-AND-ALISTS-LST _
        [for X in MACHINE
           collect (create TESTS-AND-ALISTS
                           TESTS _ (SUB-PAIR-VAR-LST FORMALS (SARGS TERM)
                                                     (fetch (TESTS-AND-CASES TESTS) of X))
                           ALISTS _ (for ARGLIST in (fetch (TESTS-AND-CASES CASES) of X)
                                       collect (for ACTUAL in (SARGS TERM) as FLG in MASK as ARG in ARGLIST
                                                  bind LOOP-ANS
                                                  do (SETQ LOOP-ANS (UNION-EQUAL
                                                                     [COND
                                                                        ((NULL FLG)
                                                                         NIL)
                                                                        [(EQ FLG (QUOTE CHANGEABLE))
                                                                         (LIST (CONS ACTUAL (SUB-PAIR-VAR
                                                                                             FORMALS
                                                                                             (SARGS TERM)
                                                                                             ARG]
                                                                        (T (for VAR in (ALL-VARS ACTUAL)
                                                                              collect (CONS VAR VAR]
                                                                     LOOP-ANS))
                                                  finally (RETURN LOOP-ANS]
        JUSTIFICATION _ JUSTIFICATION
        INDUCTION-TERM _ TERM])
```

(FLUSH-CAND1-DOWN-CAND2

```
[LAMBDA (CAND1 CAND2)                                   (* kbr: "19-Oct-85 16:56")
  (LET (SCORE1 CONTROLLERS1 CHANGED-VARS1 UNCHANGEABLES1 TESTS-AND-ALISTS-LST1 JUSTIFICATION1 TERM1
              OTHER-TERMS1 SCORE2 CONTROLLERS2 CHANGED-VARS2 UNCHANGEABLES2 TESTS-AND-ALISTS-LST2
              JUSTIFICATION2 TERM2 OTHER-TERMS2)
       (SETQ SCORE1 (fetch (CANDIDATE SCORE) of CAND1))
       (SETQ CONTROLLERS1 (fetch (CANDIDATE CONTROLLERS) of CAND1))
       (SETQ CHANGED-VARS1 (fetch (CANDIDATE CHANGED-VARS) of CAND1))
       (SETQ UNCHANGEABLES1 (fetch (CANDIDATE UNCHANGEABLE-VARS) of CAND1))
       (SETQ TESTS-AND-ALISTS-LST1 (fetch (CANDIDATE TESTS-AND-ALISTS-LST) of CAND1))
       (SETQ JUSTIFICATION1 (fetch (CANDIDATE JUSTIFICATION) of CAND1))
       (SETQ TERM1 (fetch (CANDIDATE INDUCTION-TERM) of CAND1))
       (SETQ OTHER-TERMS1 (fetch (CANDIDATE OTHER-TERMS) of CAND1))
       (SETQ SCORE2 (fetch (CANDIDATE SCORE) of CAND2))
       (SETQ CONTROLLERS2 (fetch (CANDIDATE CONTROLLERS) of CAND2))
       (SETQ CHANGED-VARS2 (fetch (CANDIDATE CHANGED-VARS) of CAND2))
       (SETQ UNCHANGEABLES2 (fetch (CANDIDATE UNCHANGEABLE-VARS) of CAND2))
       (SETQ TESTS-AND-ALISTS-LST2 (fetch (CANDIDATE TESTS-AND-ALISTS-LST) of CAND2))
       (SETQ JUSTIFICATION2 (fetch (CANDIDATE JUSTIFICATION) of CAND2))
       (SETQ TERM2 (fetch (CANDIDATE INDUCTION-TERM) of CAND2))
       (SETQ OTHER-TERMS2 (fetch (CANDIDATE OTHER-TERMS) of CAND2))
       (COND
          [(AND (SUBSETP CHANGED-VARS1 CHANGED-VARS2)
                (SUBSETP UNCHANGEABLES1 UNCHANGEABLES2)
                (PIGEON-HOLE TESTS-AND-ALISTS-LST1 TESTS-AND-ALISTS-LST2
                       [FUNCTION (LAMBDA (TA1 TA2)
                                  (AND (SUBSETP (fetch (TESTS-AND-ALISTS TESTS) of TA1)
                                                (fetch (TESTS-AND-ALISTS TESTS) of TA2))
                                       (OR (AND (NULL (fetch (TESTS-AND-ALISTS ALISTS) of TA1))
                                                (NULL (fetch (TESTS-AND-ALISTS ALISTS) of TA2)))
                                           (PIGEON-HOLE (fetch (TESTS-AND-ALISTS ALISTS) of TA1)
                                                  (fetch (TESTS-AND-ALISTS ALISTS) of TA2)
                                                  [FUNCTION (LAMBDA (ALIST1 ALIST2)
                                                              (PIGEON-HOLE ALIST1 ALIST2
                                                                     [FUNCTION (LAMBDA (PAIR1 PAIR2)
                                                                                 (AND (EQ (CAR PAIR1)
                                                                                          (CAR PAIR2))
                                                                                      (OCCUR (CDR PAIR1)
                                                                                             (CDR PAIR2]
                                                                     T T]
                                                  T T]
                       T T))
           (create CANDIDATE
                  SCORE _ (PLUS SCORE1 SCORE2)
                  CONTROLLERS _ (UNIONQ CONTROLLERS1 CONTROLLERS2)
                  CHANGED-VARS _ CHANGED-VARS2
                  UNCHANGEABLE-VARS _ UNCHANGEABLES2
                  TESTS-AND-ALISTS-LST _ TESTS-AND-ALISTS-LST2
                  JUSTIFICATION _ JUSTIFICATION2
                  INDUCTION-TERM _ TERM2
                  OTHER-TERMS _ (ADD-TO-SET TERM1 (UNION-EQUAL OTHER-TERMS1 OTHER-TERMS2]
          (T NIL])
```

(FN-SYMB0

```
[LAMBDA (X)                                             (* kbr: "20-Oct-85 13:49")
                                                        (* What type of constant is (QUOTE X) ? *)
  (COND
     [(LITATOM X)
      (COND
         ((EQ X *1*T)
          (QUOTE TRUE))
         ((EQ X *1*F)
          (QUOTE FALSE))
         (T (QUOTE PACK]
     [(FIXP X)
      (COND
         ((LESSP X 0)
          (QUOTE MINUS))
         ((EQUAL X 0)
          (QUOTE ZERO))
         (T (QUOTE ADD1]
     ((EQ (CAR X)
          *1*SHELL-QUOTE-MARK)
      (CADR X))
     (T (QUOTE CONS])
```

(FNNAMEP

```
[LAMBDA (FN TERM)                                       (* kbr: "19-Oct-85 16:31")
  (COND
     ((VARIABLEP TERM)
      NIL)
     ((FQUOTEP TERM)
      (COND
         ((OR (MEMB FN *1*BTM-OBJECTS)
```

```
               (ASSOC FN SHELL-ALIST))
           (MEMB FN (ALL-FNNAMES TERM)))
          (T NIL)))
       ((EQ FN (FFN-SYMB TERM))
        T)
       (T (for X in (FARGS TERM) thereis (FNNAMEP FN X])
```

(FNNAMEP-IF

```
[LAMBDA (TERM)                                          (* kbr: "19-Oct-85 16:31")
  (COND
     ((VARIABLEP TERM)
      NIL)
     ((FQUOTEP TERM)
      NIL)
     ((EQ (FFN-SYMB TERM)
          (QUOTE IF))
      T)
     (T (for X in (FARGS TERM) thereis (FNNAMEP-IF X])
```

(FORM-COUNT

```
[LAMBDA (TERM)                                          (* kbr: "19-Oct-85 16:31")

        (* Returns the number of open parentheses in the unabbreviated presentation of TERM.
        Also sets NUMBER-OF-VARIABLES to the number of variables in TERM.
        *)

  (SETQ NUMBER-OF-VARIABLES 0)
  (FORM-COUNT1 TERM])
```

(FORM-COUNT-EVG

```
[LAMBDA (EVG)                                           (* kbr: "20-Oct-85 15:59")
  (COND
     [(NLISTP EVG)
      (COND
         ((EQ EVG *1*T)
          1)
         ((EQ EVG *1*F)
          1)
         [(FIXP EVG)
          (COND
             ((LESSP EVG 0)
              (PLUS 2 (MINUS EVG)))
             (T (ADD1 EVG]
         (T (SETQ TEMP-TEMP (ASSOC EVG LITATOM-FORM-COUNT-ALIST))
            (COND
               (TEMP-TEMP (CDR TEMP-TEMP))
               (T (SETQ LITATOM-FORM-COUNT-ALIST (CONS [CONS EVG (PLUS 2 (TIMES 2 (NCHARS EVG))
                                                                        (for I NUMBER from 1
                                                                           to (NCHARS EVG)
                                                                           sum (NTHCHARCODE EVG I]
                                                       LITATOM-FORM-COUNT-ALIST))
                  (CDR (CAR LITATOM-FORM-COUNT-ALIST]
     [(EQ (CAR EVG)
          *1*SHELL-QUOTE-MARK)
      (ADD1 (for X in (CDDR EVG) sum (FORM-COUNT-EVG X]
     (T (PLUS 1 (FORM-COUNT-EVG (CAR EVG))
                (FORM-COUNT-EVG (CDR EVG])
```

(FORM-COUNT1

```
[LAMBDA (TERM)                                          (* kbr: "19-Oct-85 16:31")
  (COND
     ((VARIABLEP TERM)
      (SETQ NUMBER-OF-VARIABLES (ADD1 NUMBER-OF-VARIABLES))
      0)
     ((FQUOTEP TERM)
      (FORM-COUNT-EVG (CADR TERM)))
     (T (ADD1 (for T1 in (FARGS TERM) sum (FORM-COUNT1 T1])
```

(FORM-INDUCTION-CLAUSE

```
[LAMBDA (TESTS HYPS CONCL TERMS)                        (* kbr: "19-Oct-85 16:31")
  TERMS
        (* We once implemented the idea of both induction, opening up of the recursive fns in the conclusion, and generalizing away
        some recursive calls. This function did the expansion and generalization.
        If the idea is reconsidered the following theorems are worthy of consideration:
        (ORDERED (SORT X)), (IMPLIES (ORDERED X) (ORDERED
        (ADDTOLIST I X))), (IMPLIES (AND (NUMBER-LISTP X) (ORDERED X)
        (NUMBERP I) (NOT (LESSP (CAR X) I))) (EQUAL (ADDTOLIST I X)
        (CONS I X))), and (IMPLIES (AND (NUMBER-LISTP X) (ORDERED X))
        (EQUAL (SORT X) X)) . *)

  (APPEND TESTS HYPS CONCL])
```

(FORMP-SIMPLIFIER

```
[LAMBDA (TERM)                                                    (* kbr: "19-Oct-85 16:31")
  (LET (X FN TL)
       (MATCH! TERM (FORMP X))
       (COND
          ((VARIABLEP X)
           TERM)
          [(SHELLP X)
           (COND
              ((NEQ (FN-SYMB X)
                    (QUOTE CONS))
               (CONS-TERM (QUOTE LITATOM)
                      (FARGS TERM)))
              (T (SETQ FN (ARGN X 1))
                 (SETQ TL (ARGN X 2))
                 (COND
                    ((AND (QUOTEP FN)
                          (LITATOM (CADR FN)))
                     (COND
                        [(EQ (CADR FN)
                             (QUOTE QUOTE))
                         (BM-SUBST TL (QUOTE TL)
                                (QUOTE (IF (LISTP TL)
                                           (EQUAL (CDR TL)
                                                  (QUOTE NIL))
                                           (QUOTE *1*FALSE]
                        [(AND (GETPROP (CADR FN)
                                       (QUOTE TYPE-PRESCRIPTION-LST))
                              (NOT (MEMB (CADR FN)
                                         META-NAMES)))
                         (SUBLIS-VAR [LIST (CONS (QUOTE TL)
                                                 TL)
                                           (CONS (QUOTE A)
                                                 (LIST (QUOTE QUOTE)
                                                       (ARITY (CADR FN]
                                (QUOTE (IF (EQUAL A (LENGTH TL))
                                           (FORM-LSTP TL)
                                           (QUOTE *1*FALSE]
                        (T TERM)))
                    (T TERM]
          (T TERM])
```

(FORMULA-OF

```
[LAMBDA (NAME)                                                    (* kbr: "19-Oct-85 16:31")
  (LET (TEMP)
       (SETQ TEMP (GETPROP NAME (QUOTE EVENT)))
       (SELECTQ (CAR TEMP)
            ((ADD-AXIOM PROVE-LEMMA)
                 (CADDDR TEMP))
            (NIL])
```

(FREE-VAR-CHK

```
[LAMBDA (NAME ARGS FORM)                                          (* kbr: "19-Oct-85 16:31")
  (LET (TEMP)
       (SETQ FORM (ALL-VARS FORM))
       (SETQ TEMP (SET-DIFF FORM ARGS))
       [COND
          (TEMP (ERROR1 (PQUOTE (PROGN ILLEGAL FREE (PLURAL? TEMP VARIABLES VARIABLE)
                                       ,
                                       (!PPR-LIST TEMP)
                                       , IN THE DEFINITION OF (!PPR NAME NIL) !))
                        (BINDINGS (QUOTE NAME)
                               NAME
                               (QUOTE TEMP)
                               TEMP)
                        (QUOTE SOFT]
       (SETQ TEMP (SET-DIFF ARGS FORM))
       [COND
          (TEMP (ERROR1 (PQUOTE (PROGN (!LIST TEMP)
                                       (PLURAL? TEMP ARE IS)
                                       IN THE ARGLIST BUT NOT IN THE BODY OF THE DEFINITION OF
                                       (!PPR NAME NIL) !))
                        (BINDINGS (QUOTE NAME)
                               NAME
                               (QUOTE TEMP)
                               TEMP)
                        (QUOTE WARNING]
       NIL])
```

(FREE-VARSP

```
[LAMBDA (TERM ALIST)                                              (* kbr: "19-Oct-85 16:31")
  (COND
     ((VARIABLEP TERM)
```

```
        (NOT (ASSOC TERM ALIST)))
       ((FQUOTEP TERM)
        NIL)
       (T (for ARG in (FARGS TERM) thereis (FREE-VARSP ARG ALIST])
```

(GEN-VARS

```
  [LAMBDA (CL N VARIABLE-NAMES)                                    (* kbr: "19-Oct-85 16:31")
                                                                   (* Generates N skolem constants not occurring in clause CL.
                                                                   *)
    (SET-DIFF-N VARIABLE-NAMES (for LIT in CL bind LOOP-ANS do (SETQ LOOP-ANS (UNIONQ (ALL-VARS LIT)
                                                                                      LOOP-ANS))
                                    finally (RETURN LOOP-ANS))
          N])
```

(GENERALIZE-CLAUSE

```
  [LAMBDA (CL HIST)                                                (* kbr: "19-Oct-85 16:31")
    HIST
          (* Generalize the smallest common subterms in CL -- as defined by COMSUBTERMS --
          using the lemmas on GENERALIZE-LEMMAS to supply typing info.
          *)

    (PROG (COMMONSUBTERMS)
          (COND
             ((NOT (ASSOC (QUOTE BEING-PROVED)
                          STACK))
              (RETURN NIL)))
          (SETQ COMMONSUBTERMS (GENRLTERMS CL))
          (COND
             ((NULL COMMONSUBTERMS)
              (RETURN NIL))
             (T (SETQ PROCESS-CLAUSES (LIST (GENERALIZE1 CL COMMONSUBTERMS GEN-VARIABLE-NAMES1)))
                (SETQ PROCESS-HIST (LIST GENERALIZING-SKOS COMMONSUBTERMS OBVIOUS-RESTRICTIONS
                                         GENERALIZE-LEMMA-NAMES))
                (SETQ ALL-LEMMAS-USED (UNIONQ GENERALIZE-LEMMA-NAMES ALL-LEMMAS-USED))
                (RETURN T])
```

(GENERALIZE-SENT

```
  [LAMBDA (CL HIST)                                                (* kbr: "19-Oct-85 16:31")
    (EXECUTE (QUOTE GENERALIZE-CLAUSE)
           CL HIST (QUOTE SIMPLIFY-SENT)
           (QUOTE ELIMINATE-IRRELEVANCE-SENT])
```

(GENERALIZE1

```
  [LAMBDA (CL SUBTERMLST VARIABLE-NAMES)                           (* kbr: "19-Oct-85 16:31")

          (* Replaces all occurrences of the subterms in SUBTERMLST in CL by new vars, and qualifies each var with all the
          information known to GET-TYPES. *)

    (SETQ GENERALIZING-SKOS (GEN-VARS CL (LENGTH SUBTERMLST)
                                      VARIABLE-NAMES))
    (SETQ OBVIOUS-RESTRICTIONS NIL)
    (SETQ GENERALIZE-LEMMA-NAMES NIL)
    (GENERALIZE2 SUBTERMLST GENERALIZING-SKOS CL])
```

(GENERALIZE2

```
  [LAMBDA (TERMLST VARLST CL)                                      (* kbr: "19-Oct-85 20:00")
    (for LIT in (SCRUNCH (NCONC (for SUBTERM in TERMLST join (for HYP in (GET-TYPES SUBTERM CL)
                                                                    collect (DUMB-NEGATE-LIT HYP)))
                                CL))
       collect (SUB-PAIR-EXPR TERMLST VARLST LIT])
```

(GENRLT1

```
  [LAMBDA (CL)                                                     (* kbr: "19-Oct-85 16:31")
    (LET (LHS RHS)
         (for LIT in CL when [OR (BM-MATCH LIT (EQUAL LHS RHS))
                                 (BM-MATCH LIT (NOT (EQUAL LHS RHS]
            do (COMSUBTERMS LHS RHS))
         (for TAIL on CL do (for LIT2 in (CDR TAIL) do (COMSUBTERMS (CAR TAIL)
                                                                    LIT2)))
         NIL])
```

(GENRLTERMS

```
  [LAMBDA (CL)                                                     (* kbr: "19-Oct-85 16:31")
    (LET (GENRLTLIST)
         (GENRLT1 CL)
         GENRLTLIST])
```

(GET-CANDS

```
  [LAMBDA (TERM)                                                   (* kbr: "19-Oct-85 20:06")
```

```
        (* Returns all of the induction principles -- see POSSIBLE-IND-PRINCIPLES --
        connected to terms in TERM, which is the conjecture to be proved.
        *)

    (COND
       ((VARIABLEP TERM)
        NIL)
       ((QUOTEP TERM)
        NIL)
       (T (NCONC (POSSIBLE-IND-PRINCIPLES TERM)
                 (for ARG in (FARGS TERM) join (GET-CANDS ARG])
```

(GET-LISP-SEXPR

```
  [LAMBDA (FN)                                                    (* kbr: "19-Oct-85 16:31")
    (LET (SEXPR)
         [COND
            ((NULL (GETPROP FN (QUOTE LISP-CODE)))
             (ERROR1 (PQUOTE (PROGN (!PPR FN NIL)
                                    DOES NOT HAVE A RUNNABLE LISP DEFINITION %.))
                     (BINDINGS (QUOTE FN)
                               FN)
                     (QUOTE SOFT]
         (SETQ SEXPR (GETPROP (GETPROP FN (QUOTE LISP-CODE))
                              (QUOTE SEXPR)))
         (COND
            ((NULL SEXPR)
             (ERROR1 (PQUOTE (PROGN (!PPR FN NIL)
                                    IS PART OF THE BASIC SYSTEM AND HAS A HAND-CODED LISP DEFINITION %.))
                     (BINDINGS (QUOTE FN)
                               FN)
                     (QUOTE SOFT)))
            (T SEXPR])
```

(GET-LEVEL-NO

```
  [LAMBDA (FNNAME)                                                (* kbr: "19-Oct-85 16:31")
    (OR (GETPROP FNNAME (QUOTE LEVEL-NO))
        0])
```

(GET-STACK-NAME

```
  [LAMBDA (STACKV)                                                (* kbr: "26-Oct-85 13:59")
    (PACK (CONS (QUOTE *)
                (CDR (for I in (DREVERSE (GET-STACK-NAME1 STACKV)) join (CONS (QUOTE %.)
                                                                              (UNPACK I])
```

(GET-STACK-NAME1

```
  [LAMBDA (STACKV)                                                (* kbr: "19-Oct-85 16:31")
    (LET (ANS)
         (COND
            ((NULL STACKV)
             (LIST 1))
            [(EQ (CAAR STACKV)
                 (QUOTE TO-BE-PROVED))
             (SETQ ANS (GET-STACK-NAME1 (CDR STACKV)))
             (RPLACA ANS (ADD1 (CAR ANS]
            (T (CONS 1 (GET-STACK-NAME1 (CDR STACKV])
```

(GET-TYPES

```
  [LAMBDA (TERM CL)                                               (* kbr: "19-Oct-85 16:31")
    (LET (TYPE-RESTRICTION LEMMA-RESTRICTIONS TYPE PAIR INST-LEMMA)
         CL
         (SETQ TYPE (TYPE-SET TERM))
         (SETQ TYPE-RESTRICTION (COND
                                   ((SETQ PAIR (for PAIR in RECOGNIZER-ALIST when (IEQP TYPE (CDR PAIR))
                                                  do (RETURN PAIR)))
                                    (FCONS-TERM* (CAR PAIR)
                                           TERM))
                                   (T NIL)))
         [COND
            (TYPE-RESTRICTION (SETQ OBVIOUS-RESTRICTIONS (ADD-TO-SET TYPE-RESTRICTION OBVIOUS-RESTRICTIONS]
         (SETQ LEMMA-RESTRICTIONS (for LEMMA in GENERALIZE-LEMMAS unless (DISABLEDP (fetch (GENERALIZE-LEMMA NAME)
                                                                                       of LEMMA))
                                     when [AND (ARG1-IN-ARG2-UNIFY-SUBST TERM (fetch (GENERALIZE-LEMMA TERM)
                                                                                 of LEMMA))
                                               (NOT (FREE-VARSP (fetch (GENERALIZE-LEMMA TERM) of LEMMA)
                                                          UNIFY-SUBST))
                                               (NOT (FNNAMEP (FN-SYMB TERM)
                                                             (SUBST-EXPR (QUOTE X)
                                                                    TERM
                                                                    (SETQ INST-LEMMA (SUBLIS-VAR
                                                                                       UNIFY-SUBST
                                                                                       (fetch (GENERALIZE-LEMMA
```

```
                                                                                    TERM)
                                                                              of LEMMA]
                                                  collect (PROGN (SETQ GENERALIZE-LEMMA-NAMES (CONS (fetch (GENERALIZE-LEMMA
                                                                                                       NAME)
                                                                                                 of LEMMA)
                                                                                          GENERALIZE-LEMMA-NAMES))
                                                                        INST-LEMMA)))
        (COND
           (TYPE-RESTRICTION (CONS TYPE-RESTRICTION LEMMA-RESTRICTIONS))
           (T LEMMA-RESTRICTIONS])
```

(GREATEREQP

```
  [LAMBDA (I J)                                                          (* kbr: "19-Oct-85 16:31")
    (NOT (LESSP I J])
```

(GUARANTEE-CITIZENSHIP

```
  [LAMBDA (NAME)                                                         (* kbr: "19-Oct-85 16:31")
    (COND
       ([AND (NOT (GETPROP NAME (QUOTE EVENT)))
             (NOT (GETPROP NAME (QUOTE MAIN-EVENT]
        (PUT1 MAIN-EVENT-NAME (CONS NAME (GETPROP MAIN-EVENT-NAME (QUOTE SATELLITES)))
              (QUOTE SATELLITES))
        (PUT1 NAME MAIN-EVENT-NAME (QUOTE MAIN-EVENT])
```

(GUESS-RELATION-MEASURE-LST

```
  [LAMBDA (FORMALS MACHINE)                                              (* kbr: "19-Oct-85 16:31")
```

(* We assume MACHINE is a list of TESTS-AND-CASE. We will guess that the COUNT goes down with LESSP on formal tested and changed in every line of the machine. *)

```
    (for VAR in FORMALS as I from 0 when [for X in MACHINE always (AND (OCCUR-LST VAR (fetch (TESTS-AND-CASE TESTS)
                                                                                        of X))
                                                                       (NEQ VAR (BM-NTH I (fetch (TESTS-AND-CASE
                                                                                                  CASE)
                                                                                            of X]
       collect (LIST (QUOTE LESSP)
                     (LIST (QUOTE COUNT)
                           VAR])
```

(HAS-LIB-PROPS

```
  [LAMBDA (ATM)                                                          (* kbr: "19-Oct-85 16:31")
    (for TAIL on (GETPROPLIST ATM) by (QUOTE CDDR) thereis (AND (MEMB (CAR TAIL)
                                                                      LIB-PROPS)
                                                                (CADR TAIL])
```

(ILLEGAL-CALL

```
  [LAMBDA NIL                                                            (* kbr: "19-Oct-85 16:31")
    (ERROR1 (PQUOTE (PROGN SOME FUNCTION WAS CALLED WITH INAPPROPRIATE ARGUMENTS %.))
            NIL
            (QUOTE HARD])
```

(ILLEGAL-NAME

```
  [LAMBDA (NAME)                                                         (* kbr: "19-Oct-85 16:31")
    (NOT (AND (LITATOM NAME)
              (LITATOM NAME)
              (LEGAL-CHAR-CODE-SEQ (CHCON NAME])
```

(IMMEDIATE-DEPENDENTS-OF

```
  [LAMBDA (NAME)                                                         (* kbr: "19-Oct-85 16:31")
    (LET (ATM)
         (COND
            ((EQ NAME (QUOTE GROUND-ZERO))
             (REMOVE1 (QUOTE GROUND-ZERO)
                      CHRONOLOGY))
            ((NOT (GETPROP NAME (QUOTE EVENT)))
             (ERROR1 (PQUOTE (PROGN IMMEDIATE-DEPENDENTS-OF WAS CALLED ON A NONEVENT , (!PPR NAME NIL)
                                    %.))
                     (BINDINGS (QUOTE NAME)
                               NAME)
                     (QUOTE HARD)))
            ((SETQ ATM (TYPE-PRESCRIPTION-LEMMAP NAME))
```

(* NAME is a type prescription lemma hung under ATM. In this case, we must include in the dependents of NAME all events dependent upon ATM that occurred after NAME was introduced.
This clause in the UNDO mechanism is the source of doubt that the mechanism correctly identifies all of the dependents of an event. The problem starts with the fact that the use of type set lemmas is not tracked like other lemmas.
In fact, no code in the theorem prover actually notes when or how a particular type set lemma is used.
How then can we hope to determine which proofs (or other events) depend upon a type set lemma? We have tried several approaches to the question. Some have turned out incorrect. We believe the current one to be correct.
Our hand-waving proof of its correctness is this. If a type set lemma about the function FN is used in the proof of THM, then either (1) THM mentions FN, (2) some lemma used in the proof of THM

(other than a type set lemma) mentions FN, (3) some lemma used in the proof of THM mentions a function whose definition mentions FN, (3.a) some lemma used in the proof of THM uses a function whose definition mentions a function that either (3.b) mentions FN or (3.c) mentions a function whose definition mentions FN, or ...
But we believe that any such lemma introducing FN into the proof is in ALL-LEMMAS-USED when the proof is done and thus has THM as one of its IMMEDIATE-DEPENDENTS0. To put it in terms of the following code, we believe that there is a an IMMEDIATE-DEPENDENTS0 path, from FN to THM. Given that hypothesis, we then correctly identify a superset of the dependents of a type set lemma by the draconian strategy of claiming as a dependent event any event on a tree-path that took place later than the type set lemma. Note that this computation is not trying to get all of the theorems dependent (somehow) upon the type set lemma in question but only those immediately dependent --
i.e., whose proofs might have actually appealed to this type set lemma.
It is assumed that any function using IMMEDIATE-DEPENDENTS-OF to explore the logical graph of events will recurse on each of the dependent events, and thus catch things like THMs dependent upon type set lemmas dependent upon the type set lemma in question. *)

```
(UNION-EQUAL (for X in (TREE-DEPENDENTS (MAIN-EVENT-OF ATM)) when (EVENT1-OCCURRED-BEFORE-EVENT2
                                                                    NAME X CHRONOLOGY)
                    collect X)
             (for X in (GETPROP NAME (QUOTE IMMEDIATE-DEPENDENTS0)) collect X)))
      (T (for X in (GETPROP NAME (QUOTE IMMEDIATE-DEPENDENTS0)) collect X])
```

(IMPLIES?

```
[LAMBDA (TESTS TERM)                                                    (* kbr: "19-Oct-85 16:31")
  (MEMBER TERM TESTS])
```

(IMPOSSIBLE-POLYP

```
[LAMBDA (POLY)                                                          (* kbr: "19-Oct-85 16:31")
  (AND (GREATERP (fetch (POLY CONSTANT) of POLY)
                 0)
       (for PAIR in (fetch (POLY ALIST) of POLY) always (GREATEREQP (CDR PAIR)
                                                                    0])
```

(IND-FORMULA

```
[LAMBDA (TESTS-AND-ALISTS-LST TERMS CL-SET)                             (* kbr: "19-Oct-85 20:08")
```

(* TESTS-AND-ALISTS-LST is a such a list that the disjunction of the conjunctions of the TESTS components of the members is T. Furthermore, there exists a measure M, a well-founded relation R, and a sequence of variables x1, ..., xn such that for each T&Ai in TESTS-AND-ALISTS-LST, for each alist alst in the ALISTS component of T&Ai, the conjunction of the TESTS component, say qi, implies that (R (M x1 ... xn) /alst
(M x1 ... xn))%. To prove thm, the conjunction of the disjunctions of the members of CL-SET, it is sufficient, by the principle of induction, to prove instead the conjunction of the terms qi & thm' & thm'' ...
-> thm, where the primed terms are the results of substituting the alists in the ALISTS field of the ith member of TESTS-AND-ALISTS-LST into thm. If thm1, thm2, ..., thmn are the disjunctions of the members of CL-SET, then it is sufficient to prove all of the formulas qi & thm' & thm'' ... -> thmj.
This is a trivial proposition fact, to prove (IMPLIES A (AND B C)) it is sufficient to prove
(IMPLIES A B) and (IMPLIES A C) The (FOR PICK ...) expression below returns a list of clauses whose conjunction propositionally implies qi & thm' & thm'' ... -> thmj, where TA is the ith member of TESTS-AND-ALISTS-LST and CL is the jth member of CL-SET. Proof: Let THM have the form: (AND (OR a1 ...)
(OR b1 ...) ... (OR z1 ...))%. Then qi & thm' & thm'' ... -> thmj has the form:
(IMPLIES (AND qi (AND (OR a1 ...) (OR b1 ...) ... (OR z1 ...)) (QUOTE
(AND (OR a1 ...) (OR b1 ...) ... (OR z1 ...))) (QUOTE (QUOTE ...))
(AND (OR a1 ...) (OR b1 ...) ... (OR z1 ...))) (QUOTE (QUOTE (QUOTE ...'))) thmj)%.
Suppose this formula is false for some values of the free variables.
Then under those values, each disjunction in the hypothesis is true.
Thus there exists a way of choosing one literal from each of the disjunctions, all of which are true.
This choice is one of the PICKs below. But we prove that (IMPLIES
(AND qi PICK) thmj) . *)

(DELETE-TAUTOLOGIES

```
(SCRUNCH-CLAUSE-SET
  (for CL in CL-SET
     join (for TA in TESTS-AND-ALISTS-LST
             join (for PICK in [ALL-PICKS (for CL1 in CL-SET
                                             join (for ALIST in (fetch (TESTS-AND-ALISTS ALISTS) of TA)
                                                     collect (for LIT in CL1 collect (NEGATE-LIT (SUBLIS-VAR ALIST
                                                                                                             LIT]
                                collect (FORM-INDUCTION-CLAUSE (for TEST in (fetch (TESTS-AND-ALISTS TESTS) of TA)
                                                                  collect (NEGATE-LIT TEST))
                                        PICK CL TERMS])
```

(INDUCT

```
[LAMBDA (CL-SET)                                                        (* kbr: "19-Oct-85 20:09")
  (LET (GET-CANDS-ANS MERGED-CANDS-ANS PICK-HIGH-SCORES-ANS WINNING-CAND INDUCT-ANS COMPUTE-VETOES-ANS
        FAVOR-COMPLICATED-CANDIDATES-ANS)
       [SETQ WINNING-CAND
        (CAR (SETQ PICK-HIGH-SCORES-ANS
              (PICK-HIGH-SCORES (SETQ FAVOR-COMPLICATED-CANDIDATES-ANS
                                 (FAVOR-COMPLICATED-CANDIDATES
                                  (SETQ COMPUTE-VETOES-ANS
                                   (COMPUTE-VETOES (SETQ MERGED-CANDS-ANS
                                                    (TRANSITIVE-CLOSURE
                                                     (SETQ GET-CANDS-ANS
                                                      (REMOVE-UNCHANGING-VARS
                                                       (for CL in CL-SET
```

```
                                                                    join (for LIT in CL join (GET-CANDS LIT)))
                                                                  CL-SET))
                                                                (FUNCTION MERGE-CANDS]
        (COND
           [WINNING-CAND (SETQ INDUCT-ANS (IND-FORMULA (fetch (CANDIDATE TESTS-AND-ALISTS-LST) of WINNING-CAND)
                                                       (CONS (fetch (CANDIDATE INDUCTION-TERM) of WINNING-CAND)
                                                             (fetch (CANDIDATE OTHER-TERMS) of WINNING-CAND))
                                                       CL-SET))
                         (INFORM-SIMPLIFY (fetch (CANDIDATE TESTS-AND-ALISTS-LST) of WINNING-CAND)
                                (CONS (fetch (CANDIDATE INDUCTION-TERM) of WINNING-CAND)
                                      (fetch (CANDIDATE OTHER-TERMS) of WINNING-CAND]
           (T (IO (QUOTE INDUCT)
                  CL-SET NIL (LIST NIL)
                  (LIST (GET-STACK-NAME (CDR STACK))
                        NIL 0 0 0 0 0))
              (WRAPUP NIL)))
        (SETQ ALL-LEMMAS-USED (UNIONQ (fetch (JUSTIFICATION LEMMAS) of (fetch (CANDIDATE JUSTIFICATION)
                                                                               of WINNING-CAND))
                                      ALL-LEMMAS-USED))
        (IO (QUOTE INDUCT)
            CL-SET NIL INDUCT-ANS (LIST (GET-STACK-NAME (CDR STACK))
                                        WINNING-CAND
                                        (LENGTH GET-CANDS-ANS)
                                        (LENGTH MERGED-CANDS-ANS)
                                        (COND
                                           ((EQ COMPUTE-VETOES-ANS MERGED-CANDS-ANS)
                                            0)
                                           (T (LENGTH COMPUTE-VETOES-ANS)))
                                        (LENGTH PICK-HIGH-SCORES-ANS)
                                        (LENGTH FAVOR-COMPLICATED-CANDIDATES-ANS)))
        INDUCT-ANS])
```

(INDUCT-VARS

```
[LAMBDA (CAND)                                                  (* kbr: "19-Oct-85 16:31")
```

(* Get all skos occupying controller slots in any of the terms associated with this candidate.
*)

```
   (for TERM in (CONS (fetch (CANDIDATE INDUCTION-TERM) of CAND)
                      (fetch (CANDIDATE OTHER-TERMS) of CAND))
      bind LOOP-ANS
      do (SETQ LOOP-ANS (UNIONQ (for ARG in (FARGS TERM) as I from 0
                                   when [AND (VARIABLEP ARG)
                                             (for MASK in (GETPROP (FFN-SYMB TERM)
                                                                   (QUOTE CONTROLLER-POCKETS))
                                                thereis (NOT (IEQP 0 (LOGAND 1 (LSH MASK (MINUS I]
                                   collect ARG)
                                LOOP-ANS))
      finally (RETURN LOOP-ANS])
```

(INDUCTION-MACHINE

```
[LAMBDA (FNNAME TERM TESTS)                                     (* kbr: "24-Oct-85 14:57")
                                                                (* See the comment for TERMINATION-MACHINE.
                                                                *)
   (COND
      [(OR (VARIABLEP TERM)
           (FQUOTEP TERM)
           (NEQ (FFN-SYMB TERM)
                (QUOTE IF)))
       (LIST (create TESTS-AND-CASES
                     TESTS _ (REMOVE-REDUNDANT-TESTS TESTS NIL)
                     CASES _ (UNION-EQUAL (PROG (LOOP-ANS)
                                                (for TEST in TESTS do (SETQ LOOP-ANS (UNION-EQUAL (ALL-ARGLISTS
                                                                                                    FNNAME TEST)
                                                                                                  LOOP-ANS)))
                                                (RETURN LOOP-ANS))
                                          (ALL-ARGLISTS FNNAME TERM]
      (T (NCONC [INDUCTION-MACHINE FNNAME (FARGN TERM 2)
                                   (APPEND TESTS (LIST (FARGN TERM 1]
                [INDUCTION-MACHINE FNNAME (FARGN TERM 3)
                                   (APPEND TESTS (LIST (NEGATE-LIT (FARGN TERM 1])
```

(INFORM-SIMPLIFY

```
[LAMBDA (TESTS-AND-ALISTS-LST TERMS)                            (* kbr: "19-Oct-85 20:10")
```

(* Two of the variables effecting REWRITE are TERMS-TO-BE-IGNORED-BY-REWRITE and EXPAND-LST.
When any term on the former is encountered REWRITE returns it without rewriting it.
Terms on the latter must be calls of defined fns and when encountered are replaced by the rewritten body.
We believe that the theorem prover will perform significantly faster on many theorems if, after an induction, it does not waste time (a) trying to simplify the recursive calls introduced in the induction hypotheses and
(b) trying to decide whether to expand the terms inducted for in the induction conclusion.
This suspicion is due to some testing done with the idea of generalizing the recursive calls away at INDUCT time after expanding the induction terms in the conclusion. Homographication speeded the theorem-prover on many theorems but lost on several others because of the premature generalization. See the comment in FORM-INDUCTION-CLAUSE.
To avoid the generalization at INDUCT time we are going to try using TERMS-TO-BE-IGNORED-BY-REWRITE.

The idea is this, during the initial simplification of a clause produced by INDUCT we will have the recursive terms on TERMS-TO-BE-IGNORED-BY-REWRITE. When the clause settles down -- hopefully it will often be proved first -- we will restore TERMS-TO-BE-IGNORED-BY-REWRITE to its pre-INDUCT value. Note however that we have to mess with TERMS-TO-BE-IGNORED-BY-REWRITE on a clause by clause basis, not just once in INDUCT. So here is the plan. INDUCT will set INDUCTION-HYP-TERMS to the list of instances of the induction terms, and will set INDUCTION-CONCL-TERMS to the induction terms themselves.
SIMPLIFY-CLAUSE will look at the history of the clause to determine whether it has settled down since induction.
If not it will bind TERMS-TO-BE-IGNORED-BY-REWRITE to the concatenation of INDUCTION-HYP-TERMS and its old value and will analogously bind EXPAND-LST. A new process, called SETTLED-DOWN-SENT, will be used to mark when in the history the clause settled down. *)

```
(SETQ INDUCTION-CONCL-TERMS TERMS)
(SETQ INDUCTION-HYP-TERMS (for TA in TESTS-AND-ALISTS-LST join (for ALIST in (fetch (TESTS-AND-ALISTS ALISTS)
                                                                                    of TA)
                                                                          join (SUBLIS-VAR-LST ALIST TERMS])
```

(INIT-LEMMA-STACK

```
[LAMBDA NIL                                                    (* kbr: "19-Oct-85 16:31")
  (SETQ LEMMA-STACK ORIG-LEMMA-STACK)
  NIL])
```

(INIT-LIB

```
[LAMBDA (PROPS VARS)                                           (* kbr: "19-Oct-85 16:31")
```

(* Initialize the variables used to keep track of what is on the lib file.
*)

```
(KILL-LIB)
(SETQ LIB-PROPS PROPS)
(SETQ LIB-VARS VARS)
(for VAR in LIB-VARS do (SET VAR NIL))
(SETQ LIB-ATOMS-WITH-PROPS NIL)
(SETQ LIB-ATOMS-WITH-DEFS NIL)
(SETQ LIB-FILE NIL])
```

(INIT-LINEARIZE-ASSUMPTIONS-STACK

```
[LAMBDA NIL                                                    (* kbr: "19-Oct-85 16:31")
  (SETQ LINEARIZE-ASSUMPTIONS-STACK ORIG-LINEARIZE-ASSUMPTIONS-STACK)
  NIL])
```

(INTERESTING-SUBTERMS

```
[LAMBDA (FORM)                                                 (* kbr: "19-Oct-85 16:31")
```

(* Returns a list of all of the subterms of FORM that are not variables or quotes or terms whose function symbol is CAR CDR LISTP EQ NEQ NOT. Returns the EQ subterms. This fact is used to catch and optimize common subexpression evaluation. *)

```
(COND
   ((VARIABLEP FORM)
    NIL)
   ((FQUOTEP FORM)
    NIL)
   [(MEMB (FFN-SYMB FORM)
          (QUOTE (CAR CDR LISTP EQ NEQ NOT)))
    (for ARG in (FARGS FORM) do (APPEND (INTERESTING-SUBTERMS ARG]
   (T (CONS FORM (for ARG in (FARGS FORM) do (APPEND (INTERESTING-SUBTERMS ARG])
```

(INTERSECTP

```
[LAMBDA (X Y)                                                  (* kbr: "19-Oct-85 16:31")
  (for E in X thereis (MEMBER E Y])
```

(INTRODUCE-ANDS

```
[LAMBDA (X)                                                    (* kbr: "19-Oct-85 16:31")
  (LET (REST1 REST2)
       (COND
          ((NLISTP X)
           X)
          ((EQ (CAR X)
               (QUOTE QUOTE))
           X)
          [(BM-MATCH X (*2*IF & & (QUOTE NIL)))
           (SETQ REST1 (INTRODUCE-ANDS (CADR X)))
           (SETQ REST2 (INTRODUCE-ANDS (CADDR X)))
           (COND
              [(AND (LISTP REST1)
                    (EQ (CAR REST1)
                        (QUOTE AND)))
               (COND
                  ((AND (LISTP REST2)
                        (EQ (CAR REST2)
                            (QUOTE AND)))
```

```
                    (APPEND REST1 (CDR REST2)))
                   (T (APPEND REST1 (CONS REST2 NIL]
               [(AND (LISTP REST2)
                     (EQ (CAR REST2)
                         (QUOTE AND)))
                (CONS (QUOTE AND)
                      (CONS REST1 (CDR REST2]
               (T (LIST (QUOTE AND)
                        REST1 REST2]
          (T (CONS (CAR X)
                   (for ARG in (CDR X) collect (INTRODUCE-ANDS ARG])
```

(INTRODUCE-LISTS

```
[LAMBDA (X)                                                    (* kbr: "19-Oct-85 16:31")
  (LET (REST)
       (COND
          ((NLISTP X)
           X)
          ((EQ (CAR X)
               (QUOTE QUOTE))
           (KWOTE (CADR X)))
          [(EQ (CAR X)
               (QUOTE CONS))
           (SETQ REST (INTRODUCE-LISTS (CADDR X)))
           (COND
              [(NULL REST)
               (LIST (QUOTE LIST)
                     (INTRODUCE-LISTS (CADR X]
              [(AND (LISTP REST)
                    (EQ (CAR REST)
                        (QUOTE LIST)))
               (CONS (QUOTE LIST)
                     (CONS (INTRODUCE-LISTS (CADR X))
                           (CDR REST]
              (T (LIST (QUOTE CONS)
                       (INTRODUCE-LISTS (CADR X))
                       REST]
          (T (CONS (CAR X)
                   (for ARG in (CDR X) collect (INTRODUCE-LISTS ARG])
```

(JUMPOUTP

```
[LAMBDA (OLD NEW)                                              (* kbr: " 4-Jul-86 17:22")
```

(* It is claimed that JUMPOUTP is a mere optimization of the book version of the rewriter.
The proof rests on two observations. The first is that if any subterm of the rewritten function body fails to satisfy REWRITE-FNCALLP then the entire body fails -- i.e., it does not matter if other parts are super-good.
This means that as soon as we lay our hands on a subterm that is GUARANTEED to survive future rewriting and be returned as part of the value of the REWRITE call in REWRITE-FNCALL we can check that it satisfies REWRITE-FNCALLP and if not, abort then and there. The second lemma is that if the DEFN-FLG of REWRITE is T then the value of that rewrite will survive to be part of the value computed by the REWRITE call in REWRITE-FNCALL.
Proof of this is by inspection of the places REWRITE is called. In particular, if REWRITE's value is that of a recursive call, the call may be passed the same value of the DEFN-FLG, the DEFN-FLG may be turned on only by REWRITE-FNCALL, and must be NIL in rewriting arguments to non-IFs (which might disappear as a result of higher level rewrites), tests to IF's even on the main path through a defn (because the tests may be eliminated by
(IF ITIMES y y)) and in rewrite calls to relieve hyps (which do not have any relation to what is seen by the REWRITE-FNCALLP check in REWRITE-FNCALL) the most subtle part of the proof is that if you are simplifying an (IF test left right) that is guaranteed to participate in the value returned to REWRITE-FNCALL, then both the values of left and right will be -- at least, they will be when they are non-trivial values that might possible offend REWRITE-FNCALLP.
The proof of this is by inspection of REWRITE-IF1 which either returns the newly consed up IF of the values, which is perfect, or else returns pieces (i.e., test, or left, or right's value alone) under conditions that guarantee that nothing is lost.
Thus, if the DEFN-FLG is on, JUMPOUTP can call REWRITE-FNCALLP and jump out of the lowest REWRITE-FNCALL if the newly computed value offends it. Since JUMPOUTP is only called on the branches of IFs there must still be a call of REWRITE-FNCALLP on the final answer in REWRITE-FNCALL since tests
(which could have been eliminated by (IF ITIMES y y)) might still offend.
Finally, to avoid calling REWRITE-FNCALLP exponentially while backing out of an IF-tree, we do not even bother to call it if the old value of the term was itself an IF, since JUMPOUTP okay'd its branches --
but not its test -- earlier. *)

```
   (COND
      (NIL (NOT (EQUAL NEW (SUBLIS ALIST OLD)))
           (SHOWPRINT (SUBLIS ALIST OLD))
           (SHOWPRINT NEW)
           (\GETKEY)))
   (COND
      [(AND DEFN-FLG (NVARIABLEP OLD)
            (NEQ (FN-SYMB OLD)
                 (QUOTE IF))
            (NOT (REWRITE-FNCALLP (CAR FNSTACK)
                                  NEW)))
       (POP-LEMMA-FRAME)
       (RETFROM (QUOTE REWRITE-FNCALL)
              (LET ((TYPE-ALIST *TYPE-ALIST*))
                   (REWRITE-SOLIDIFY (CONS-TERM *FNNAME* *ARGLIST*]
      (T NEW])
```

(KILL-EVENT

```
[LAMBDA (NAME)                                                    (* kbr: "19-Oct-85 16:31")
  (COND
     ((EQ NAME (QUOTE GROUND-ZERO))
      (KILL-LIB))
     (T (for TUPLE in (GETPROP NAME (QUOTE LOCAL-UNDO-TUPLES)) do (ADD-SUB-FACT NIL NIL NIL TUPLE NIL))
        (for SATELLITE in (GETPROP NAME (QUOTE SATELLITES)) do (KILLPROPLIST1 SATELLITE))
        (KILLPROPLIST1 NAME)
        (SETQ CHRONOLOGY (REMOVE1 NAME CHRONOLOGY))
        NAME])
```

(KILL-LIB

```
[LAMBDA NIL                                                       (* kbr: "19-Oct-85 16:31")
                                                                  (* Erase all trace of the lib file. *)
  (COND
     ((BOUNDP (QUOTE LIB-PROPS))
      (for ATM in LIB-ATOMS-WITH-PROPS do (KILLPROPLIST1 ATM))
      (for FN in LIB-ATOMS-WITH-DEFS do (KILL-DEFINITION FN))
      (for VAR in LIB-VARS do (MAKUNBOUND VAR))
      (MAKUNBOUND (QUOTE LIB-VARS))
      (MAKUNBOUND (QUOTE LIB-ATOMS-WITH-PROPS))
      (MAKUNBOUND (QUOTE LIB-ATOMS-WITH-DEFS))
      (MAKUNBOUND (QUOTE LIB-PROPS))
      (MAKUNBOUND (QUOTE LIB-FILE])
```

(KILLPROPLIST1

```
[LAMBDA (ATM)                                                     (* kbr: "19-Oct-85 16:31")
                                                                  (* Kill all properties of ATM that are maintained by the lib file.
                                                                  *)
  (for PROP in LIB-PROPS do (REMPROP ATM PROP))
  (REMPROP ATM (QUOTE LIB-LOC])
```

(LEGAL-CHAR-CODE-SEQ

```
[LAMBDA (LST)                                                     (* kbr: "17-Nov-85 15:38")

        (* WARNING The EVG-OCCUR functions make delicate use of the ascii codes permitted in litatoms in evgs.
        *)

  (AND (LISTP LST)
       [for TAIL on LST bind C until (NLISTP TAIL) always (PROGN (SETQ C (CAR TAIL))
                                                                 (AND (FIXP C)
                                                                      (OR (AND (LESSEQP (CHARCODE A)
                                                                                        C)
                                                                               (LESSEQP C (CHARCODE Z)))
                                                                          (AND (LESSEQP (CHARCODE 0)
                                                                                        C)
                                                                               (LESSEQP C (CHARCODE 9)))
                                                                          (EQUAL C (CHARCODE -]
       (NOT (EQUAL (CAR LST)
                   (CHARCODE -)))
       (NOT (AND (LESSEQP (CHARCODE 0)
                          (CAR LST))
                 (LESSEQP (CAR LST)
                          (CHARCODE 9])
```

(LENGTH-TO-ATOM

```
[LAMBDA (L)                                                       (* kbr: "19-Oct-85 16:31")
  (for TAIL on L until (NLISTP TAIL) count T])
```

(LESSEQP

```
[LAMBDA (I J)                                                     (* kbr: "19-Oct-85 16:31")
  (NOT (LESSP J I])
```

(LEXORDER

```
[LAMBDA (X Y)                                                     (* kbr: "20-Oct-85 18:47")

        (* LEXORDER is a total ordering on LISP objects constructed from numbers, litatoms, and conses.
        See the comment in TERM-ORDER for the definitions of *)

  (COND
     ((NLISTP X)
      (COND
         ((NLISTP Y)

          (* From the VM one can conclude that ALPHORDER is a total ordering when restricted to ATOMs.
          *)

          (ALPHORDER X Y))
         (T T)))
     ((NLISTP Y)
```

```
        NIL)
       ((EQUAL (CAR X)
               (CAR Y))
        (LEXORDER (CDR X)
                  (CDR Y)))
       (T (LEXORDER (CAR X)
                    (CAR Y])
```

(LINEARIZE

```
  [LAMBDA (TERM FLG)                                          (* kbr: "22-Oct-85 15:27")
```

(* If FLG is T linearize TERM, else linearize the negation of TERM.
We store TERM in the LITERALS field regardless of FLG. ADD-EQUATIONS looks in the LITERALS field to see if the CURRENT-LIT is a father of a POLY and if so does not use it in cancellation.
Similarly ADD-EQUATIONS looks in the LEMMAS field for members of the original clause, i.e.,
LITS-THAT-MAY-BE-ASSUMED-FALSE. *)

```
    (LET (LHS RHS LST CONTRA)
         (SETQ LST (COND
                      [[COND
                          (FLG (BM-MATCH TERM (LESSP LHS RHS)))
                          (T (BM-MATCH TERM (NOT (LESSP LHS RHS]
                       (LIST (LIST (COMPRESS-POLY (ADD-LINEAR-TERM (CONS-TERM (QUOTE ADD1)
                                                                              (LIST LHS))
                                                                   (QUOTE POSITIVE)
                                                                   (ADD-LINEAR-TERM RHS (QUOTE NEGATIVE)
                                                                                    (ZERO-POLY TERM]
                      ([COND
                          (FLG (BM-MATCH TERM (EQUAL LHS RHS)))
                          (T (BM-MATCH TERM (NOT (EQUAL LHS RHS]
                       (COND
                          [(OR (POSSIBLY-NUMERIC LHS)
                               (POSSIBLY-NUMERIC RHS))
                           (LIST (LIST [COMPRESS-POLY (ADD-LINEAR-TERM LHS (QUOTE POSITIVE)
                                                                       (ADD-LINEAR-TERM RHS (QUOTE NEGATIVE)
                                                                                        (ZERO-POLY TERM]
                                       (COMPRESS-POLY (ADD-LINEAR-TERM RHS (QUOTE POSITIVE)
                                                                       (ADD-LINEAR-TERM LHS (QUOTE NEGATIVE)
                                                                                        (ZERO-POLY TERM]
                          (T NIL)))
                      [[COND
                          [FLG (BM-MATCH TERM (NOT (LESSP LHS RHS]
                          (T (BM-MATCH TERM (LESSP LHS RHS]
                       (LIST (LIST (COMPRESS-POLY (ADD-LINEAR-TERM RHS (QUOTE POSITIVE)
                                                                   (ADD-LINEAR-TERM LHS (QUOTE NEGATIVE)
                                                                                    (ZERO-POLY TERM]
                      ([COND
                          [FLG (BM-MATCH TERM (NOT (EQUAL LHS RHS]
                          (T (BM-MATCH TERM (EQUAL LHS RHS]
                       (COND
                          [(OR (POSSIBLY-NUMERIC LHS)
                               (POSSIBLY-NUMERIC RHS))
                           (LIST [LIST (ADD-NUMBERP-ASSUMPTION-TO-POLY
                                        LHS
                                        (ADD-NUMBERP-ASSUMPTION-TO-POLY RHS (COMPRESS-POLY
                                                                             (ADD-LINEAR-TERM
                                                                              (CONS-TERM (QUOTE ADD1)
                                                                                         (LIST LHS))
                                                                              (QUOTE POSITIVE)
                                                                              (ADD-LINEAR-TERM RHS
                                                                                               (QUOTE NEGATIVE)
                                                                                               (ZERO-POLY TERM]
                                       (LIST (ADD-NUMBERP-ASSUMPTION-TO-POLY
                                              LHS
                                              (ADD-NUMBERP-ASSUMPTION-TO-POLY RHS (COMPRESS-POLY
                                                                                   (ADD-LINEAR-TERM
                                                                                    (CONS-TERM (QUOTE ADD1)
                                                                                               (LIST RHS))
                                                                                    (QUOTE POSITIVE)
                                                                                    (ADD-LINEAR-TERM LHS
                                                                                                     (QUOTE NEGATIVE)
                                                                                                     (ZERO-POLY TERM]
                          (T NIL)))
                       (T NIL)))
         (SETQ LST (for L in LST collect (for POLY in L unless (MEMBER FALSE (fetch (POLY ASSUMPTIONS) of POLY))
                                      collect POLY)))
         [COND
            ((IEQP (LENGTH LST)
                   2)
```

(* If either member of LST contains a contradiction, we delete that member from LST after moving into each member of the other member of LST the assumptions and lemmas upon which the contradiction depends.
*)

```
             (COND
                [(SETQ CONTRA (for POLY in (CAR LST) when (IMPOSSIBLE-POLYP POLY) do (RETURN POLY)))
```

```
                [for POLY in (CADR LST) do (replace (POLY ASSUMPTIONS) of POLY with (UNION-EQUAL
                                                                                     (fetch (POLY ASSUMPTIONS)
                                                                                        of CONTRA)
                                                                                     (fetch (POLY ASSUMPTIONS)
                                                                                        of POLY)))
                                                (replace (POLY LEMMAS) of POLY with (UNIONQ (fetch (POLY LEMMAS)
                                                                                              of CONTRA)
                                                                                           (fetch (POLY LEMMAS)
                                                                                              of POLY]
                (SETQ LST (LIST (CADR LST]
               ((SETQ CONTRA (for POLY in (CADR LST) when (IMPOSSIBLE-POLYP POLY) do (RETURN POLY)))
                [for POLY in (CAR LST) do (replace (POLY ASSUMPTIONS) of POLY with (UNION-EQUAL (fetch (POLY
                                                                                                    ASSUMPTIONS
                                                                                                    )
                                                                                                 of CONTRA)
                                                                                              (fetch (POLY ASSUMPTIONS)
                                                                                                 of POLY)))
                                                (replace (POLY LEMMAS) of POLY with (UNIONQ (fetch (POLY LEMMAS)
                                                                                              of CONTRA)
                                                                                           (fetch (POLY LEMMAS)
                                                                                              of POLY]
                (SETQ LST (LIST (CAR LST]
        LST])
```

(LISTABLE

```
[LAMBDA (X)                                                    (* kbr: "19-Oct-85 16:31")
  (LET (LHS RHS)
       (COND
          ((VARIABLEP X)
           NIL)
          ((FQUOTEP X)
           NIL)
          ((BM-MATCH X (LIST (QUOTE CONS)
                             LHS RHS))
           (COND
              ((EQUAL RHS (QUOTE (QUOTE NIL)))
               (LIST LHS))
              ((SETQ TEMP-TEMP (LISTABLE RHS))
               (CONS LHS TEMP-TEMP))
              (T NIL)))
          (T NIL])
```

(LOGSUBSETP

```
[LAMBDA (X Y)                                                  (* kbr: "19-Oct-85 16:31")
  (IEQP (LOGAND X Y)
        X])
```

(LOOKUP-HYP

```
[LAMBDA (HYP)                                                  (* kbr: "19-Oct-85 16:31")
```

(* See if HYP is true by type alist or LITS-THAT-MAY-BE-ASSUMED-FALSE considerations --
possibly extending the UNIFY-SUBST if necessary. If successful return T and side-effect UNIFY-SUBST and the current lemma frame appropriately. If unsuccessful, return NIL and side-effect nothing.
*)

```
  (PROG (TERM NOT-FLG TYPE NEG-HYP LIT)
        (COND
           ((BM-MATCH HYP (NOT TERM))
            (SETQ NOT-FLG T))
           (T (SETQ NOT-FLG NIL)
              (SETQ TERM HYP)))
        [COND
           ((AND (NVARIABLEP TERM)
                 (NOT (FQUOTEP TERM))
                 (SETQ TEMP-TEMP (ASSOC (FFN-SYMB TERM)
                                        RECOGNIZER-ALIST)))
            (SETQ TYPE (CDR TEMP-TEMP))
            (SETQ TERM (FARGN TERM 1)))
           (T (SETQ TYPE (LOGNOT TYPE-SET-FALSE]
        [COND
           [NOT-FLG (COND
                       ([for PAIR in TYPE-ALIST thereis (AND (IEQP 0 (LOGAND TYPE (CDR PAIR)))
                                                             (ONE-WAY-UNIFY1 TERM (CAR PAIR]
                        (RETURN T]
           (T (COND
                 ([for PAIR in TYPE-ALIST thereis (AND (LOGSUBSETP (CDR PAIR)
                                                                   TYPE)
                                                       (ONE-WAY-UNIFY1 TERM (CAR PAIR]
                  (RETURN T]                                   (* Having failed to find HYP on the type alist, we now try
                                                               LITS-THAT-MAY-BE-ASSUMED-FALSE.
                                                               *)
        (COND
           [LITS-THAT-MAY-BE-ASSUMED-FALSE (SETQ NEG-HYP (DUMB-NEGATE-LIT HYP))
                  (COND
```

```
                    ((SETQ LIT (for LIT in LITS-THAT-MAY-BE-ASSUMED-FALSE when (ONE-WAY-UNIFY1 NEG-HYP LIT)
                                  do (RETURN LIT)))
                     (PUSH-LEMMA LIT)
                     (RETURN T))
                    (T (RETURN NIL]
          (T (RETURN NIL])
```

(LOOP-STOPPER

```
[LAMBDA (TERM)                                                    (* kbr: "19-Oct-85 16:31")
  (LET (LHS RHS ALL-VARS)
       (COND
          ((AND (BM-MATCH TERM (EQUAL LHS RHS))
                (VARIANTP LHS RHS))
           (SETQ ALL-VARS (ALL-VARS LHS))
           (for PAIR in UNIFY-SUBST when (MEMB (CAR PAIR)
                                               (CDR (MEMB (CDR PAIR)
                                                          ALL-VARS)))
              collect PAIR))
          (T NIL])
```

(MAIN-EVENT-OF

```
[LAMBDA (NAME)                                                    (* kbr: "19-Oct-85 16:31")
  (COND
     ((GETPROP NAME (QUOTE EVENT))
      NAME)
     ((GETPROP NAME (QUOTE MAIN-EVENT)))
     (T (ERROR1 (PQUOTE (PROGN MAIN-EVENT-OF HAS BEEN CALLED ON AN OBJECT , NAMELY (!PPR NAME NIL)
                               , THAT IS NEITHER AN EVENT NOR A SATELLITE OF ANOTHER EVENT !))
                (BINDINGS (QUOTE NAME)
                          NAME)
                (QUOTE HARD]))
```

(CREATE-EVENT

```
[LAMBDA (NAME EVENT)                                              (* kbr: "19-Oct-85 16:31")
  (PUT1 NAME EVENT (QUOTE EVENT))
  (PUT1 NAME (IDATE)
        (QUOTE IDATE))
  (SETQ CHRONOLOGY (CONS NAME CHRONOLOGY))
  (SETQ MAIN-EVENT-NAME NAME])
```

(MAKE-FLATTENED-MACHINE

```
[LAMBDA (FNNAME TERM TESTS)                                       (* kbr: "19-Oct-85 20:01")
```

(* This function builds a list of TESTS-AND-CASE representing the function FNNAME with body TERM.
For each call of FNNAME in body, a TESTS-AND-CASE is returned whose TESTS are all the tests that govern the call and whose CASE is the arglist of the call. This code is a vast change from the previous version, which did not consider terms with or within calls of FNNAME as governors. *)

```
  (COND
     ((OR (VARIABLEP TERM)
          (FQUOTEP TERM))
      NIL)
     [(EQ (FFN-SYMB TERM)
          (QUOTE IF))
      (NCONC (MAKE-FLATTENED-MACHINE FNNAME (FARGN TERM 1)
                                     TESTS)
             [MAKE-FLATTENED-MACHINE FNNAME (FARGN TERM 2)
                                     (APPEND TESTS (LIST (FARGN TERM 1]
             (MAKE-FLATTENED-MACHINE FNNAME (FARGN TERM 3)
                                     (APPEND TESTS (LIST (NEGATE-LIT (FARGN TERM 1]
     [(EQ FNNAME (FFN-SYMB TERM))
      (CONS (create TESTS-AND-CASE
                    TESTS _ TESTS
                    CASE _ (FARGS TERM))
            (for ARG in (FARGS TERM) join (MAKE-FLATTENED-MACHINE FNNAME ARG TESTS]
     (T (for ARG in (FARGS TERM) join (MAKE-FLATTENED-MACHINE FNNAME ARG TESTS])
```

(MAKE-NEW-NAME

```
[LAMBDA NIL                                                       (* kbr: "26-Oct-85 12:59")
  (LET (TEMP)
       (while (NULL (CHK-NEW-NAME (SETQ TEMP (GENSYM (QUOTE G)))
                                  T))
          do NIL)
       TEMP])
```

(MAKE-REWRITE-RULES

```
[LAMBDA (NAME HYPS CONCL)                                         (* kbr: "19-Oct-85 16:31")
```

(* This fn once entertained the idea of returning as many rewrite rules as there were paths through the IF structure of HYPS. That blew us out of the water on a thm whose hyp was (AND (NOT
(EQUAL X Y)) (NOT (LESSP X Y))) because it generated 75 paths! So the fn now returns just one rewrite rule --
or none if CONCL is an explicit value. The rule is LISTed so that the higher level functions still allow the possibility of it

someday returning more than one -- BUT they are all hung under the same fn symbol so this probably is not a useful feature. *)

```
(PROG (LHS RHS)
      [COND
         ((QUOTEP CONCL)
          (RETURN NIL))
         ((BM-MATCH CONCL (EQUAL LHS RHS))
          (SETQ CONCL (LIST (QUOTE EQUAL)
                            LHS
                            (NORMALIZE-IFS (EXPAND-BOOT-STRAP-NON-REC-FNS RHS)
                                           NIL NIL]
      (RETURN (LIST (CREATE-REWRITE-RULE NAME HYPS CONCL NIL])
```

(MAKE-TYPE-RESTRICTION

```
[LAMBDA (TR DV RECOGNIZER TYPE-NO)                                    (* kbr: "19-Oct-85 16:31")
  (LET (TYPE-SET)
       (SETQ TYPE-SET (for R in (CDR TR) bind (LOOP-ANS _ 0)
                         do [SETQ LOOP-ANS (LOGOR LOOP-ANS (CDR (ASSOC R (CONS (CONS RECOGNIZER (LOGBIT
                                                                                                  TYPE-NO
                                                                                                  ))
                                                                                      RECOGNIZER-ALIST]
                         finally (RETURN LOOP-ANS)))
       [COND
          ((EQ (CAR TR)
               (QUOTE NONE-OF))
           (SETQ TYPE-SET (LOGNOT TYPE-SET]
       (create TYPE-RESTRICTION
               TERM _ (COND
                         ((EQ (CAR TR)
                              (QUOTE ONE-OF))
                          (DISJOIN (for R in (CDR TR) collect (FCONS-TERM* R (QUOTE X)))
                                   NIL))
                         (T (CONJOIN [for R in (CDR TR) collect (DUMB-NEGATE-LIT (FCONS-TERM* R (QUOTE X]
                                     NIL)))
               TYPE-SET _ TYPE-SET
               DEFAULT _ (CONS-TERM DV NIL])
```

(MAX-FORM-COUNT

```
[LAMBDA (X)                                                           (* kbr: " 4-Jul-86 18:32")
```

(* The size of the most complicated path in X regarded as a tree of IFs.
*)

```
(COND
   ((VARIABLEP X)
    0)
   ((FQUOTEP X)
```

(* MAX-FORM-COUNT once used FORM-COUNT-EVG to compute the size of an evg.
But that function computed MAX-FORM-COUNT for 1000 that was bigger than for 999 and so the REWRITE package believed it was making progress and would open up something like
(LESSP X 1000)%. We have decided to try just measuring the LISP size of the evg, as a better estimation of whether we are making progress. *)

```
    (COUNT (CADR X)))
   [(EQ (FFN-SYMB X)
        (QUOTE IF))
    (MAX (MAX-FORM-COUNT (FARGN X 2))
         (MAX-FORM-COUNT (FARGN X 3]
   (T (ADD1 (for ARG in (FARGS X) sum (MAX-FORM-COUNT ARG])
```

(MAXIMAL-ELEMENTS

```
[LAMBDA (LST MEASURE)                                                 (* kbr: "19-Oct-85 16:31")
  (LET (ANS MAX TEMP)
       [for X in LST do (SETQ TEMP (APPLY* MEASURE X))
                        (COND
                           ((OR (NULL MAX)
                                (GREATERP TEMP MAX))
                            (SETQ MAX TEMP)
                            (SETQ ANS (LIST X)))
                           ((EQUAL TEMP MAX)
                            (SETQ ANS (NCONC1 ANS X]
       ANS])
```

(MEANING-SIMPLIFIER

```
[LAMBDA (TERM)                                                        (* kbr: "19-Oct-85 16:31")
```

(* When the theorem-prover assents to a theorem or accepts a definition, in which theory is it working? Heretofore, the answer has been= in the theory consisting of chapter 3 of ACL plus the user's definitions and axioms.
Because of the addition of metatheorems, the answer to that question is no longer so simple.
To answer the question, we first elaborate the notion of a presented in the meta paper.
Let us say that an event is a pair (ev term) where ev is either DEFN, ADD-SHELL, ADD-AXIOM, PROVE-LEMMA, or DCL.

An member (ev term) if ev is DCL, then term is a function symbol and otherwise term is a term.
Given a list of events, we say that a function symbol is has not been defined, has not been DCLed, and is not mentioned in the basic axioms. A the concept being defined is new, and all the other function symbols are not new for a shell invocation all the introduced symbols (excepting possibly the default objects) are new for a DCL, the symbol is new for a theorem or arbitrary axiom, none of the symbols used are new and the theorems are provable from the preceding axioms (including definitions and shell invocations)%. We define a symbols on META-NAMES are DCLed or defined at the beginning of the chronology as in BOOT-STRAP-INSTRS, immediately after each DCL and DEFINITION the MEANING and ARITY axioms for the newly introduced function symbol are added as arbitrary axioms, and there is otherwise no mention of any META-NAME except in theorems. We vouch that our theorem-prover only calls proved in the user chronology.
We now make a like to work in a chronology with all those metaaxioms To make the user happy, we show that corresponding to any user chronology is a axioms and definitions only by the addition of more definitions.
Furthermore, we observe that in the real chronology, all the theorems of the user chronology
(the ones the theorem-prover proved) are theorems in the real chronology after we replace each function symbol in META-NAMES with another function symbol. Thus, any theorem proved in the user chronology about concepts he has defined or DCLed are literally theorems in the real chronology. If he objects to having extra definitions around, then tough luck for him. Given a initial BOOT-STRAP events that mention META-NAMES, replacing the MEANING and ARITY axiom after each DCL or definition with the collection of definitions called the metadefinitions in the meta paper for the i non-new function symbols at that point in the chronology -- amended by indexing each META-NAME with i --
and altering each theorem by adding to each META-NAME the appropriate index.
Note we do not have to index user supplied axioms or definitions since they may not contain META-NAMES.
Note we are forbidding the user from using META-NAMES in definitions even if he want to define concepts to help him prove metalemmas! Why is a indexed theorems can be proved? The answer is that at any point i in the user chronology (that is, after i definitions and declarations) and for each axiom about a META-NAME in the user chronology, we can prove, in the real chronology, at the corresponding point, the indexed version of the axiom.
The proof of this assertion is merely the observation that the metaaxioms follow from the metadefinitions, so the indexed metaaxioms follow from the indexed metadefinitions. The foregoing facts are independent of the use of metalemmas.
Now let us consider how metalemmas are used. Suppose that a metalemma is proved at some constructive point i in a user chronology and that at some point i+p we use the metalemma. We claim that the inference can be proved at point i+p in the user chronology. As a corollary to what has been said before, we also conclude that the inference can be proved in the corresponding chronology is obvious since the metatheorem at i was proved about the same symbol MEANING we will use at i+p to lift and drop the formulas in question. *)

```
(LET (X ALIST FN TL)
     (MATCH! TERM (MEANING X ALIST))
     (COND
        ((VARIABLEP X)
         TERM)
        [(SHELLP X)
         (COND
            ((NEQ (FN-SYMB X)
                  (QUOTE CONS))
             (CONS-TERM (QUOTE LOOKUP)
                    (FARGS TERM)))
            (T (SETQ FN (ARGN X 1))
               (SETQ TL (ARGN X 2))
               (COND
                  ((AND (QUOTEP FN)
                        (LITATOM (CADR FN)))
                   (COND
                      ((EQ (CADR FN)
                           (QUOTE QUOTE))
                       (FCONS-TERM* (QUOTE CAR)
                                    TL))
                      [(AND (GETPROP (CADR FN)
                                     (QUOTE TYPE-PRESCRIPTION-LST))
                            (NOT (MEMB (CADR FN)
                                       META-NAMES)))
                       (CONS-TERM (CADR FN)
                                  (for I from 1 to (ARITY (CADR FN))
                                     collect (FCONS-TERM* (QUOTE MEANING)
                                                          (FCONS-TERM* (QUOTE CAR)
                                                                       (CELL (SUB1 I)
                                                                             TL))
                                                          ALIST]
                      (T TERM)))
                  (T TERM]
         (T TERM])
```

(MEMB-NEGATIVE

```
[LAMBDA (LIT CL)                                          (* kbr: "19-Oct-85 16:31")
  (COND
     ((NLISTP CL)
      NIL)
     ((COMPLEMENTARYP LIT (CAR CL))
      T)
     (T (MEMB-NEGATIVE LIT (CDR CL])
```

(MENTIONSQ

```
[LAMBDA (AT TREE)                                         (* kbr: "19-Oct-85 16:31")
  (COND
     ((NLISTP TREE)
      (EQ AT TREE))
     (T (OR (MENTIONSQ AT (CAR TREE))
            (MENTIONSQ AT (CDR TREE])
```

(MENTIONSQ-LST

```
[LAMBDA (LST TREE)                                          (* kbr: "19-Oct-85 16:31")
  (COND
     ((NLISTP TREE)
      (MEMB TREE LST))
     (T (OR (MENTIONSQ-LST LST (CAR TREE))
            (MENTIONSQ-LST LST (CDR TREE])
```

(MERGE-CAND1-INTO-CAND2

```
[LAMBDA (CAND1 CAND2)                                       (* kbr: "19-Oct-85 16:56")

        (* Note: The guts of this function is MERGE-TESTS-AND-ALISTS-LSTS.
        The tests preceding it are heuristic only. If MERGE-TESTS-AND-ALISTS-LSTS returns non-NIL then it returns a sound
        induction indeed, it merely extends some of the substitutions in the second candidate.
        *)

  (LET (SCORE1 CONTROLLERS1 CHANGED-VARS1 UNCHANGEABLES1 TESTS-AND-ALISTS-LST1 JUSTIFICATION1 TERM1
              OTHER-TERMS1 SCORE2 CONTROLLERS2 CHANGED-VARS2 UNCHANGEABLES2 TESTS-AND-ALISTS-LST2
              JUSTIFICATION2 TERM2 OTHER-TERMS2 ALISTS TESTS-AND-ALISTS-LST VARS)
       (SETQ SCORE1 (fetch (CANDIDATE SCORE) of CAND1))
       (SETQ CONTROLLERS1 (fetch (CANDIDATE CONTROLLERS) of CAND1))
       (SETQ CHANGED-VARS1 (fetch (CANDIDATE CHANGED-VARS) of CAND1))
       (SETQ UNCHANGEABLES1 (fetch (CANDIDATE UNCHANGEABLE-VARS) of CAND1))
       (SETQ TESTS-AND-ALISTS-LST1 (fetch (CANDIDATE TESTS-AND-ALISTS-LST) of CAND1))
       (SETQ JUSTIFICATION1 (fetch (CANDIDATE JUSTIFICATION) of CAND1))
       (SETQ TERM1 (fetch (CANDIDATE INDUCTION-TERM) of CAND1))
       (SETQ OTHER-TERMS1 (fetch (CANDIDATE OTHER-TERMS) of CAND1))
       (SETQ SCORE2 (fetch (CANDIDATE SCORE) of CAND2))
       (SETQ CONTROLLERS2 (fetch (CANDIDATE CONTROLLERS) of CAND2))
       (SETQ CHANGED-VARS2 (fetch (CANDIDATE CHANGED-VARS) of CAND2))
       (SETQ UNCHANGEABLES2 (fetch (CANDIDATE UNCHANGEABLE-VARS) of CAND2))
       (SETQ TESTS-AND-ALISTS-LST2 (fetch (CANDIDATE TESTS-AND-ALISTS-LST) of CAND2))
       (SETQ JUSTIFICATION2 (fetch (CANDIDATE JUSTIFICATION) of CAND2))
       (SETQ TERM2 (fetch (CANDIDATE INDUCTION-TERM) of CAND2))
       (SETQ OTHER-TERMS2 (fetch (CANDIDATE OTHER-TERMS) of CAND2))

        (* We once merged only if both cands agreed on the intersection of the CHANGED-VARS.
        But the theorem that, under suitable conditions, (EV FLG X VA FA N) IEQP
        (EV FLG X VA FA K) made us realize it was important only to agree on the intersection of the controllers.
        Note in fact that we mean the changing controllers -- there seems to be no need to merge two inductions if they only share
        unchanging controllers. However the theorem that (GETPROP I
        (SET J VAL MEM)) IEQP ... (GETPROP I MEM) ... illustrates the situation in which the controllers, {I} and {J} do not even
        overlap but the accumulators {MEM} do and we want a merge.
        So we want agreement on the intersection of the changing controllers
        (if that is nonempty) or on the accumulators. For soundness it does not matter what list of vars we want to agree on because
        no matter what, MERGE-TESTS-AND-ALISTS-LSTS returns either NIL or an extension of the second candidates alists.
        *)

       (AND (SETQ VARS (OR (INTERSECTION CONTROLLERS1 (INTERSECTION CONTROLLERS2 (INTERSECTION CHANGED-VARS1
                                                                                    CHANGED-VARS2)))
                           (INTERSECTION CHANGED-VARS1 CHANGED-VARS2)))
            (NOT (INTERSECTP UNCHANGEABLES1 CHANGED-VARS2))
            (NOT (INTERSECTP UNCHANGEABLES2 CHANGED-VARS1))
            (SETQ TESTS-AND-ALISTS-LST (MERGE-TESTS-AND-ALISTS-LSTS TESTS-AND-ALISTS-LST1
                                                TESTS-AND-ALISTS-LST2 VARS))
            (create CANDIDATE
                    SCORE _ (PLUS SCORE1 SCORE2)
                    CONTROLLERS _ (UNIONQ CONTROLLERS1 CONTROLLERS2)
                    CHANGED-VARS _ (UNIONQ CHANGED-VARS1 CHANGED-VARS2)
                    UNCHANGEABLE-VARS _ (UNIONQ UNCHANGEABLES1 UNCHANGEABLES2)
                    TESTS-AND-ALISTS-LST _ TESTS-AND-ALISTS-LST
                    JUSTIFICATION _ JUSTIFICATION2
                    INDUCTION-TERM _ TERM2
                    OTHER-TERMS _ (ADD-TO-SET TERM1 (UNION-EQUAL OTHER-TERMS1 OTHER-TERMS2])
```

(MERGE-CANDS

```
[LAMBDA (CAND1 CAND2)                                       (* kbr: "19-Oct-85 16:31")
  (OR (FLUSH-CAND1-DOWN-CAND2 CAND1 CAND2)
      (FLUSH-CAND1-DOWN-CAND2 CAND2 CAND1)
      (MERGE-CAND1-INTO-CAND2 CAND1 CAND2)
      (MERGE-CAND1-INTO-CAND2 CAND2 CAND1])
```

(MERGE-DESTRUCTOR-CANDIDATES

```
[LAMBDA (LST)                                               (* kbr: "19-Oct-85 16:31")

        (* The elements of LST are lists of terms. Whenever the CARs of two elements are EQUAL we UNIONQ together the CDRs.
        *)

  (TRANSITIVE-CLOSURE LST (FUNCTION (LAMBDA (X Y)
                                      (COND
                                         [(EQUAL (CAR X)
                                                 (CAR Y))
                                          (CONS (CAR X)
```

```
                                                    (UNION-EQUAL (CDR X)
                                                                 (CDR Y]
                                                (T NIL])
```

(MERGE-TESTS-AND-ALISTS

```
[LAMBDA (TA1 TA2)                                                    (* kbr: "19-Oct-85 16:31")
  (AND (SETQ ALISTS (PIGEON-HOLE (fetch (TESTS-AND-ALISTS ALISTS) of TA1)
                                 (fetch (TESTS-AND-ALISTS ALISTS) of TA2)
                                 [FUNCTION (LAMBDA (ALIST1 ALIST2)
```

(* Union the two alists if they have a non-trivial intersection, that is, they intersect with a pair other than one like (ITIMES), and they agree on their intersection. *)

```
                                           (AND (for PAIR1 in ALIST1 thereis (AND (NEQ (CAR PAIR1)
                                                                                       (CDR PAIR1))
                                                                                  (MEMBER PAIR1 ALIST2)))
                                                (for PAIR1 in ALIST1 bind PAIR2 when (SETQ PAIR2 (ASSOC (CAR PAIR1)
                                                                                                        ALIST2))
                                                   always (EQUAL PAIR2 PAIR1))
                                                (UNION-EQUAL ALIST1 ALIST2]
                                 T NIL))
       (create TESTS-AND-ALISTS
               TESTS _ (fetch (TESTS-AND-ALISTS TESTS) of TA2)
               ALISTS _ ALISTS])
```

(MERGE-TESTS-AND-ALISTS-LSTS

```
[LAMBDA (TESTS-AND-ALISTS-LST1 TESTS-AND-ALISTS-LST2 VARS)      (* kbr: "20-Oct-85 19:21")
```

(* If every alist in TESTS-AND-ALISTS-LST1 fits into an alist in TESTS-AND-ALISTS-LST2, then return the new TESTS-AND-ALISTS-LST obtained by putting each alist in TESTS-AND-ALISTS-LST1 into every alist in TESTS-AND-ALISTS-LST2 into which it fits. Else return NIL. ALIST1 fits into ALIST2 iff the two agree on every var in VARS. To merge one alist into another we extend the second alist by adding to it every pair of the first, provided that pair does not clash with an existing pair of the second. *)

```
  (LET (BUCKETS ALIST FLG)
       [SETQ BUCKETS (for TA in TESTS-AND-ALISTS-LST2 collect (for ALIST in (fetch (TESTS-AND-ALISTS ALISTS)
                                                                                   of TA)
                                                                 collect (CONS ALIST NIL]
       (COND
          [(for TA1 in TESTS-AND-ALISTS-LST1
              always (for ALIST1 in (fetch (TESTS-AND-ALISTS ALISTS) of TA1)
                        always (PROGN (SETQ FLG NIL)
                                      [for BUCKET in BUCKETS
                                         do (for PAIR in BUCKET
                                               do (COND
                                                     ((FITS ALIST1 (CAR PAIR)
                                                            VARS)
                                                      (RPLACD PAIR (ADD-TO-SET (EXTEND-ALIST ALIST1
                                                                                             (CAR PAIR))
                                                                               (CDR PAIR)))
                                                      (SETQ FLG T]
                                      FLG)))
           (for TA in TESTS-AND-ALISTS-LST2 as BUCKET in BUCKETS
              collect (create TESTS-AND-ALISTS
                              TESTS _ (fetch (TESTS-AND-ALISTS TESTS) of TA)
                              ALISTS _ (for X in BUCKET bind LOOP-ANS do (SETQ LOOP-ANS (UNION-EQUAL
                                                                                         (OR (CDR X)
                                                                                             X)
                                                                                         LOOP-ANS))
                                       finally (RETURN LOOP-ANS]
          (T NIL])
```

(META-LEMMAP

```
[LAMBDA (X)                                                          (* kbr: "19-Oct-85 16:31")
  (NLISTP (fetch (REWRITE-RULE CONCL) of X])
```

(MULTIPLE-PIGEON-HOLE

```
[LAMBDA (PIGEONS HOLES FN)                                           (* kbr: "19-Oct-85 16:31")
  (LET (TEMP PAIRLST)
       (SETQ PAIRLST (for X in HOLES collect (CONS NIL X)))
       (COND
          ((for PIGEON in PIGEONS always (for PAIR in PAIRLST bind FLG do (SETQ TEMP (APPLY* FN PIGEON
                                                                                             (CDR PAIR)))
                                                                          (COND
                                                                             (TEMP (RPLACD PAIR TEMP)
                                                                                   (SETQ FLG T)))
                                         finally (RETURN FLG)))
           (for PAIR in PAIRLST collect (CDR PAIR)))
          (T (ERROR1 (PQUOTE (PROGN MULTIPLE-PIGEON-HOLE FAILED TO GETPROP EVERYTHING IN A POT.))
                     (BINDINGS)
                     (QUOTE HARD])

)
```

```
(RPAQQ CODE-N-RCOMS
       ((* CODE-N-R *)
        (FNS BM-NEGATE NEGATE-LIT NEXT-AVAILABLE-TYPE-NO NO-CROWDINGP NO-DUPLICATESP NO-OP NON-RECURSIVE-DEFNP
             NORMALIZE-IFS NOT-EQUAL-0? NOT-IDENT NOT-LESSP? NOT-TO-BE-REWRITTENP NUMBERP? OBJ-TABLE OCCUR
             OCCUR-CNT OCCUR-LST ONE-WAY-UNIFY ONE-WAY-UNIFY-LIST ONE-WAY-UNIFY1 ONE-WAY-UNIFY11 ONEIFY
             ONEIFY-ASSUME-FALSE ONEIFY-ASSUME-TRUE ONEIFY-TEST OPTIMIZE-COMMON-SUBTERMS PARTITION
             PARTITION-CLAUSES PATH-ADD-TO-SET PATH-EQ PATH-POT-SUBSUMES PATH-UNION PEGATE-LIT PETITIO-PRINCIPII
             PICK-HIGH-SCORES PIGEON-HOLE PIGEON-HOLE-IN-ALL-POSSIBLE-WAYS PIGEON-HOLE1 PLUSJOIN POLY-MEMBER
             POP-CLAUSE-SET POP-LEMMA-FRAME POP-LINEARIZE-ASSUMPTIONS-FRAME POPU POSSIBLE-IND-PRINCIPLES
             POSSIBLY-NUMERIC POWER-EVAL POWER-REP PPC PPE PPE-LST BM-PPR PPRINDENT PPSD PPSD-LST PREPROCESS
             PREPROCESS-HYPS PRETTYIFY-CLAUSE PRETTYIFY-LISP PRIMITIVE-RECURSIVEP PRIMITIVEP PRINT-STACK
             PRINT-STATS PRINT-TO-DISPLAY PROCESS-EQUATIONAL-POLYS PROPERTYLESS-SYMBOLP PROVE PROVE-TERMINATION
             PROVEALL PUSH-CLAUSE-SET PUSH-LEMMA PUSH-LEMMA-FRAME PUSH-LINEARIZE-ASSUMPTION
             PUSH-LINEARIZE-ASSUMPTIONS-FRAME PUSHU PUT-CURSOR PUT-INDUCTION-INFO PUT-LEVEL-NO
             PUT-TYPE-PRESCRIPTION PUT0 PUT00 PUT1 PUT1-LST PUTD1 QUICK-BLOCK-INFO QUICK-BLOCK-INFO1
             QUICK-WORSE-THAN R REDO! REDO-UNDONE-EVENTS BM-REDUCE REDUCE1 REFLECT0 RELIEVE-HYPS
             RELIEVE-HYPS-NOT-OK RELIEVE-HYPS1 REMOVE-*2*IFS REMOVE-NEGATIVE REMOVE-REDUNDANT-TESTS REMOVE1
             REMOVE-TRIVIAL-EQUATIONS REMOVE-UNCHANGING-VARS REMPROP1 RESTART RESTART-BATCH REWRITE
             REWRITE-FNCALL REWRITE-FNCALLP REWRITE-IF REWRITE-IF1 REWRITE-LINEAR-CONCL REWRITE-SOLIDIFY
             REWRITE-TYPE-PRED REWRITE-WITH-LEMMAS REWRITE-WITH-LINEAR RPLACAI)))

          (* * CODE-N-R *)

(DEFINEQ
```

(BM-NEGATE

```
  [LAMBDA (TERM)                                                    (* kbr: "19-Oct-85 16:31")
    (COND
       ((FALSE-NONFALSEP TERM)
        (COND
           (DEFINITELY-FALSE TRUE)
           (T FALSE)))
       ((VARIABLEP TERM)
        (LIST (QUOTE NOT)
              TERM))
       (T (SELECTQ (FFN-SYMB TERM)
                   (NOT (COND
                           ((BOOLEAN (FARGN TERM 1))
                            (FARGN TERM 1))
                           (T (FCONS-TERM* (QUOTE IF)
                                           (FARGN TERM 1)
                                           TRUE FALSE))))
                   (AND (DISJOIN2 (BM-NEGATE (FARGN TERM 1))
                                  (BM-NEGATE (FARGN TERM 2))
                                  NIL))
                   (OR (CONJOIN2 (BM-NEGATE (FARGN TERM 1))
                                 (BM-NEGATE (FARGN TERM 2))
                                 NIL))
                   (FCONS-TERM* (QUOTE NOT)
                                TERM])
```

(NEGATE-LIT

```
  [LAMBDA (TERM)                                                    (* kbr: "19-Oct-85 16:31")
    (COND
       ((FALSE-NONFALSEP TERM)
        (COND
           (DEFINITELY-FALSE TRUE)
           (T FALSE)))
       ((VARIABLEP TERM)
        (FCONS-TERM* (QUOTE NOT)
                     TERM))
       ((EQ (FFN-SYMB TERM)
            (QUOTE NOT))
        (FARGN TERM 1))
       (T (FCONS-TERM* (QUOTE NOT)
                       TERM])
```

(NEXT-AVAILABLE-TYPE-NO

```
  [LAMBDA NIL                                                       (* kbr: "19-Oct-85 16:31")
    (LET (TYPE-NO)
         (SETQ TYPE-NO (for I from 0 when (NOT (for PAIR in SHELL-ALIST thereis (EQUAL (CDR PAIR)
                                                                                      I)))
                          do (RETURN I)))
         [COND
            ((GREATERP TYPE-NO 30)
             (ERROR1 (PQUOTE (PROGN TOO MANY SHELLS ! BECAUSE OF OUR USE OF 32-BIT WORDS TO REPRESENT SETS OF
                                    SHELL TYPES , THE NEED TO RESERVE ONE BIT FOR INTERNAL USE ,
                                      AND THE EXISTENCE OF 31 PREVIOUSLY DEFINED SHELLS , WE CANNOT ACCEPT
                                    FURTHER ADD-SHELL COMMANDS %.))
                     (BINDINGS)
                     (QUOTE HARD]
         TYPE-NO])
```

(NO-CROWDINGP

```
[LAMBDA (HOLES PRED PICKS)                                  (* kbr: "19-Oct-85 16:31")
  (COND
     ((NULL HOLES)
      T)
     ([for X in (CAR HOLES) thereis (AND (for Y in PICKS never (APPLY* PRED X Y))
                                         (NO-CROWDINGP (CDR HOLES)
                                                PRED
                                                (CONS X PICKS]
      T)
     (T NIL])
```

(NO-DUPLICATESP

```
[LAMBDA (L)                                                 (* kbr: "19-Oct-85 16:31")
  (for TAIL on L never (MEMB (CAR TAIL)
                             (CDR TAIL])
```

(NO-OP

```
[LAMBDA NIL                                                 (* kbr: "19-Oct-85 16:31")
  NIL])
```

(NON-RECURSIVE-DEFNP

```
[LAMBDA (FNNAME)                                            (* kbr: "19-Oct-85 16:31")
                                                            (* We use the fact that this AND returns the SDEFN! *)
  (AND (NOT (DISABLEDP FNNAME))
       (NOT (GETPROP FNNAME (QUOTE INDUCTION-MACHINE)))
       (GETPROP FNNAME (QUOTE SDEFN])
```

(NORMALIZE-IFS

```
[LAMBDA (TERM TRUE-TERMS FALSE-TERMS)                       (* kbr: "19-Oct-85 16:31")
  (LET (T1 T2 T3 T11 T12 T13 BAD-ARG)
       (COND
          ((VARIABLEP TERM)
           (COND
              ((MEMB TERM FALSE-TERMS)
               FALSE)
              (T TERM)))
          ((FQUOTEP TERM)
           TERM)
          [(BM-MATCH TERM (IF T1 T2 T3))
           (SETQ T1 (NORMALIZE-IFS T1 TRUE-TERMS FALSE-TERMS))
           (COND
              ((OR (EQUAL T1 TRUE)
                   (MEMBER T1 TRUE-TERMS))
               (NORMALIZE-IFS T2 TRUE-TERMS FALSE-TERMS))
              ((OR (EQUAL T1 FALSE)
                   (MEMBER T1 FALSE-TERMS))
               (NORMALIZE-IFS T3 TRUE-TERMS FALSE-TERMS))
              ((BM-MATCH T1 (IF T11 T12 T13))
               (NORMALIZE-IFS (FCONS-TERM* (QUOTE IF)
                                           T11
                                           (FCONS-TERM* (QUOTE IF)
                                                  T12 T2 T3)
                                           (FCONS-TERM* (QUOTE IF)
                                                  T13 T2 T3))
                              TRUE-TERMS FALSE-TERMS))
              (T (SETQ T2 (NORMALIZE-IFS T2 (CONS T1 TRUE-TERMS)
                                         FALSE-TERMS))
                 (SETQ T3 (NORMALIZE-IFS T3 TRUE-TERMS (CONS T1 FALSE-TERMS)))
                 (COND
                    ((EQUAL T2 T3)
                     T2)
                    ((AND (BOOLEAN T1)
                          (EQUAL T2 TRUE)
                          (AND (FALSE-NONFALSEP T3)
                               DEFINITELY-FALSE))
                     T1)
                    (T (FCONS-TERM* (QUOTE IF)
                                    T1 T2 T3]
          (T [SETQ TERM (CONS-TERM (CAR TERM)
                                   (for ARG in (FARGS TERM) collect (NORMALIZE-IFS ARG TRUE-TERMS FALSE-TERMS]
             [COND
                ((BM-MATCH TERM (EQUAL T1 T2))
                 (COND
                    ((EQUAL T1 T2)
                     (SETQ TERM TRUE))
                    ((NOT-IDENT T1 T2)
                     (SETQ TERM FALSE]
             (COND
                ((FQUOTEP TERM)
                 TERM)
                [(SETQ BAD-ARG (for ARG in (FARGS TERM) when [BM-MATCH ARG (COND
```

```
                                                                    ((T1 T2 T3]
                              do (RETURN ARG)))
                  (FCONS-TERM* (QUOTE IF)
                               T1
                               (NORMALIZE-IFS (SUBST-EXPR T2 BAD-ARG TERM)
                                              (CONS T1 TRUE-TERMS)
                                              FALSE-TERMS)
                               (NORMALIZE-IFS (SUBST-EXPR T3 BAD-ARG TERM)
                                              TRUE-TERMS
                                              (CONS T1 FALSE-TERMS]
                 ((MEMBER TERM FALSE-TERMS)
                  FALSE)
                 ((AND (MEMBER TERM TRUE-TERMS)
                       (BOOLEAN TERM))
                  TRUE)
                 (T TERM])
```

(NOT-EQUAL-0?

```
[LAMBDA (TERM)                                          (* kbr: "19-Oct-85 16:31")
  (PROG (X Y TEMP EQUALITY)
        (COND
           [(BM-MATCH TERM (DIFFERENCE X Y))
            (RETURN (BM-NEGATE (NOT-LESSP? Y X]
           ([OR (BM-MATCH TERM (ADD1 &))
                (AND (QUOTEP TERM)
                     (NOT (EQUAL (CADR TERM)
                                 0]
            (RETURN TRUE)))
        (SETQ EQUALITY (FCONS-TERM* (QUOTE EQUAL)
                                    TERM ZERO))
        (SETQ TEMP (TYPE-SET EQUALITY))
        (COND
           ((IEQP TEMP TYPE-SET-TRUE)
            (RETURN FALSE))
           ((IEQP TEMP TYPE-SET-FALSE)
            (RETURN TRUE))
           (T (RETURN (FCONS-TERM* (QUOTE NOT)
                                   EQUALITY])
```

(NOT-IDENT

```
[LAMBDA (TERM1 TERM2)                                   (* kbr: "19-Oct-85 16:31")
  (COND
     ((AND (VALUEP TERM1)
           (VALUEP TERM2)
           (NOT (EQUAL TERM1 TERM2)))
      T)
     ((OR (AND (BTM-OBJECTP TERM1)
               (SHELL-CONSTRUCTORP TERM2))
          (AND (BTM-OBJECTP TERM2)
               (SHELL-CONSTRUCTORP TERM1)))
```

(* Note, we do not even bother to check that they are of the same type, since if they weren't they'd be unequal on type considerations alone. *)

```
      T)
     ((IEQP 0 (LOGAND (TYPE-SET TERM1)
                      (TYPE-SET TERM2)))
      T)
     ((SHELL-OCCUR TERM1 TERM2)
      T)
     ((SHELL-OCCUR TERM2 TERM1)
      T)
     (T NIL])
```

(NOT-LESSP?

```
[LAMBDA (X Y)                                           (* kbr: "19-Oct-85 16:31")
  (PROG (TEMP TERM)
        [COND
           ((AND (EQUAL Y (QUOTE (QUOTE 1)))
                 (IEQP (TYPE-SET X)
                       TYPE-SET-NUMBERS))
            (RETURN (NOT-EQUAL-0? X]
        [SETQ TEMP (TYPE-SET (SETQ TERM (FCONS-TERM* (QUOTE LESSP)
                                                     X Y]
        (RETURN (COND
                   ((IEQP TEMP TYPE-SET-FALSE)
                    TRUE)
                   ((IEQP TEMP TYPE-SET-TRUE)
                    FALSE)
                   (T (BM-NEGATE TERM])
```

(NOT-TO-BE-REWRITTENP

```
[LAMBDA (TERM ALIST)                                    (* kbr: "19-Oct-85 16:31")
```

```
        (* We assume TERM is a nonvariable nonQUOTEP and that TERMS-TO-BE-IGNORED-BY-REWRITE contains no vars or
        QUOTEPs. Let term' be (SUBLIS-VAR ALIST TERM)%. If term' is a member of TERMS-TO-BE-IGNORED-BY-REWRITE
        we return term' else NIL. We would like to do the membership test without doing the substitution, but the maintenance of
        QUOTE-normal form by SUBLIS-VAR complicates matters. We first ask whether the FFN-SYMB of TERM is the FFN-SYMB
        of any term to be ignored. If not, we return NIL. Else we do the substitution and member check.
        The correctness of this function is obvious in the case that we do the substitution.
        So suppose we return NIL without doing the substitution. Suppose, contrary to correctness that term' is a member of the to
        be ignored list. Then term' is not a QUOTEP. But in that case the FFN-SYMB of term' is that of TERM and must have
        passed our initial test. *)

    (COND
       ((AND (for X in TERMS-TO-BE-IGNORED-BY-REWRITE thereis (EQ (FFN-SYMB TERM)
                                                                  (FFN-SYMB X)))
             (MEMBER (SETQ TEMP-TEMP (SUBLIS-VAR ALIST TERM))
                     TERMS-TO-BE-IGNORED-BY-REWRITE))
        TEMP-TEMP)
       (T NIL])
```

(NUMBERP?

```
  [LAMBDA (TERM)                                                    (* kbr: "19-Oct-85 16:31")
    (LET (TEMP)
         (SETQ TEMP (TYPE-SET TERM))
         (COND
            ((IEQP TEMP TYPE-SET-NUMBERS)
             TRUE)
            ((NOT (LOGSUBSETP TYPE-SET-NUMBERS TEMP))
             FALSE)
            (T (FCONS-TERM* (QUOTE NUMBERP)
                      TERM])
```

(OBJ-TABLE

```
  [LAMBDA (TYPE-SET OBJECTIVE ID-IFF)                               (* kbr: "19-Oct-85 16:31")
    (SELECTQ OBJECTIVE
         (TRUE (COND
                   ((IEQP TYPE-SET TYPE-SET-TRUE)
                    TRUE)
                   ((EQ ID-IFF (QUOTE ID))
                    NIL)
                   ((LOGSUBSETP TYPE-SET-FALSE TYPE-SET)
                    NIL)
                   (T TRUE)))
         (FALSE (COND
                    ((IEQP TYPE-SET TYPE-SET-FALSE)
                     FALSE)
                    (T NIL)))
         (? (COND
               ((IEQP TYPE-SET TYPE-SET-FALSE)
                FALSE)
               ((IEQP TYPE-SET TYPE-SET-TRUE)
                TRUE)
               ((EQ ID-IFF (QUOTE ID))
                NIL)
               ((LOGSUBSETP TYPE-SET-FALSE TYPE-SET)
                NIL)
               (T TRUE)))
         (ERROR1 (PQUOTE (PROGN UNRECOGNIZED REWRITE OBJECTIVE , (!PPR OBJECTIVE NIL)
                                %.))
                 (BINDINGS (QUOTE OBJECTIVE)
                        OBJECTIVE)
                 (QUOTE HARD])
```

(OCCUR

```
  [LAMBDA (TERM1 TERM2)                                             (* kbr: "20-Oct-85 14:23")
                                                                    (* Does TERM1 occur inside TERM2? *)
    (COND
       ((VARIABLEP TERM2)
        (EQ TERM1 TERM2))
       ((FQUOTEP TERM2)
        (COND
           [(QUOTEP TERM1)

          (* Consider that TERM1 is built up out of smaller constants, one of which could be constant TERM2.
          *)

             (COND
                ((FIXP (CADR TERM1))
                 (EVG-OCCUR-NUMBER (CADR TERM1)
                        (CADR TERM2)))
                ((AND (LEGAL-CHAR-CODE-SEQ (CADR TERM1))
                      (EQUAL (CDR (LAST (CADR TERM1)))
                             0))
                 (EVG-OCCUR-LEGAL-CHAR-CODE-SEQ (CADR TERM1)
                        (CADR TERM2)))
```

```
            (T (EVG-OCCUR-OTHER (CADR TERM1)
                      (CADR TERM2]
         (T NIL)))
      ((EQUAL TERM1 TERM2)
       T)
      (T (for ARG in (FARGS TERM2) thereis (OCCUR TERM1 ARG])
```

(OCCUR-CNT

```
[LAMBDA (TERM1 TERM2)                                   (* kbr: "19-Oct-85 16:31")

        (* Return a lower bound on the number of times TERM1 occurs in TERM2.
        We do not go inside of QUOTEs in TERM2. *)

  (COND
     ((EQUAL TERM1 TERM2)
      1)
     ((VARIABLEP TERM2)
      0)
     ((FQUOTEP TERM2)
      0)
     (T (for ARG in (FARGS TERM2) sum (OCCUR-CNT TERM1 ARG])
```

(OCCUR-LST

```
[LAMBDA (X LST)                                         (* kbr: "19-Oct-85 16:31")
  (for Y in LST thereis (OCCUR X Y])
```

(ONE-WAY-UNIFY

```
[LAMBDA (TERM1 TERM2)                                   (* kbr: "19-Oct-85 16:31")
  (SETQ UNIFY-SUBST NIL)
  (ONE-WAY-UNIFY1 TERM1 TERM2])
```

(ONE-WAY-UNIFY-LIST

```
[LAMBDA (TERM1-LIST TERM2-LIST)                         (* kbr: "19-Oct-85 16:31")
                                                        (* Like ONE-WAY-UNIFY except operates on lists of terms.
                                                        *)
  (SETQ UNIFY-SUBST NIL)
  (for TERM1 in TERM1-LIST as TERM2 in TERM2-LIST always (ONE-WAY-UNIFY1 TERM1 TERM2])
```

(ONE-WAY-UNIFY1

```
[LAMBDA (TERM1 TERM2)                                   (* kbr: "19-Oct-85 16:31")
  (LET (OLD-ALIST)
       (SETQ COMMUTED-EQUALITY-FLG NIL)
       (SETQ OLD-ALIST UNIFY-SUBST)
       (COND
          ((ONE-WAY-UNIFY11 TERM1 TERM2)
           T)
          (T (SETQ UNIFY-SUBST OLD-ALIST)
             NIL])
```

(ONE-WAY-UNIFY11

```
[LAMBDA (TERM1 TERM2)                                   (* kbr: "19-Oct-85 16:31")
  (COND
     [(VARIABLEP TERM1)
      (COND
         ((SETQ TEMP-TEMP (ASSOC TERM1 UNIFY-SUBST))
          (EQUAL (CDR TEMP-TEMP)
                 TERM2))
         (T (SETQ UNIFY-SUBST (CONS (CONS TERM1 TERM2)
                                    UNIFY-SUBST]
     ((FQUOTEP TERM1)

        (* Since TERM1 is the only one whose variables we instantiate, and is constant, and all terms are in the QUOTE-normal
        form discussed in CONS-TERM, these two terms unify iff they are EQUAL.
        *)

      (EQUAL TERM1 TERM2))
     ((VARIABLEP TERM2)
      NIL)
     [(EQ (FFN-SYMB TERM1)
          (FN-SYMB TERM2))
      (COND
         [(EQ (FFN-SYMB TERM1)
              (QUOTE EQUAL))
          (LET ((SAVED-UNIFY-SUBST UNIFY-SUBST))
               (COND
                  ((AND (ONE-WAY-UNIFY11 (FARGN TERM1 1)
                                         (FARGN TERM2 1))
                        (ONE-WAY-UNIFY11 (FARGN TERM1 2)
                                         (FARGN TERM2 2)))
                   T)
                  (T (SETQ UNIFY-SUBST SAVED-UNIFY-SUBST)
```

```
                    (AND (ONE-WAY-UNIFY11 (FARGN TERM1 2)
                                          (FARGN TERM2 1))
                         (ONE-WAY-UNIFY11 (FARGN TERM1 1)
                                          (FARGN TERM2 2))
                         (SETQ COMMUTED-EQUALITY-FLG T]
          (T (for ARG1 in (FARGS TERM1) as ARG2 in (SARGS TERM2) always (ONE-WAY-UNIFY11 ARG1 ARG2]
       (T NIL])
```

(ONEIFY

```
  [LAMBDA (TERM TESTS)                                          (* kbr: "19-Oct-85 16:31")
    (COND
       ((VARIABLEP TERM)
        TERM)
       ((FQUOTEP TERM)
        TERM)
       (T (SELECTQ (FFN-SYMB TERM)
              (IF (LIST (QUOTE *2*IF)
                        (ONEIFY-TEST (FARGN TERM 1)
                                     TESTS)
                        (ONEIFY (FARGN TERM 2)
                                (ONEIFY-ASSUME-TRUE (FARGN TERM 1)
                                                    TESTS))
                        (ONEIFY (FARGN TERM 3)
                                (ONEIFY-ASSUME-FALSE (FARGN TERM 1)
                                                     TESTS))))
              (CONS (LIST (QUOTE CONS)
                          (ONEIFY (FARGN TERM 1)
                                  TESTS)
                          (ONEIFY (FARGN TERM 2)
                                  TESTS)))
              (CAR [COND
                      ((IMPLIES? TESTS (FCONS-TERM* (QUOTE LISTP)
                                                    (FARGN TERM 1)))
                       (LIST (QUOTE CAR)
                             (ONEIFY (FARGN TERM 1)
                                     TESTS)))
                      (T (LIST (QUOTE *1*CAR)
                               (ONEIFY (FARGN TERM 1)
                                       TESTS])
              (CDR [COND
                      ((IMPLIES? TESTS (FCONS-TERM* (QUOTE LISTP)
                                                    (FARGN TERM 1)))
                       (LIST (QUOTE CDR)
                             (ONEIFY (FARGN TERM 1)
                                     TESTS)))
                      (T (LIST (QUOTE *1*CDR)
                               (ONEIFY (FARGN TERM 1)
                                       TESTS])
              ((LISTP EQUAL)
                   (LIST (QUOTE *2*IF)
                         (ONEIFY-TEST TERM TESTS)
                         (KWOTE *1*T)
                         (KWOTE *1*F)))
              (CONS (PACK (LIST STRING-WEIRD (FFN-SYMB TERM)))
                    (for ARG in (FARGS TERM) collect (ONEIFY ARG TESTS])
```

(ONEIFY-ASSUME-FALSE

```
  [LAMBDA (TEST TESTS)                                          (* kbr: "19-Oct-85 16:31")
    (CONS (NEGATE-LIT TEST)
          TESTS])
```

(ONEIFY-ASSUME-TRUE

```
  [LAMBDA (TEST TESTS)                                          (* kbr: "19-Oct-85 16:31")
    (COND
       ((NLISTP TEST)
        (CONS TEST TESTS))
       ((FQUOTEP TEST)
        (CONS TEST TESTS))
       ((AND (EQ (FFN-SYMB TEST)
                 (QUOTE IF))
             (EQUAL (FARGN TEST 3)
                    FALSE))
        (ONEIFY-ASSUME-TRUE (FARGN TEST 1)
               (ONEIFY-ASSUME-TRUE (FARGN TEST 2)
                      TESTS)))
       (T (CONS TEST TESTS])
```

(ONEIFY-TEST

```
  [LAMBDA (TERM TESTS)                                          (* kbr: " 6-Jul-86 09:55")
    (COND
       ((VARIABLEP TERM)
        (LIST (QUOTE NEQ)
              TERM
```

```
          (QUOTE *1*F)))
       ((FQUOTEP TERM)
        (NOT (EQ (CADR TERM)
                 *1*F)))
       (T (SELECTQ (FFN-SYMB TERM)
            (IF (LIST (QUOTE *2*IF)
                      (ONEIFY-TEST (FARGN TERM 1)
                                   TESTS)
                      (ONEIFY-TEST (FARGN TERM 2)
                                   (ONEIFY-ASSUME-TRUE (FARGN TERM 1)
                                                       TESTS))
                      (ONEIFY-TEST (FARGN TERM 3)
                                   (ONEIFY-ASSUME-FALSE (FARGN TERM 1)
                                                        TESTS))))
            (LISTP
```

(* We have to COPY the result of this SUB-PAIR so we do not have two EQ occurrences of the arg in the X positions.
*)

```
                [COPY (SUB-PAIR (QUOTE (X *1*SHELL-QUOTE-MARK))
                                (LIST (ONEIFY (FARGN TERM 1)
                                              TESTS)
                                      (KWOTE *1*SHELL-QUOTE-MARK))
                                (QUOTE (*2*IF (LISTP X)
                                              (NEQ (CAR X)
                                                   *1*SHELL-QUOTE-MARK)
                                              NIL]))
            (EQUAL [COND
                       ([AND (QUOTEP (FARGN TERM 1))
                             (LITATOM (CADR (FARGN TERM 1]
                        (LIST (QUOTE EQ)
                              (ONEIFY (FARGN TERM 2)
                                      TESTS)
                              (FARGN TERM 1)))
                       ([AND (QUOTEP (FARGN TERM 2))
                             (LITATOM (CADR (FARGN TERM 2]
                        (LIST (QUOTE EQ)
                              (ONEIFY (FARGN TERM 1)
                                      TESTS)
                              (FARGN TERM 2)))
                       (T (LIST (QUOTE EQUAL)
                                (ONEIFY (FARGN TERM 1)
                                        TESTS)
                                (ONEIFY (FARGN TERM 2)
                                        TESTS])
            (LIST (QUOTE NEQ)
                  (ONEIFY TERM TESTS)
                  (QUOTE *1*F])
```

(OPTIMIZE-COMMON-SUBTERMS

```
[LAMBDA (FORM)                                                    (* kbr: "26-Oct-85 14:01")
  (PROG (SUBTERMS COMMONSUBTERMS PATHS DECISIONS OCC OCC1 OCC2 VAR-ALIST PARTI DOUBLE-TERMS NEW-FORM
         ISOLATED-CNT FIRST-CNT SECOND-CNT)
```

(* We are interested in evaluating certain LISP FORMs that are constructed out of variables
(i.e., SYMBOLPS (none of which begin with 2)), objects of the form
(QUOTE ITIMES) and FORMs which are proper lists beginning with SYMBOLPs which are either *2*IF or which have LAMBDA spread definitions. *2*IF behaves as though it had the MACRO
((X Y Z) (COND (X Y) (T Z)))%. We assume that no function associated with any function symbol has any effect on the LISP state. We assume that no variable is bound to the LITATOM *1*X.
We assume that there is no structure sharing among the non-QUOTE subexpressions of FORM.
Under these hypotheses, we generate and return a LISP form which when evaluated returns the the same value as would be returned by evaluating FORM. We intentionally ignore the fact that in LISP, if a variable is bound to NOBIND, the evaluation of that variable causes an error. This does not happened in compiled code.
*)

```
(SETQ SUBTERMS (INTERESTING-SUBTERMS FORM))
(SETQ COMMONSUBTERMS (for TERM in SUBTERMS when (for TERM2 in SUBTERMS
                                                      thereis (AND (NEQ TERM2 TERM)
                                                                   (EQUAL TERM2 TERM)))
                            collect TERM))
(COND
   ((NULL COMMONSUBTERMS)
    (RETURN FORM)))
(SETQ PARTI (PARTITION COMMONSUBTERMS))
(SETQ COMMONSUBTERMS (for PART in PARTI unless (for PART2 in PARTI thereis (PATH-POT-SUBSUMES PART2 PART)
                                                                          )
                            join (APPEND PART NIL)))
[SETQ PATHS (for P in (ALL-PATHS FORM) collect (REVERSE (CDR P]
```

(* For each term that occurs more than once in FORM, we calculate just how that occurrence occurs on the paths through the FORM. Given a path, we say the term occurs ISOLATED if no other EQUAL term occurs on the path.
We say the term appears FIRST on the path if some EQUAL term follows it but no EQUAL term precedes it.
We say the term appears SECOND on the path if it occurs on the path but the occurrence is not ISOLATED and is not FIRST, i.e., there is some EQUAL term that has a preceding occurrence on the path.
*)

```
        (for TERM in COMMONSUBTERMS do (SETQ ISOLATED-CNT 0)
                                       (SETQ FIRST-CNT 0)
                                       (SETQ SECOND-CNT 0)
                                       [for PATH in PATHS when (SETQ OCC (MEMB TERM PATH))
                                          do (SETQ OCC1 (MEMBER TERM PATH))
                                             (SETQ OCC2 (MEMBER TERM (CDR OCC)))
                                             (COND
                                                ((AND (EQ OCC OCC1)
                                                      (NULL OCC2))
                                                 (SETQ ISOLATED-CNT (ADD1 ISOLATED-CNT)))
                                                ((EQ OCC OCC1)
                                                 (SETQ FIRST-CNT (ADD1 FIRST-CNT)))
                                                (T (SETQ SECOND-CNT (ADD1 SECOND-CNT]

        (* For each common subterm, we now decide what to replace the term with.
        There are 5 alternatives. 1.0 (SET) Replace the term with (SETQ
        (v term) term) where (v term) is a LITATOM beginning with 2 and such that for all non-EQUAL common subterms s and t of
        FORM, (v t) is not (v s)%. 2.0 (VAR) Replace term with (v term)%.
        3.0 (TEST) Replace term with (*2*IF (EQ (v term) *1*X) term (v term))%.
        4.0 (TEST-AND-SET) Replace term with (*2*if (EQ (v term) *1*x)
        (SETQ (v term) term) (v term))%. 5.0 Do nothing. *)

                                       (COND
                                          [(GREATERP FIRST-CNT 0)
                                           (COND
                                              ((GREATERP SECOND-CNT 0)
                                               (SETQ DECISIONS (CONS (CONS TERM (QUOTE TEST-AND-SET))
                                                                     DECISIONS)))
                                              (T (SETQ DECISIONS (CONS (CONS TERM (QUOTE SET))
                                                                       DECISIONS]
                                          [(GREATERP SECOND-CNT 0)
                                           (COND
                                              ((GREATERP ISOLATED-CNT 0)
                                               (SETQ DECISIONS (CONS (CONS TERM (QUOTE TEST))
                                                                     DECISIONS)))
                                              (T

        (* This is the only decision that deserves serious consideration.
        All of the other decisions obviously result in correct behavior. Here, we know that the term always occurs second.
        Thus we are guaranteed that on every path to term, an equal term will have previously been evaluated.
        For each such path, some EQUAL term will have a FIRST occurrence and every term that is ever first is always SET or
        TEST-AND-SET. *)

                                                  (SETQ DECISIONS (CONS (CONS TERM (QUOTE VAR))
                                                                        DECISIONS]
                                          (T NIL)))

        (* We now construct a list of the common subterms, omitting EQUAL duplications.
        We wish to associate a unique variable *2*TEMPi, for some i, with all EQUAL common subterms.
        *)

        (SETQ DOUBLE-TERMS (for D in DECISIONS bind LOOP-ANS do (SETQ LOOP-ANS (ADD-TO-SET (CAR D)
                                                                                           LOOP-ANS))
                                finally (RETURN LOOP-ANS)))
        [SETQ VAR-ALIST (for D in DOUBLE-TERMS as I from 1 collect (CONS D (PACK (CONS STRING-WEIRD2
                                                                                     (CONS (QUOTE TEMP)
                                                                                           (UNPACK I]
                                                                   (* Using DOUBLE-TERMS and VAR-ALIST, COMMON-SWEEP
    now carries out the DECISIONS. *)
        (SETQ NEW-FORM (COMMON-SWEEP FORM))
        (RETURN (LIST (QUOTE LET)
                      [for PAIR in VAR-ALIST collect (LIST (CDR PAIR)
                                                           (QUOTE (QUOTE *1*X]
                      NEW-FORM])
```

(PARTITION

```
  [LAMBDA (L)                                                  (* kbr: "19-Oct-85 16:31")

        (* Returns a list of lists. Each member of L is a MEMBer of exactly one the of list of lists.
        Each MEMBer of each list is a MEMBer of L. *)

    (LET (POT TEMP)
         [for L1 in L do (SETQ TEMP (SASSOC L1 POT))
                         (COND
                            ((NULL TEMP)
                             (SETQ POT (CONS (LIST L1)
                                             POT)))
                            (T (NCONC1 TEMP L1]
         POT])
```

(PARTITION-CLAUSES

```
  [LAMBDA (LST)                                                (* kbr: "20-Oct-85 19:37")
    (LET (ALIST FLG POCKETS N)
         (SETQ LST (for CL in LST collect (CONS NIL CL)))
```

```
        [for PAIR in LST do (for LIT in (CDR PAIR) do (SETQ FLG (BM-MATCH LIT (NOT LIT)))
                                                      (SETQ TEMP-TEMP (SASSOC LIT ALIST))
                                                      (COND
                                                         ((NULL TEMP-TEMP)
                                                          (SETQ TEMP-TEMP (LIST LIT FLG PAIR))
                                                          (SETQ ALIST (CONS TEMP-TEMP ALIST)))
                                                         ((EQUAL (CADR TEMP-TEMP)
                                                                 0)
                                                          NIL)
                                                         ((NEQ FLG (CADR TEMP-TEMP))
                                                          (RPLACA (CDR TEMP-TEMP)
                                                                  0))
                                                         (T (RPLACD (CDR TEMP-TEMP)
                                                                    (CONS PAIR (CDDR TEMP-TEMP]
        (SETQ N (LENGTH LST))
        (for PAIR in ALIST when (AND (NOT (EQUAL (CADR PAIR)
                                                 0))
                                     (NOT (IEQP (LENGTH (CDDR PAIR))
                                                N)))
           do (SETQ POCKETS (CONS (for PAIR in (CDDR PAIR) unless (CAR PAIR) collect (PROGN (RPLACA PAIR T)
                                                                                            (CDR PAIR)))
                                  POCKETS)))
        [COND
           ((SETQ TEMP-TEMP (for PAIR in LST unless (CAR PAIR) collect (CDR PAIR)))
            (SETQ POCKETS (CONS TEMP-TEMP POCKETS]
        POCKETS])
```

(PATH-ADD-TO-SET

```
  [LAMBDA (X Y)                                                    (* kbr: "19-Oct-85 16:31")
    (COND
       ((for Y1 in Y thereis (PATH-EQ X Y1))
        Y)
       (T (CONS X Y])
```

(PATH-EQ

```
  [LAMBDA (X Y)                                                    (* kbr: "19-Oct-85 16:31")
    (AND (IEQP (LENGTH X)
               (LENGTH Y))
         (for X1 in X as Y1 in Y always (EQ X1 Y1])
```

(PATH-POT-SUBSUMES

```
  [LAMBDA (LARGER SMALLER)                                         (* kbr: "19-Oct-85 16:31")
    (for I from 1 to (SUB1 (LENGTH (CAR LARGER))) thereis (for S in SMALLER
                                                             always (for L in LARGER
                                                                       thereis (EQ S (FARGN L I])
```

(PATH-UNION

```
  [LAMBDA (X Y)                                                    (* kbr: "19-Oct-85 16:31")
    (NCONC (for X1 in X unless (for Y1 in Y thereis (PATH-EQ X1 Y1)) collect X1)
           Y])
```

(PEGATE-LIT

```
  [LAMBDA (TERM)                                                   (* kbr: "19-Oct-85 16:31")
    (COND
       ((FALSE-NONFALSEP TERM)
        (COND
           (DEFINITELY-FALSE FALSE)
           (T TRUE)))
       (T TERM])
```

(PETITIO-PRINCIPII

```
  [LAMBDA (EVENTS ALL-FLG FAILURE-ACTION DETACH-FLG DO-NOT-PRINT-FIRST-EVENT-FLG DO-NOT-PRINT-DATE-LINE-FLG)
                                                                   (* kbr: "19-Oct-85 16:31")
    (REDO-UNDONE-EVENTS (for X in EVENTS collect (COND
                                                    ((EQ (CAR X)
                                                         (QUOTE PROVE-LEMMA))
                                                     (LIST (QUOTE ADD-AXIOM)
                                                           (CADR X)
                                                           (CADDR X)
                                                           (CADDDR X)))
                                                    (T X)))
          ALL-FLG FAILURE-ACTION DETACH-FLG DO-NOT-PRINT-FIRST-EVENT-FLG DO-NOT-PRINT-DATE-LINE-FLG])
```

(PICK-HIGH-SCORES

```
  [LAMBDA (CANDLST)                                                (* kbr: "19-Oct-85 16:31")
                                                                   (* Returns the list of elements of CAND-LIST tied for the highest
    CAR. *)
    (MAXIMAL-ELEMENTS CANDLST (FUNCTION (LAMBDA (CAND)
                                          (fetch (CANDIDATE SCORE) of CAND])
```

(PIGEON-HOLE

```
[LAMBDA (PIGEONS HOLES FN DO-NOT-CROWD-FLG DO-NOT-SMASH-FLG)   (* kbr: "19-Oct-85 16:31")
  (LET (PAIRLST)
       (SETQ PAIRLST (for X in HOLES collect (CONS NIL X)))
       (COND
          [(PIGEON-HOLE1 PIGEONS PAIRLST FN DO-NOT-CROWD-FLG DO-NOT-SMASH-FLG)
           (COND
              (DO-NOT-SMASH-FLG HOLES)
              (T (for PAIR in PAIRLST collect (CDR PAIR]
          (T NIL])
```

(PIGEON-HOLE-IN-ALL-POSSIBLE-WAYS

```
[LAMBDA (PIGEONS HOLES FN DO-NOT-CROWD-FLG)                    (* kbr: "20-Oct-85 19:25")
  (LET (ANS POT X)
       (COND
          [[for PIGEON in PIGEONS always (PROGN (SETQ POT (for HOLE in HOLES when (SETQ X (APPLY* FN PIGEON HOLE)
                                                                                   )
                                                                                collect (CONS HOLE X)))
                                                (COND
                                                   (POT (SETQ ANS (NCONC1 ANS POT)))
                                                   (T NIL]
           (COND
              ((AND DO-NOT-CROWD-FLG (NOT (NO-CROWDINGP ANS [FUNCTION (LAMBDA (X Y)
                                                                       (EQ (CAR X)
                                                                           (CAR Y]
                                                 NIL)))
               NIL)
              (T (UNION-EQUAL (for X in ANS join (for Y in X collect (CDR Y)))
                              (for HOLE in HOLES unless (for X in ANS thereis (ASSOC HOLE X)) collect HOLE]
          (T NIL])
```

(PIGEON-HOLE1

```
[LAMBDA (PIGEONS PAIRLST FN DO-NOT-CROWD-FLG DO-NOT-SMASH-FLG) (* kbr: "19-Oct-85 16:31")
  (LET (TEMP OLD-FLG OLD-HOLE)
       (COND
          ((NULL PIGEONS)
           T)
          ((for PAIR in PAIRLST unless (AND DO-NOT-CROWD-FLG (CAR PAIR))
              thereis (COND
                         ((SETQ TEMP (APPLY* FN (CAR PIGEONS)
                                                (CDR PAIR)))
                          (SETQ OLD-FLG (CAR PAIR))
                          (SETQ OLD-HOLE (CDR PAIR))
                          (OR DO-NOT-SMASH-FLG (RPLACD PAIR TEMP))
                          (RPLACA PAIR T)
                          (COND
                             ((PIGEON-HOLE1 (CDR PIGEONS)
                                     PAIRLST FN DO-NOT-CROWD-FLG DO-NOT-SMASH-FLG)
                              T)
                             (T (RPLACD PAIR OLD-HOLE)
                                (RPLACA PAIR OLD-FLG)
                                NIL)))
                         (T NIL)))
           T)
          (T NIL])
```

(PLUSJOIN

```
[LAMBDA (LST)                                                  (* kbr: "19-Oct-85 16:31")
  (COND
     ((NULL LST)
      (QUOTE (ZERO)))
     ((NULL (CDR LST))
      (CAR LST))
     (T (FCONS-TERM* (QUOTE PLUS)
                (CAR LST)
                (PLUSJOIN (CDR LST])
```

(POLY-MEMBER

```
[LAMBDA (POLY LST)                                             (* kbr: "19-Oct-85 16:31")
  (for POLY2 in LST thereis (AND (EQUAL (fetch (POLY CONSTANT) of POLY)
                                        (fetch (POLY CONSTANT) of POLY2))
                                 (EQUAL (fetch (POLY ALIST) of POLY)
                                        (fetch (POLY ALIST) of POLY2])
```

(POP-CLAUSE-SET

```
[LAMBDA NIL                                                    (* kbr: "20-Apr-86 18:42")
  (PROG (CL-SET TEMP)
    TOP [COND
           ((NULL STACK)
            (WRAPUP T))
           ((EQ (CAAR STACK)
```

```
                (QUOTE BEING-PROVED))
            (SETQ TEMP (CADR (CAR STACK)))
            (SETQ STACK (CDR STACK))
            (IO (QUOTE POP)
                TEMP NIL NIL (LIST (GET-STACK-NAME STACK)))
            (GO TOP))
           (T (SETQ CL-SET (CADR (CAR STACK)))
              (SETQ STACK (CDR STACK]
      (COND
         ([for STACK-TAIL on STACK do (COND
                                        ((for CL2 in CL-SET always (for CL1 in (CADR (CAR STACK-TAIL))
                                                                        thereis (SUBSUMES CL1 CL2)))
                                         (COND
                                            ((EQ (CAR (CAR STACK-TAIL))
                                                 (QUOTE BEING-PROVED))
                                             [IO (QUOTE SUBSUMED-BY-PARENT)
                                                 CL-SET NIL NIL (LIST (GET-STACK-NAME STACK)
                                                                      (GET-STACK-NAME (CDR STACK-TAIL))
                                                                      (CADR (CAR STACK-TAIL]
                                             (WRAPUP NIL))
                                            (T [IO (QUOTE SUBSUMED-BELOW)
                                                   CL-SET NIL NIL (LIST (GET-STACK-NAME STACK)
                                                                        (GET-STACK-NAME (CDR STACK-TAIL))
                                                                        (CADR (CAR STACK-TAIL]
                                               (GO TOP]
          (GO TOP)))
      (SETQ STACK (CONS (LIST (QUOTE BEING-PROVED)
                              CL-SET)
                        STACK))
      (RETURN CL-SET])
```

(POP-LEMMA-FRAME

```
[LAMBDA NIL                                                    (* kbr: "19-Oct-85 22:23")
  (PROG1 (CAR LEMMA-STACK)
      [COND
         ((NULL LEMMA-STACK)
          (ERROR1 (PQUOTE (PROGN LEMMA-STACK IS TOO POOPED TO POP !))
                 NIL
                 (QUOTE HARD)))
         (T (SETQ LEMMA-STACK (CDR LEMMA-STACK])])
```

(POP-LINEARIZE-ASSUMPTIONS-FRAME

```
[LAMBDA NIL                                                    (* kbr: "19-Oct-85 22:02")
  (PROG1 (CAR LINEARIZE-ASSUMPTIONS-STACK)
      [COND
         ((NULL LINEARIZE-ASSUMPTIONS-STACK)
          (ERROR1 (PQUOTE (PROGN LINEARIZE-ASSUMPTIONS-STACK IS TOO POOPED TO POP !))
                 NIL
                 (QUOTE HARD)))
         (T (SETQ LINEARIZE-ASSUMPTIONS-STACK (CDR LINEARIZE-ASSUMPTIONS-STACK])])
```

(POPU

```
[LAMBDA NIL                                                    (* kbr: "19-Oct-85 16:31")
  (SETQ UNDONE-EVENTS (CAR UNDONE-EVENTS-STACK))
  (SETQ UNDONE-EVENTS-STACK (CDR UNDONE-EVENTS-STACK))
  UNDONE-EVENTS])
```

(POSSIBLE-IND-PRINCIPLES

```
[LAMBDA (TERM)                                                 (* kbr: "19-Oct-85 16:31")

        (* TERM is a non-QUOTE fn call and this fn returns all the induction principles suggested by it.
        See FLESH-OUT-IND-PRIN for the form of an induction prin. *)

  (LET (MACHINE FORMALS QUICK-BLOCK-INFO MASK)
       [SETQ FORMALS (CADR (GETPROP (FFN-SYMB TERM)
                                    (QUOTE SDEFN]
       (SETQ QUICK-BLOCK-INFO (GETPROP (FFN-SYMB TERM)
                                       (QUOTE QUICK-BLOCK-INFO)))
       (SETQ MACHINE (GETPROP (FFN-SYMB TERM)
                              (QUOTE INDUCTION-MACHINE)))
       (COND
          ((DISABLEDP (FFN-SYMB TERM))
           NIL)
          (T (for J in (GETPROP (FFN-SYMB TERM)
                                (QUOTE JUSTIFICATIONS))
                when (SETQ MASK (SOUND-IND-PRIN-MASK TERM J FORMALS QUICK-BLOCK-INFO))
                collect (FLESH-OUT-IND-PRIN TERM FORMALS MACHINE J MASK QUICK-BLOCK-INFO])
```

(POSSIBLY-NUMERIC

```
[LAMBDA (TERM)                                                 (* kbr: "19-Oct-85 16:31")
  (LET ((TYPE-ALIST (OR HEURISTIC-TYPE-ALIST TYPE-ALIST)))
       (IEQP (TYPE-SET TERM)
```

```
              TYPE-SET-NUMBERS])
```

(**POWER-EVAL**
```
  [LAMBDA (L B)                                                    (* kbr: "25-Oct-85 18:13")
    (COND
       ((NLISTP L)
        0)
       (T (PLUS (CAR L)
                (TIMES B (POWER-EVAL (CDR L)
                                     B])
```

(**POWER-REP**
```
  [LAMBDA (N B)                                                    (* kbr: "19-Oct-85 16:31")
    (COND
       ((LESSP N B)
        (LIST N))
       (T (CONS (REMAINDER N B)
                (POWER-REP (QUOTIENT N B)
                       B])
```

(**PPC**
```
  [LAMBDA (CL)                                                     (* kbr: "19-Oct-85 16:31")
    (BM-PPR (PRETTYIFY-CLAUSE CL)
            NIL)
    NIL])
```

(**PPE**
```
  [LAMBDA (X)                                                      (* kbr: "19-Oct-85 16:31")
    (PPE-LST (LIST X])
```

(**PPE-LST**
```
  [LAMBDA (X)                                                      (* kbr: "19-Oct-85 16:31")
    (for NAME in X do (ITERPRI NIL)
                      (BM-PPR [OR (GETPROP NAME (QUOTE EVENT))
                                  [AND (GETPROP NAME (QUOTE MAIN-EVENT))
                                       (LIST (QUOTE *****)
                                             NAME
                                             (QUOTE IS)
                                             (QUOTE A)
                                             (QUOTE SATELLITE)
                                             (QUOTE OF)
                                             (GETPROP (GETPROP NAME (QUOTE MAIN-EVENT))
                                                      (QUOTE EVENT]
                                  (CONS (QUOTE *****)
                                        (CONS NAME (QUOTE (IS NEITHER AN EVENT NOR SATELLITE]
                              NIL)
                      (ITERPRI NIL])
```

(**BM-PPR**
```
  [LAMBDA (FMLA PPRFILE)                                           (* kbr: "19-Oct-85 16:31")
    (LET (LEFTMARGINCHAR)
         (PPRIND FMLA 0 0 PPR-MACRO-LST PPRFILE)
         NIL])
```

(**PPRINDENT**
```
  [LAMBDA (TERM LEFTMARGIN RPARCNT FILE)                           (* kbr: "19-Oct-85 16:31")
    (COND
       ((IGREATERP (IPOSITION FILE NIL NIL)
                   LEFTMARGIN)
        (ITERPRISPACES LEFTMARGIN FILE))
       (T (TABULATE LEFTMARGIN FILE)))
    (PPRIND TERM LEFTMARGIN (OR RPARCNT 0)
           PPR-MACRO-LST FILE])
```

(**PPSD**
```
  [LAMBDA (X)                                                      (* kbr: "19-Oct-85 16:31")
    (PPSD-LST (LIST X])
```

(**PPSD-LST**
```
  [LAMBDA (X)                                                      (* kbr: "19-Oct-85 16:31")
    (for FNNAME in X do (BM-PPR (LIST FNNAME (OR (GETPROP FNNAME (QUOTE SDEFN))
                                                 (QUOTE UNDEFINED)))
                            NIL)
                        (ITERPRI NIL)
                        (ITERPRI NIL])
```

(**PREPROCESS**

```
[LAMBDA (TERM)                                                    (* kbr: "19-Oct-85 16:31")

          (* Returns a set of clauses whose conjunction is equivalent to TERM and sets ABBREVIATIONS-USED to the list of fn
          symbols and rewrite rules applied. *)

    (LET (TYPE-ALIST)
         (SETQ ABBREVIATIONS-USED NIL)
         (CLAUSIFY-INPUT (EXPAND-ABBREVIATIONS TERM NIL])
```

(PREPROCESS-HYPS

```
[LAMBDA (HYPS)                                                    (* kbr: "19-Oct-85 20:11")
                                                                  (* Expand NLISTP and NOT ZEROP hyps.
                                                                  *)
    (for HYP in HYPS bind X join (COND
                                    [(BM-MATCH HYP (NOT (ZEROP X)))
                                     (LIST (FCONS-TERM* (QUOTE NUMBERP)
                                                        X)
                                           (FCONS-TERM* (QUOTE NOT)
                                                  (FCONS-TERM* (QUOTE EQUAL)
                                                               X ZERO]
                                    [(BM-MATCH HYP (NLISTP X))
                                     (LIST (FCONS-TERM* (QUOTE NOT)
                                                  (FCONS-TERM* (QUOTE LISTP)
                                                               X]
                                    (T (LIST HYP])
```

(PRETTYIFY-CLAUSE

```
[LAMBDA (CL)                                                      (* kbr: "19-Oct-85 16:31")
  (COND
     ((NULL CL)
      FALSE)
     ((NULL (CDR CL))
      (CAR CL))
     ((NULL (CDDR CL))
      (LIST (QUOTE IMPLIES)
            (DUMB-NEGATE-LIT (CAR CL))
            (CADR CL)))
     (T (LIST (QUOTE IMPLIES)
              [CONS (QUOTE AND)
                    (for TAIL on CL unless (NULL (CDR TAIL)) collect (DUMB-NEGATE-LIT (CAR TAIL]
              (CAR (LAST CL])
```

(PRETTYIFY-LISP

```
[LAMBDA (X)                                                       (* kbr: "19-Oct-85 16:31")
  (REMOVE-*2*IFS (INTRODUCE-ANDS (INTRODUCE-LISTS X])
```

(PRIMITIVE-RECURSIVEP

```
[LAMBDA (FNNAME)                                                  (* kbr: "19-Oct-85 16:31")
  (LET (FORMALS)
       [SETQ FORMALS (CADR (GETPROP FNNAME (QUOTE SDEFN]
       (COND
          ((DISABLEDP FNNAME)
           T)
          (T (for X in (GETPROP FNNAME (QUOTE INDUCTION-MACHINE))
                always (for CASE in (fetch (TESTS-AND-CASES CASES) of X)
                          always (for VAR in FORMALS as TERM in CASE always (SHELL-DESTRUCTOR-NESTP VAR TERM])
```

(PRIMITIVEP

```
[LAMBDA (TERM)                                                    (* kbr: "19-Oct-85 16:31")
  (OR (VARIABLEP TERM)
      (FQUOTEP TERM)
      (AND (OR (NULL (GETPROP (FFN-SYMB TERM)
                              (QUOTE SDEFN)))
               (DISABLEDP (FFN-SYMB TERM))
               (EQ (FFN-SYMB TERM)
                   (QUOTE NOT)))
           (for ARG in (FARGS TERM) always (PRIMITIVEP ARG])
```

(PRINT-STACK

```
[LAMBDA (Y)                                                       (* kbr: "19-Oct-85 16:31")
  (for X on Y by (QUOTE CADR) do (IPRINT (CAR X)
                                         T])
```

(PRINT-STATS

```
[LAMBDA (ELAPSED IO FILE)                                         (* kbr: "22-Oct-85 16:20")
  (ITERPRI FILE)
  (IPRINC "(" FILE)
  (ISPACES 1 FILE)
  (IPRINC ELAPSED FILE)
  (ISPACES 1 FILE)
```

```
    (IPRINC IO FILE)
    (ISPACES 1 FILE)
    (IPRINC ")" FILE])
```

(PRINT-TO-DISPLAY

```
  [LAMBDA (MSG1 MSG2 MSG3)                                  (* kbr: "20-Oct-85 17:24")
    (COND
       ((NULL LEMMA-DISPLAY-FLG))
       ((EQ LEMMA-DISPLAY-FLG (QUOTE MODEL33))
        (for I from 1 to (SUB1 (STACK-DEPTH LEMMA-STACK)) do

          (* STACK-DEPTH starts at 1 and we want 0 leading chars at first.
          In LEMMA-DISPLAY mode T we use STACK-DEPTH because lines on the screen are numbered from 1.0 The CONSTANT
          below is just vertical bar, but if typed explicitly it is brought up from emacs incorrectly.
          *)

                                                            (IPRINC "/" T))
        (IPRINC "*" T)
        (IPRINC MSG1 T)
        (COND
           (MSG2 (ISPACES 1 T)
                 (IPRINC MSG2 T)))
        (COND
           (MSG3 (IPRINC MSG3 T)))
        (ITERPRI T))
       (T (PUT-CURSOR 1 (STACK-DEPTH LEMMA-STACK))
          (ERASE-EOP)
          (IPRINC MSG1 T)
          (COND
             (MSG2 (ISPACES 1 T)
                   (IPRINC MSG2 T)))
          (COND
             (MSG3 (IPRINC MSG3 T])
```

(PROCESS-EQUATIONAL-POLYS

```
  [LAMBDA (CL HIST POT-LST)                                 (* kbr: "19-Oct-85 16:31")

          (* Deduce from POT-LST all the interesting equations in it and add them to CL unless they have already been generated
          and recorded in HIST. This function has no effect on the lemma and assumptions stacks but sets the globals
          LEMMAS-USED-BY-LINEAR and LINEAR-ASSUMPTIONS if it changes CL.
          When it adds an equation to CL it adds an entry to LEMMAS-USED-BY-LINEAR that will ultimately be copied into the new
          hist for the clause. The entry is of the form ((FIND-EQUATIONAL-POLYS lhs)) --
          the apparently redundant level of parens is there to insure that the element cannot be confused with a term.
          Thus, when it is thrown into the list PROCESS-HIST with lemma names and literals used, we can filter out the literals.
          SIMPLIFY-CLAUSE handles this filtering above us. FIND-EQUATIONAL-POLY is the function that adds such entries to
          LEMMAS-USED-BY-LINEAR and that looks for them in the HIST.
          *)

    (SETQ LEMMAS-USED-BY-LINEAR NIL)
    (SETQ LINEAR-ASSUMPTIONS NIL)
    [for POT in POT-LST bind PAIR when (SETQ PAIR (FIND-EQUATIONAL-POLY HIST POT))
       do                                                   (* When FIND-EQUATIONAL-POLY returns nonNIL it
                                                            side-effects the two global collection sites above.
                                                            *)
          (SETQ CL (COND
                      ([AND (VARIABLEP (CAR PAIR))
                            (NOT (OCCUR (CAR PAIR)
                                        (CDR PAIR]
                       (SUBST-VAR-LST (CDR PAIR)
                              (CAR PAIR)
                              CL))
                      ([AND (VARIABLEP (CDR PAIR))
                            (NOT (OCCUR (CDR PAIR)
                                        (CAR PAIR]
                       (SUBST-VAR-LST (CAR PAIR)
                              (CDR PAIR)
                              CL))
                      (T (CONS (FCONS-TERM* (QUOTE NOT)
                                            (FCONS-TERM* (QUOTE EQUAL)
                                                         (CAR PAIR)
                                                         (CDR PAIR)))
                               (SUBST-EXPR-LST (CDR PAIR)
                                      (CAR PAIR)
                                      CL]
    CL])
```

(PROPERTYLESS-SYMBOLP

```
  [LAMBDA (X)                                               (* kbr: "19-Oct-85 16:31")
    (OR (CAR-CDRP X)
        (MEMB X (QUOTE (NIL QUOTE LIST T F])
```

(PROVE

```
  [LAMBDA (FORM)                                            (* kbr: " 6-Jul-86 09:44")
```

```
    (PROG (THM CLAUSES VARS)
          (SETQ THM (TRANSLATE FORM))
          (SETQ CLAUSES (PREPROCESS THM))
          (SETUP FORM CLAUSES ABBREVIATIONS-USED)

          (* Basic control loop of the Boyer Moore theorem prover is to simplify the clause set, induct, repeat again.
          *)

      LOOP
          (SETQ VARS (for CL in CLAUSES bind LOOP-ANS do (SETQ LOOP-ANS (UNIONQ (ALL-VARS-LST CL)
                                                                                LOOP-ANS))
                                finally (RETURN LOOP-ANS)))
          (SETQ ELIM-VARIABLE-NAMES1 (SET-DIFF ELIM-VARIABLE-NAMES VARS))
          (SETQ GEN-VARIABLE-NAMES1 (SET-DIFF GEN-VARIABLE-NAMES VARS))
                                                                  (* Simplification = Simplify, Settle Down, Eliminate Destructors,
     Fertilize, Generalize, Eliminate Irrelevance *)
          (SIMPLIFY-LOOP CLAUSES)                                 (* Induction *)
          (SETQ CLAUSES (INDUCT (POP-CLAUSE-SET)))
          (GO LOOP])
```

(PROVE-TERMINATION

```
  [LAMBDA (FORMALS RM MACHINE)                                    (* kbr: "19-Oct-85 16:31")
    (SETQ PROVE-TERMINATION-LEMMAS-USED NIL)
    (for X in MACHINE always (COND
                               ((AND [SIMPLIFY-CLAUSE-MAXIMALLY (NCONC1 (for H in (fetch (TESTS-AND-CASE TESTS)
                                                                                         of X)
                                                                           collect (NEGATE-LIT H))
                                                                        (CONS-TERM
                                                                         (CAR RM)
                                                                         (LIST (SUB-PAIR-VAR FORMALS
                                                                                             (fetch (TESTS-AND-CASE CASE)
                                                                                                of X)
                                                                                             (CADR RM))
                                                                               (CADR RM]
                                     (NULL PROCESS-CLAUSES))
                                (SETQ PROVE-TERMINATION-LEMMAS-USED (UNION-EQUAL PROCESS-HIST
                                                                                 PROVE-TERMINATION-LEMMAS-USED))
                                T)
                               (T NIL])
```

(PROVEALL

```
  [LAMBDA (EVENT-LST DETACH-FLG FILENAME)                         (* kbr: "24-Oct-85 17:19")
    DETACH-FLG
    (SETQ FAILED-THMS NIL)
    (SETQ MASTER-ROOT-NAME (OR FILENAME (QUOTE PROVEALL)))
    [SETQ PROVE-FILE (AND NIL (OPENSTREAM (EXTEND-FILE-NAME MASTER-ROOT-NAME (QUOTE PROOFS))
                                          (QUOTE OUTPUT]
    [SETQ TTY-FILE (AND NIL (OPENSTREAM (EXTEND-FILE-NAME MASTER-ROOT-NAME (QUOTE TTY))
                                        (QUOTE OUTPUT]
    (REDO-UNDONE-EVENTS EVENT-LST T (QUOTE A)
           DETACH-FLG NIL NIL)
    (MAKE-LIB MASTER-ROOT-NAME])
```

(PUSH-CLAUSE-SET

```
  [LAMBDA (CL-SET)                                                (* kbr: "19-Oct-85 16:31")
    (SETQ STACK (CONS (LIST (QUOTE TO-BE-PROVED)
                            CL-SET)
                      STACK])
```

(PUSH-LEMMA

```
  [LAMBDA (ELE)                                                   (* kbr: "19-Oct-85 16:31")
    (COND
       ((MEMB ELE (CAR LEMMA-STACK))
        NIL)
       (T (RPLACA LEMMA-STACK (CONS ELE (CAR LEMMA-STACK)))
          NIL])
```

(PUSH-LEMMA-FRAME

```
  [LAMBDA NIL                                                     (* kbr: "19-Oct-85 22:23")
    (SETQ LEMMA-STACK (CONS NIL LEMMA-STACK))
    NIL])
```

(PUSH-LINEARIZE-ASSUMPTION

```
  [LAMBDA (ELE)                                                   (* kbr: "19-Oct-85 16:31")
    (RPLACA LINEARIZE-ASSUMPTIONS-STACK (ADD-TO-SET ELE (CAR LINEARIZE-ASSUMPTIONS-STACK)))
    NIL])
```

(PUSH-LINEARIZE-ASSUMPTIONS-FRAME

```
  [LAMBDA NIL                                                     (* kbr: "19-Oct-85 22:00")
    (SETQ LINEARIZE-ASSUMPTIONS-STACK (CONS NIL LINEARIZE-ASSUMPTIONS-STACK))
```

```
    NIL])
```

(PUSHU

```
  [LAMBDA NIL                                                    (* kbr: "19-Oct-85 16:31")
    (SETQ UNDONE-EVENTS-STACK (CONS UNDONE-EVENTS UNDONE-EVENTS-STACK))
    (SETQ UNDONE-EVENTS NIL])
```

(PUT-CURSOR

```
  [LAMBDA (X Y)                                                  (* kbr: "19-Oct-85 16:31")
    (CURSORPOS Y X])
```

(PUT-INDUCTION-INFO

```
  [LAMBDA (FNNAME FORMALS BODY RELATION-MEASURE-LST TAK0)        (* kbr: "19-Oct-85 16:31")

          (* If FNNAME is recursive we store JUSTIFICATIONS, INDUCTION-MACHINE, and QUICK-BLOCK-INFO properties.
          If only one JUSTIFICATION is stored and in it the RELATION is NIL then we did not establish termination.
          ALL-LEMMAS-USED is side-effected to contain lemma names used to clean up the INDUCTION-MACHINE.
          If TAK0 is nonNIL, then we are considering a reflexive definition.
          (tak0) IEQP body is allegedly the justifying lemma for the definition
          (fnname) IEQP body', where body' results from replacing all calls of tak0 with fnname.
          *)

    (PROG (T-MACHINE I-MACHINE)
          (SETQ T-MACHINE (TERMINATION-MACHINE (OR TAK0 FNNAME)
                               BODY NIL))
          (COND
             ((NULL T-MACHINE)
              (SETQ ALL-LEMMAS-USED NIL)
              (RETURN NIL)))
          (OR RELATION-MEASURE-LST (SETQ RELATION-MEASURE-LST (GUESS-RELATION-MEASURE-LST FORMALS T-MACHINE)))
          [ADD-FACT FNNAME (QUOTE JUSTIFICATIONS)
                 (OR (for RM in RELATION-MEASURE-LST when (PROVE-TERMINATION FORMALS RM T-MACHINE)
                        collect (create JUSTIFICATION
                                        SUBSET _ (ALL-VARS (CADR RM))
                                        MEASURE-TERM _ (CADR RM)
                                        RELATION _ (CAR RM)
                                        LEMMAS _ PROVE-TERMINATION-LEMMAS-USED))
                     (LIST (create JUSTIFICATION
                                   SUBSET _ FORMALS]
          (SETQ ALL-LEMMAS-USED NIL)

          (* We set ALL-LEMMAS-USED to NIL to forget the lemmas put there by PROVE so we can now accumulate the lemmas
          used by REMOVE-REDUNDANT-TESTS in INDUCTION-MACHINE.
          *)

          (SETQ I-MACHINE (INDUCTION-MACHINE FNNAME (COND
                                                      (TAK0 (SUBST-FN FNNAME TAK0 BODY))
                                                      (T BODY))
                               NIL))
          (ADD-FACT FNNAME (QUOTE INDUCTION-MACHINE)
                 I-MACHINE)
          (ADD-FACT FNNAME (QUOTE QUICK-BLOCK-INFO)
                 (QUICK-BLOCK-INFO FORMALS I-MACHINE))
          (RETURN NIL])
```

(PUT-LEVEL-NO

```
  [LAMBDA (FNNAME)                                               (* kbr: "24-Oct-85 15:48")
    (LET (BODY MAX)
         [SETQ BODY (CADDR (GETPROP FNNAME (QUOTE SDEFN]
         (SETQ MAX 0)
         (for FN in (ALL-FNNAMES BODY) when (NEQ FN FNNAME) do (SETQ MAX (IMAX (GET-LEVEL-NO FN)
                                                                              MAX)))
         (ADD-FACT FNNAME (QUOTE LEVEL-NO)
                (COND
                   ((FNNAMEP FNNAME BODY)
                    (ADD1 MAX))
                   (T MAX])
```

(PUT-TYPE-PRESCRIPTION

```
  [LAMBDA (NAME)                                                 (* kbr: "20-Apr-86 18:34")

          (* THIS FUNCTION WILL BE COMPLETELY UNSOUND IF TYPE-SET IS EVER REACHABLE FROM WITHIN IT.
          IN PARTICULAR, BOTH THE TYPE-ALIST AND THE TYPE-PRESCRIPTION FOR THE FN BEING PROCESSED ARE
          SET TO ONLY PARTIALLY ACCURATE VALUES AS THIS FN COMPUTES THE REAL TYPE-SET.
          *)

    (PROG (OLD-TYPE-PRESCRIPTION NEW-TYPE-PRESCRIPTION BODY FORMALS TYPE-ALIST ANS TEMP)
          (SETQ BODY (GETPROP NAME (QUOTE SDEFN)))
          (SETQ FORMALS (CADR BODY))
          (SETQ BODY (CADDR BODY))
          [SETQ TYPE-ALIST (for ARG in FORMALS collect (CONS ARG (CONS 0 (LIST ARG]
          (SETQ OLD-TYPE-PRESCRIPTION (CONS 0 (for ARG in FORMALS collect NIL)))
```

```
          (ADD-FACT NAME (QUOTE TYPE-PRESCRIPTION-LST)
                 (CONS NAME OLD-TYPE-PRESCRIPTION))
      LOOP
          (RPLACD [CAR (SETQ TEMP (GETPROP NAME (QUOTE TYPE-PRESCRIPTION-LST]
                 OLD-TYPE-PRESCRIPTION)

          (* It is very unusual to be mucking about with RPLACDs on data that is part of the event level abstraction.
          But by virtue of the fact that we know what the abstraction is and how it works --
          i.e., by violating the abstraction! -- we know what we're doing here.
          The TYPE-PRESCRIPTION-LST at this moment is a singleton list containing just the CONS added above.
          The CAR of that CONS is the name of the event that gave rise to the type prescription and the CDR is the type prescription.
          The RPLACD above smashes the type prescription in the CDR to a new that includes all the information contained in the
          current guess. The fundamental difficulty with destructively changing event level data arises because the ADD-SUB-FACT
          mechanism stores certain undo information about each added fact, and if you change the data without being aware of that,
          you might make the data inconsistent with the undoing information about it.
          But we know that all ADD-SUB-FACT stores in this case is the name of the lemma, that is, the CAR of the
          TYPE-PRESCRIPTION-NAME-AND-PAIR, and so by smashing the CDR we're consistently fooling it.
          *)

          (PUTPROP NAME (QUOTE TYPE-PRESCRIPTION-LST)
                 TEMP)

          (* Why do we both RPLACD the structure on the property list AND do the PUTPROP? The answer is that we are afraid that
          someday perhaps we will permit a SWAPOUT to occur anytime.
          Note that if that happened after we did the GETPROP but before the RPLACD happened we would lose.
          *)

          (SETQ ANS (DEFN-TYPE-SET BODY))
          [SETQ NEW-TYPE-PRESCRIPTION (CONS (CAR ANS)
                                            (for ARG in FORMALS collect (COND
                                                                          ((MEMB ARG (CDR ANS))
                                                                           T)
                                                                          (T NIL]
          (COND
             ((EQUAL OLD-TYPE-PRESCRIPTION NEW-TYPE-PRESCRIPTION)
              (RETURN NIL))
             ((AND (LOGSUBSETP (CAR NEW-TYPE-PRESCRIPTION)
                               (CAR OLD-TYPE-PRESCRIPTION))
                   (for FLG1 in (CDR NEW-TYPE-PRESCRIPTION) as FLG2 in (CDR OLD-TYPE-PRESCRIPTION)
                      always (OR (NOT FLG1)
                                 FLG2)))
              (ERROR1 (PQUOTE (PROGN AN UNEXPECTED SITUATION HAS ARISEN ! THE DEFN-TYPE-SET ITERATION STOPPED
                                     BECAUSE OF A PROPER SUBSET CHECK RATHER THAN THE EQUALITY OF THE OLD
                                       AND NEW TYPE SETS %.))
                      NIL
                      (QUOTE WARNING))
              (RETURN NIL)))
          [SETQ OLD-TYPE-PRESCRIPTION (CONS (LOGOR (CAR OLD-TYPE-PRESCRIPTION)
                                                   (CAR NEW-TYPE-PRESCRIPTION))
                                            (for FLG1 in (CDR OLD-TYPE-PRESCRIPTION) as FLG2
                                               in (CDR NEW-TYPE-PRESCRIPTION) collect (OR FLG1 FLG2]
          (GO LOOP])
```

(PUT0

```
  [LAMBDA (ATM PROP VAL HIGHER-PROPS)                                    (* kbr: "19-Oct-85 16:31")

          (* This function is conceptually hidden from the user of the lib file package.
          It may be called internally provided ATM is known to be in the PROP-HASH-ARRAY already.
          HIGHER-PROPS is the list of properties with higher priority than this one.
          If it is NIL this function assumes that it hasn't been computed by the caller and computes it.
          If the computation returns NIL, then PROP is not a member of LIB-PROPS and an error is caused.
          The reason this function does not just have three args and always compute HIGHER-PROPS --
          rather than allowing the caller to do it but not believing the caller when he says NIL --
          is that the main use of PUT0 is from PUT1, who must decide whether PROP is a member of LIB-PROPS before updating
          the hash array for ATM. So this implementation allows PUT1 to pass its answer down rather than require PUT0 to do the
          work again. At the moment, the only other calls of PUT0 do not bother to compute HIGHER-PROPS and just let PUT0 do it.
          But even if they did, and computed NIL, and did not check it but forced PUT0 to compute the NIL again, the time wasted is
          not important since we're going to then cause an error anyway.
          *)

    (OR HIGHER-PROPS (SETQ HIGHER-PROPS (MEMB PROP LIB-PROPS))
        (ERROR1 (PQUOTE (PROGN ATTEMPT TO PUT1 THE NON-LIB-PROPS PROPERTY (!PPR PROP NIL)
                               %.))
                (BINDINGS (QUOTE PROP)
                       PROP)
                (QUOTE HARD)))
    (SETPROPLIST ATM (PUT00 (GETPROPLIST ATM)
                            PROP VAL))
    VAL])
```

(PUT00

```
  [LAMBDA (TAIL PROP VAL)                                                (* kbr: "19-Oct-85 16:31")
    (COND
       ((NULL TAIL)
        (LIST PROP VAL))
```

```
       ((EQ PROP (CAR TAIL))
        (RPLACA (CDR TAIL)
               VAL)
        TAIL)
       [(MEMB (CAR TAIL)
              HIGHER-PROPS)
        (COND
           ((CDDR TAIL)
            (RPLACD (CDR TAIL)
                   (PUT00 (CDDR TAIL)
                          PROP VAL))
            TAIL)
           (T (NCONC TAIL (LIST PROP VAL]
       (T (CONS PROP (CONS VAL TAIL])
```

(PUT1

```
[LAMBDA (ATM VAL PROP)                                    (* kbr: "19-Oct-85 16:31")
```

(* Like PUTPROP except keeps the properties in the order specified by LIB-PROPS, causing an error if PROP is not on LIB-PROPS, and insures that ATM is a memb of LIB-ATOMS-WITH-PROPS *)

```
    (LET (HIGHER-PROPS)
         [COND
            ((NOT (BOUNDP (QUOTE LIB-PROPS)))
             (ERROR1 (PQUOTE (PROGN THEOREM PROVER NOT INITIALIZED %.))
                    NIL
                    (QUOTE HARD)))
            ((NULL (SETQ HIGHER-PROPS (MEMB PROP LIB-PROPS)))
             (ERROR1 (PQUOTE (PROGN ATTEMPT TO USE PUT1 TO STORE THE NON-LIB-PROPS PROPERTY (!PPR PROP NIL)
                                    %.))
                    (BINDINGS (QUOTE PROP)
                           PROP)
                    (QUOTE HARD)))
            ((NOT (MEMB ATM LIB-ATOMS-WITH-PROPS))
             (SETQ LIB-ATOMS-WITH-PROPS (CONS ATM LIB-ATOMS-WITH-PROPS]
           (PUT0 ATM PROP VAL HIGHER-PROPS])
```

(PUT1-LST

```
[LAMBDA (ATM PROPS)                                       (* kbr: "19-Oct-85 16:31")
```

(* PROPS is a list of the form (prop1 val1 prop2 val2 ...)%. This function is equivalent to doing (PUT1 ATM vali propi) for each i, but is faster. *)

```
    (SETPROPLIST ATM (APPEND PROPS (GETPROPLIST ATM])
```

(PUTD1

```
[LAMBDA (ATM EXPR)                                        (* kbr: "26-Oct-85 13:52")
```

(* If EXPR is NIL, remove ATM from LIB-ATOMS-WITH-DEFS and erase its function definition and EXPR property. If EXPR is non-NIL, add ATM to LIB-ATOMS-WITH-DEFS, make the compiled version of EXPR be the definition of ATM, and store EXPR under the EXPR prop. *)

```
    (COND
       ((NULL EXPR)
        (SETQ LIB-ATOMS-WITH-DEFS (DREMOVE ATM LIB-ATOMS-WITH-DEFS))
        (KILL-DEFINITION ATM))
       (T (SETQ LIB-ATOMS-WITH-DEFS (CONS ATM LIB-ATOMS-WITH-DEFS))
          (STORE-DEFINITION ATM EXPR])
```

(QUICK-BLOCK-INFO

```
[LAMBDA (FORMALS TESTS-AND-CASES-LST)                     (* kbr: "19-Oct-85 16:31")
```

(* Return a list of SELF-REFLEXIVE, or QUESTIONABLE, indicating how the corresponding arg position is changed in the calls enumerated. This is used to help quickly decide if a blocked formal can be tolerated in induction. *)

```
    (LET (BLOCK-TYPES)
         (SETQ BLOCK-TYPES (for VAR in FORMALS collect (QUOTE UN-INITIALIZED)))
         [for TESTS-AND-CASES in TESTS-AND-CASES-LST
            do (for CASE in (fetch (TESTS-AND-CASES CASES) of TESTS-AND-CASES)
                  do (for VAR in FORMALS as ARG in CASE as TAIL on BLOCK-TYPES
                        do (SELECTQ (CAR TAIL)
                                    (QUESTIONABLE NIL)
                                    (UN-INITIALIZED
                                        (RPLACA TAIL (QUICK-BLOCK-INFO1 VAR ARG)))
                                    (OR (EQ (CAR TAIL)
                                            (QUICK-BLOCK-INFO1 VAR ARG))
                                        (RPLACA TAIL (QUOTE QUESTIONABLE]
         BLOCK-TYPES])
```

(QUICK-BLOCK-INFO1

```
[LAMBDA (VAR TERM)                                        (* kbr: "19-Oct-85 16:31")
```

```
    (COND
       ((EQ VAR TERM)
        (QUOTE UNCHANGING))
       ((OCCUR VAR TERM)
        (QUOTE SELF-REFLEXIVE))
       (T (QUOTE QUESTIONABLE])
```

(QUICK-WORSE-THAN

```
  [LAMBDA (TERM1 TERM2)                                    (* kbr: "19-Oct-85 16:31")
    (COND
       [(VARIABLEP TERM2)
        (COND
           ((EQ TERM1 TERM2)
            NIL)
           (T (OCCUR TERM2 TERM1]
       ((FQUOTEP TERM2)
        (COND
           ((VARIABLEP TERM1)
            T)
           [(FQUOTEP TERM1)
            (GREATERP (FORM-COUNT-EVG (CADR TERM1))
                      (FORM-COUNT-EVG (CADR TERM2]
           (T T)))
       ((VARIABLEP TERM1)
        NIL)
       ((FQUOTEP TERM1)
        NIL)
       [(EQ (FFN-SYMB TERM1)
            (FFN-SYMB TERM2))
        (COND
           ((EQUAL TERM1 TERM2)
            NIL)
           ((for ARG1 in (FARGS TERM1) as ARG2 in (FARGS TERM2)
               thereis (OR [AND (OR (VARIABLEP ARG1)
                                    (VALUEP ARG1))
                                (NOT (OR (VARIABLEP ARG2)
                                         (VALUEP ARG2]
                           (WORSE-THAN ARG2 ARG1)))
            NIL)
           (T (for ARG1 in (FARGS TERM1) as ARG2 in (FARGS TERM2) thereis (WORSE-THAN ARG1 ARG2]
       (T NIL])
```

(R

```
  [LAMBDA (FORM)                                           (* kbr: "19-Oct-85 16:31")
    (COND
       ([NOT (ERRSET (SETQ FORM (TRANSLATE FORM]
        NIL)
       ((EQ (SETQ TEMP-TEMP (BM-REDUCE FORM R-ALIST))
            (QUOTE *1*FAILED))
        (QUOTE (NOT REDUCIBLE)))
       (T (EXPAND-PPR-MACROS TEMP-TEMP])
```

(REDO!

```
  [LAMBDA (NAME)                                           (* kbr: "19-Oct-85 16:31")
    (REDO-UNDONE-EVENTS (UNDO-NAME NAME)
           T
           (QUOTE C)
           NIL T T])
```

(REDO-UNDONE-EVENTS

```
  [LAMBDA (EVENTS ALL-FLG FAILURE-ACTION DETACH-FLG DO-NOT-PRINT-FIRST-EVENT-FLG DO-NOT-PRINT-DATE-LINE-FLG)
                                                           (* kbr: " 6-Jul-86 09:48")
    [COND
       (IN-REDO-UNDONE-EVENTS-FLG (ERROR1 (PQUOTE (PROGN IT IS ILLEGAL TO ENTER A THEOREM PROVER FUNCTION
                                                         WHILE YOU ARE RECURSIVELY UNDER ANOTHER THEOREM
                                                         PROVER FUNCTION %.))
                                          NIL
                                          (QUOTE HARD]
    (LET (ANS ANSLST FORM (IN-REDO-UNDONE-EVENTS-FLG T))
         (PROG NIL
               (OR FAILURE-ACTION (SETQ FAILURE-ACTION (QUOTE Q)))
               (COND
                  ((NOT (OPENP TTY-FILE))
                   (SETQ TTY-FILE NIL)))
               (COND
                  ((NOT (OPENP PROVE-FILE))
                   (SETQ PROVE-FILE NIL)))
               (PREPARE-FOR-THE-NIGHT)
               (COND
                  (DETACH-FLG (SETQ ALL-FLG T)
                              (DETACH)))
               (SETQ UNDONE-EVENTS EVENTS)
               (COND
```

```
            ((NOT DO-NOT-PRINT-DATE-LINE-FLG)
             (PRINT-DATE-LINE)))
        LOOP
            (COND
               ((NULL UNDONE-EVENTS)
                (GO EXIT)))
            (SETQ FORM (CAR UNDONE-EVENTS))
            [COND
               ((OR (NOT DO-NOT-PRINT-FIRST-EVENT-FLG)
                    (NEQ FORM (CAR EVENTS))
                    (NEQ PROVE-FILE NIL))
                (ITERPRIN 1 PROVE-FILE)
                (IPRINC EVENT-SEPARATOR-STRING PROVE-FILE)
                (ITERPRIN 2 PROVE-FILE)
                (COND
                   (BOOK-SYNTAX-FLG (DUMP (LIST FORM)
                                          PROVE-FILE 5 (LINEL PROVE-FILE)
                                          NIL T))
                   (T (PPRIND FORM 0 0 PPR-MACRO-LST PROVE-FILE)))
                (ITERPRI PROVE-FILE)
                (COND
                   ((AND (NEQ PROVE-FILE NIL)
                         (NOT (DETACHEDP)))
                    (IPRINC (CADR FORM)
                            T]
            [COND
               ((OR (MEMB (CAR FORM)
                          (QUOTE (DEFN REFLECT)))
                    ALL-FLG
                    (EQ FORM (CAR EVENTS))
                    (IPRINC "DO YOU WANT TO REDO THIS EVENT?" NIL))
                (START-STATS)
                [SETQ ANS (LET (UNDONE-EVENTS)
                               (APPLY (CAR FORM)
                                      (CDR FORM]
                (STOP-STATS)
                [COND
                   ((EQ ANS (QUOTE *****ERROR*****))
                    (ERROR (QUOTE REDO-UNDONE-EVENTS))
                    (GO LOOP))
                   ((OR (NOT DO-NOT-PRINT-FIRST-EVENT-FLG)
                        (NEQ FORM (CAR EVENTS))
                        (NEQ PROVE-FILE NIL))
                    (IPRINT ANS PROVE-FILE)
                    (COND
                       ((AND (NOT (DETACHEDP))
                             (NEQ PROVE-FILE NIL))
                        (COND
                           ((EQ ANS NIL)
                            (ITERPRI T)
                            (IPRINC FAILURE-MSG T)
                            (ITERPRI T))
                           (T (IPRINC "," T]
                (SETQ ANSLST (NCONC1 ANSLST ANS))
                (COND
                   ((EQ ANS NIL)
                    (COND
                       ((AND (EQ FAILURE-ACTION (QUOTE A))
                             (EQ (CAR FORM)
                                 (QUOTE PROVE-LEMMA)))
                        (ITERPRIN 2 PROVE-FILE)
                        (BM-PPR (LIST (QUOTE COMMENT)
                                      (LIST (QUOTE ADD-AXIOM)
                                            (BM-NTH 1 FORM)
                                            (BM-NTH 2 FORM)
                                            (BM-NTH 3 FORM)))
                                PROVE-FILE)
                        (ITERPRI PROVE-FILE)
                        (IPRINT (APPLY (QUOTE ADD-AXIOM)
                                       (LIST (BM-NTH 1 FORM)
                                             (BM-NTH 2 FORM)
                                             (BM-NTH 3 FORM)))
                                PROVE-FILE))
                       ([OR (EQ FAILURE-ACTION (QUOTE Q))
                            (MEMB (CAR FORM)
                                  (QUOTE (ADD-AXIOM ADD-SHELL DCL]
                        (GO EXIT]
            (SETQ UNDONE-EVENTS (CDR UNDONE-EVENTS))
            (SETQ EVENTS NIL)
            (GO LOOP)
        EXIT
            (COND
               ((NOT (EQUAL PROVE-FILE NIL))
                (ITERPRIN 1 PROVE-FILE)
                (IPRINC EVENT-SEPARATOR-STRING PROVE-FILE)
                (PRINT-SYSTEM PROVE-FILE)
                (IPRINC "REDO-UNDONE-EVENTS COMPLETED. HERE IS FAILED-THMS:" PROVE-FILE)
```

```
                (ITERPRI PROVE-FILE)
                (BM-PPR FAILED-THMS PROVE-FILE)
                (ITERPRI PROVE-FILE)
                (CLOSEF PROVE-FILE)
                (SETQ PROVE-FILE NIL)))
            (COND
               ((NOT (EQUAL TTY-FILE NIL))
                (CLOSEF TTY-FILE)
                (SETQ TTY-FILE NIL)))
            (RETURN ANSLST])
```

(BM-REDUCE

```
  [LAMBDA (TERM ALIST)                                    (* kbr: "19-Oct-85 16:31")

          (* TERM is a term. ALIST is an alist dotting variable names to EVGs.
          Reduce TERM under the assumptions that each var is equal to the corresponding constant.
          Return the resulting term or *1*FAILED if TERM is not reducible.
          BM-REDUCE is just serving as a name from which REDUCE1 sometimes RETFROMs.
          *)

    (LIST (QUOTE QUOTE)
          (REDUCE1 TERM ALIST])
```

(REDUCE1

```
  [LAMBDA (TERM ALIST)                                    (* kbr: "19-Oct-85 16:31")
    (COND
       [(VARIABLEP TERM)
        (COND
           ((SETQ TEMP-TEMP (ASSOC TERM ALIST))
            (CDR TEMP-TEMP))
           (T (RETFROM (QUOTE BM-REDUCE)
                       (QUOTE *1*FAILED]
       ((FQUOTEP TERM)
        (CADR TERM))
       [(EQ (FFN-SYMB TERM)
            (QUOTE IF))
        (COND
           ((EQ (REDUCE1 (FARGN TERM 1)
                         ALIST)
                *1*F)
            (REDUCE1 (FARGN TERM 3)
                     ALIST))
           (T (REDUCE1 (FARGN TERM 2)
                       ALIST]
       [(SETQ TEMP-TEMP (GETPROP (FFN-SYMB TERM)
                                 (QUOTE LISP-CODE)))

          (* We special case the fns of arity 0, 1, 2, and 3 to avoid consing up the arg list.
          *)

        (SELECTQ (LENGTH TERM)
            (1 (APPLY* TEMP-TEMP))
            (2 (APPLY* TEMP-TEMP (REDUCE1 (FARGN TERM 1)
                                          ALIST)))
            (3 (APPLY* TEMP-TEMP (REDUCE1 (FARGN TERM 1)
                                          ALIST)
                       (REDUCE1 (FARGN TERM 2)
                                ALIST)))
            (4 (APPLY* TEMP-TEMP (REDUCE1 (FARGN TERM 1)
                                          ALIST)
                       (REDUCE1 (FARGN TERM 2)
                                ALIST)
                       (REDUCE1 (FARGN TERM 3)
                                ALIST)))
            (APPLY TEMP-TEMP (for ARG in (FARGS TERM) collect (REDUCE1 ARG ALIST]
       (T (RETFROM (QUOTE BM-REDUCE)
                   (QUOTE *1*FAILED])
```

(REFLECT0

```
  [LAMBDA (NAME SATISFACTION-LEMMA-NAME RELATION-MEASURE-LST FLG)
                                                          (* kbr: "20-Oct-85 19:37")
    (LET (TRANSLATED-BODY CONTROL-VARS FN ARGS BODY (META-NAMES (CONS NAME META-NAMES)))
         (BM-MATCH (FORMULA-OF SATISFACTION-LEMMA-NAME)
                   (EQUAL (CONS FN ARGS)
                          BODY))
         (SETQ TRANSLATED-BODY (TRANSLATE BODY))
         [SETQ RELATION-MEASURE-LST (for TEMP in RELATION-MEASURE-LST collect (LIST (CAR TEMP)
                                                                                   (TRANSLATE (CADR TEMP]
         (PUT-INDUCTION-INFO NAME ARGS TRANSLATED-BODY RELATION-MEASURE-LST FN)
         (ADD-FACT NAME (QUOTE SDEFN)
                   (LIST (QUOTE LAMBDA)
                         ARGS
                         (SUBST-FN NAME FN TRANSLATED-BODY)))
         [ADD-FACT NAME (QUOTE TYPE-PRESCRIPTION-LST)
```

```
          (CAR (GETPROP FN (QUOTE TYPE-PRESCRIPTION-LST]
      (PUT-LEVEL-NO NAME)
      [AND (GETPROP NAME (QUOTE JUSTIFICATIONS))
           (ADD-FACT NAME (QUOTE CONTROLLER-POCKETS)
                  (SCRUNCH (for TEMP in (GETPROP NAME (QUOTE JUSTIFICATIONS))
                               collect (PROGN (SETQ CONTROL-VARS (fetch (JUSTIFICATION SUBSET) of TEMP))
                                              (for FORMAL in ARGS as I from 0 bind (LOOP-ANS _ 0)
                                                 when (MEMB FORMAL CONTROL-VARS)
                                                 do (SETQ LOOP-ANS (LOGOR LOOP-ANS (LSH 1 I)))
                                                 finally (RETURN LOOP-ANS]
      [COND
         [FLG (ADD-FACT NAME (QUOTE LISP-CODE)
                        (PACK (LIST STRING-WEIRD NAME]
         ([for FN in (ALL-FNNAMES TRANSLATED-BODY) always (OR (EQ FN NAME)
                                                              (GETPROP FN (QUOTE LISP-CODE]
          (ADD-DCELL NAME (PACK (LIST STRING-WEIRD NAME))
                 (LIST (QUOTE LAMBDA)
                       [SETQ TEMP-TEMP (for ARG in ARGS collect (PACK (LIST STRING-WEIRD3 ARG]
                       (TRANSLATE-TO-LISP (SUB-PAIR-VAR ARGS TEMP-TEMP (SUBST-FN NAME FN TRANSLATED-BODY]
      [COND
         ((NOT (TOTAL-FUNCTIONP NAME))
          (ERROR1 (PQUOTE (PROGN THE RECURSION IN (!PPR NAME NIL)
                                 IS UNJUSTIFIED %.))
                 (BINDINGS (QUOTE NAME)
                           NAME)
                 (QUOTE WARNING]
      NIL])
```

(RELIEVE-HYPS

```
[LAMBDA (HYPS LEMMA-NAME)                                   (* kbr: "19-Oct-85 16:31")
  (PUSH-LEMMA-FRAME)
  (PUSH-LINEARIZE-ASSUMPTIONS-FRAME)
  (COND
     ((RELIEVE-HYPS1 HYPS LEMMA-NAME)
      (for X in (POP-LEMMA-FRAME) do (PUSH-LEMMA X))
      (for X in (POP-LINEARIZE-ASSUMPTIONS-FRAME) do (PUSH-LINEARIZE-ASSUMPTION X))
      T)
     (T (POP-LEMMA-FRAME)
        (POP-LINEARIZE-ASSUMPTIONS-FRAME)
        NIL])
```

(RELIEVE-HYPS-NOT-OK

```
[LAMBDA (LIT)                                               (* kbr: "20-Oct-85 19:32")
  (LET (LIT-ATOM ANS-ATOM)
       (SETQ LIT-ATOM LIT)
       (BM-MATCH LIT (NOT LIT-ATOM))
       (for ANS in ANCESTORS thereis (PROGN (SETQ ANS-ATOM ANS)
                                            (BM-MATCH ANS (NOT ANS-ATOM))
                                            (COND
                                               ((EQUAL LIT ANS)
                                                (SETQ RELIEVE-HYPS-NOT-OK-ANS T)
                                                T)
                                               ((AND (GREATEREQP (FORM-COUNT LIT-ATOM)
                                                                 (FORM-COUNT ANS-ATOM))
                                                     (WORSE-THAN-OR-EQUAL LIT-ATOM ANS-ATOM))
                                                (SETQ RELIEVE-HYPS-NOT-OK-ANS NIL)
                                                T)
                                               (T NIL])
```

(RELIEVE-HYPS1

```
[LAMBDA (HYPS LEMMA-NAME)                                   (* kbr: "20-Oct-85 19:12")
  (COND
     ((for HYP in HYPS as I from 1 bind (SPLIT-FLG CHECK-FLG LHS RHS)
         always [PROGN (PRINT-TO-DISPLAY LEMMA-NAME I (QUOTE ?))
                       [OR (SETQ SPLIT-FLG (BM-MATCH HYP (SPLIT HYP)))
                           (SETQ CHECK-FLG (BM-MATCH HYP (CHECK HYP]
                       (COND
                          ((LOOKUP-HYP HYP)
                           T)
                          ((FREE-VARSP HYP UNIFY-SUBST)
                           (COND
                              ((AND (BM-MATCH HYP (EQUAL LHS RHS))
                                    (VARIABLEP LHS)
                                    (NOT (ASSOC LHS UNIFY-SUBST))
                                    (NOT (FREE-VARSP RHS UNIFY-SUBST)))
                               (SETQ UNIFY-SUBST (CONS (CONS LHS (REWRITE RHS UNIFY-SUBST TYPE-ALIST
                                                                          (QUOTE ?)
                                                                          (QUOTE ID)
                                                                          NIL))
                                                       UNIFY-SUBST)))
                              ((SEARCH-GROUND-UNITS HYP)
                               T)
                              (T (SETQ LAST-EXIT (QUOTE FREE-VARSP))
                                 NIL)))
```

```
                              ((RELIEVE-HYPS-NOT-OK (SETQ INST-HYP (SUBLIS-VAR UNIFY-SUBST HYP)))
                               (SETQ LAST-EXIT (QUOTE RELIEVE-HYPS-NOT-OK))
                               RELIEVE-HYPS-NOT-OK-ANS)
                              ((FALSE-NONFALSEP INST-HYP)
                               (SETQ LAST-EXIT (QUOTE FALSE-NONFALSEP))
                               (NOT DEFINITELY-FALSE))
                              ((MEMBER INST-HYP LITS-THAT-MAY-BE-ASSUMED-FALSE)
                               (SETQ LAST-EXIT (QUOTE LITS-THAT-MAY-BE-ASSUMED-FALSE))
                               NIL)
                              (SPLIT-FLG (PUSH-LINEARIZE-ASSUMPTION INST-HYP)
                                     T)
                              (CHECK-FLG (SETQ LAST-EXIT (QUOTE CHECK-FLG))
                                     NIL)
                              ((BM-MATCH HYP (NOT HYP))
                               (LET ((ANCESTORS (CONS (DUMB-NEGATE-LIT INST-HYP)
                                                      ANCESTORS)))
                                    (SETQ LAST-EXIT (REWRITE HYP UNIFY-SUBST TYPE-ALIST (QUOTE FALSE)
                                                             (QUOTE IFF)
                                                             NIL))
                                    (EQUAL LAST-EXIT FALSE)))
                              (T (LET ((ANCESTORS (CONS (DUMB-NEGATE-LIT INST-HYP)
                                                        ANCESTORS)))
                                      (SETQ LAST-EXIT (REWRITE HYP UNIFY-SUBST TYPE-ALIST (QUOTE TRUE)
                                                               (QUOTE IFF)
                                                               NIL)) (* Could be NOT-IDENT FALSE but LAST-EXIT was just
                                                                        rewritten with IFF. *)
                                      (EQUAL LAST-EXIT TRUE]
          finally (SETQ LAST-HYP HYP))
        (PRINT-TO-DISPLAY LEMMA-NAME NIL (QUOTE !))
        T)
       (T NIL])
```

(REMOVE-*2*IFS

```
[LAMBDA (X)                                                    (* kbr: "19-Oct-85 16:31")
  (LET (REST)
       (COND
          ((NLISTP X)
           X)
          ((EQ (CAR X)
               (QUOTE QUOTE))
           X)
          [(EQ (CAR X)
               (QUOTE *2*IF))
           (SETQ REST (REMOVE-*2*IFS (CADDDR X)))
           (CONS (QUOTE COND)
                 (CONS (LIST (REMOVE-*2*IFS (CADR X))
                             (REMOVE-*2*IFS (CADDR X)))
                       (COND
                          ((AND (LISTP REST)
                                (EQ (CAR REST)
                                    (QUOTE COND)))
                           (CDR REST))
                          (T (LIST (LIST T REST]
          (T (CONS (CAR X)
                   (for ARG in (CDR X) collect (REMOVE-*2*IFS ARG])
```

(REMOVE-NEGATIVE

```
[LAMBDA (LIT CL)                                               (* kbr: "19-Oct-85 16:31")
  (COND
     ((NLISTP CL)
      NIL)
     ((COMPLEMENTARYP LIT (CAR CL))
      (CDR CL))
     (T (CONS (CAR CL)
              (REMOVE-NEGATIVE LIT (CDR CL])
```

(REMOVE-REDUNDANT-TESTS

```
[LAMBDA (TO-DO DONE)                                           (* kbr: "19-Oct-85 16:31")
```

(* When this function was conceived, we used to run the following code.
However, we have trivialized the effect because we found that it sometimes hurt.
In particular, if the tests were (LISTP X) and (EQUAL (CAAR X)
(QUOTE FOO)), the LISTP could get removed. But then the LISTP has to be rederived when it comes up during a proof.
It is speculated that the original motivation for this function was messy base cases, which was altered if not fixed by carrying around the base cases in the INDUCTION-MACHINE. The following code is left in case a real removal of tests is deemed necessary. (COND ((NULL TO-DO) DONE) ((AND (SIMPLIFY-CLAUSE-MAXIMALLY
(CONS (CAR TO-DO) (APPEND (FOR X IN (CDR TO-DO) COLLECT
(NEGATE-LIT X)) (FOR X IN DONE COLLECT (NEGATE-LIT X)))))
(NULL PROCESS-CLAUSES)) The lemmas on PROCESS-HIST will have been added to ALL-LEMMAS-USED by SIMPLIFY-CLAUSE under SIMPLIFY-CLAUSE-MAXIMALLY and ALL-LEMMAS-USED is correctly initialized and processed by DEFN-SETUP and the post processing in DEFN. (REMOVE-REDUNDANT-TESTS
(CDR TO-DO) DONE)) (T (REMOVE-REDUNDANT-TESTS (CDR TO-DO)
(CONS (CAR TO-DO) DONE)))) . *)

```
    (APPEND TO-DO DONE])
```

(REMOVE1

```
  [LAMBDA (X LST)                                          (* kbr: "19-Oct-85 16:31")
    (COND
       ((NLISTP LST)
        NIL)
       ((EQ X (CAR LST))
        (CDR LST))
       (T (CONS (CAR LST)
                (REMOVE1 X (CDR LST])
```

(REMOVE-TRIVIAL-EQUATIONS

```
  [LAMBDA (CL)                                             (* kbr: "20-Oct-85 13:36")

          (* First cut down on variables by eliminating any inequality involving a variable LHS with a RHS that doesn't contain LHS.
          *)

    [bind (LHS RHS) while (for LIT in CL thereis (AND (OR [AND (BM-MATCH LIT (NOT (EQUAL LHS RHS)))
                                                               (OR (AND (VARIABLEP LHS)
                                                                        (NOT (OCCUR LHS RHS)))
                                                                   (AND (PROG2 (swap LHS RHS)
                                                                               T)
                                                                        (VARIABLEP LHS)
                                                                        (NOT (OCCUR LHS RHS]
                                                          (AND (VARIABLEP LIT)
                                                               (PROGN (SETQ LHS LIT)
                                                                      (SETQ RHS FALSE)
                                                                      T)))
                                                     (PROGN (SETQ CL (for LIT2 in CL unless (EQ LIT LIT2)
                                                                        collect (SUBST-VAR RHS LHS LIT2)))
                                                            T]

          (* Next any inequality between a LHS and a constant RHS is used to replace occurences of LHS.
          (But we cannot get rid of the original inequality.) *)

    [bind (LHS RHS) while (for LIT in CL thereis (AND (BM-MATCH LIT (NOT (EQUAL LHS RHS)))
                                                     (OR (AND (NOT (QUOTEP LHS))
                                                              (QUOTEP RHS))
                                                         (AND (PROG2 (swap LHS RHS)
                                                                     T)
                                                              (NOT (QUOTEP LHS))
                                                              (QUOTEP RHS)))
                                                     (for LIT2 in CL when (NEQ LIT LIT2)
                                                        thereis (OCCUR LHS LIT2))
                                                     (SETQ CL (for LIT2 in CL
                                                                 collect (COND
                                                                            ((OR (EQ LIT LIT2)
                                                                                 (NOT (OCCUR LHS LIT2)))
                                                                             LIT2)
                                                                            (T (SUBST-EXPR RHS LHS LIT2]
    CL])
```

(REMOVE-UNCHANGING-VARS

```
  [LAMBDA (CAND-LST CL-SET)                                (* kbr: "19-Oct-85 16:31")
    (LET (NOT-CHANGING-VARS)
         (SETQ NOT-CHANGING-VARS (for CL in CL-SET bind LOOP-ANS
                                    do (SETQ LOOP-ANS (UNIONQ (for LIT in CL bind LOOP-ANS
                                                                 do (SETQ LOOP-ANS (UNIONQ (UNCHANGING-VARS
                                                                                             LIT)
                                                                                           LOOP-ANS))
                                                                 finally (RETURN LOOP-ANS))
                                                              LOOP-ANS))
                                    finally (RETURN LOOP-ANS)))
         (OR (for CAND in CAND-LST unless (INTERSECTP (fetch (CANDIDATE CHANGED-VARS) of CAND)
                                                      NOT-CHANGING-VARS)
                collect CAND)
             CAND-LST])
```

(REMPROP1

```
  [LAMBDA (AT PROP)                                        (* kbr: "19-Oct-85 16:31")
    AT PROP (ERROR1 (PQUOTE (PROGN IT IS NOT PERMITTED TO USE REMPROP1 ON PROPERTIES MAINTAINED BY PUT1
                                   AND GETPROP !))
                    (BINDINGS)
                    (QUOTE HARD])
```

(RESTART

```
  [LAMBDA (X)                                              (* kbr: "19-Oct-85 16:31")
    (REDO-UNDONE-EVENTS (OR X UNDONE-EVENTS)
           T
           (QUOTE Q)
           NIL NIL NIL])
```

(RESTART-BATCH

```
[LAMBDA (LST)                                                    (* kbr: "19-Oct-85 16:31")
  (PROG NIL
        (SETQ UNDONE-BATCH-COMMANDS LST)
    TOP (COND
           ((NULL UNDONE-BATCH-COMMANDS)
            (RETURN NIL)))
        (EVAL (CAR UNDONE-BATCH-COMMANDS))
        (SETQ UNDONE-BATCH-COMMANDS (CDR UNDONE-BATCH-COMMANDS))
        (GO TOP])
```

(REWRITE

```
[LAMBDA (TERM ALIST TYPE-ALIST OBJECTIVE ID-IFF DEFN-FLG)        (* kbr: " 4-Jul-86 16:41")

        (* Returns a term that is equal (modulo ID-IFF) to the result of substituting ALIST into TERM under the hypotheses of
        (a) TYPE-ALIST, (b) the conjunction of the top frame of LINEARIZE-ASSUMPTIONS-STACK,
        (c) and (d) some subset S of SIMPLIFY-CLAUSE-POT-LST such that if ITIMES IEQP
        (LIST (QUOTE MARK)) is MEMB the LEMMAS field of some poly in S, then ITIMES is a member of the top frame of the
        LEMMA-STACK. *)

        (* DEFN-FLG = T if TERM is the body of a definition that is being opened.
        *)

  (COND
     [(VARIABLEP TERM)
      (REWRITE-SOLIDIFY (COND
                           ((SETQ TEMP-TEMP (ASSOC TERM ALIST))
                            (CDR TEMP-TEMP))
                           (T TERM]
     ((FQUOTEP TERM)
      TERM)
     ((EQ (FFN-SYMB TERM)
          (QUOTE IF))
      (REWRITE-IF (REWRITE (FARGN TERM 1)
                           ALIST TYPE-ALIST (QUOTE ?)
                           (QUOTE IFF)
                           NIL)
                  (FARGN TERM 2)
                  (FARGN TERM 3)
                  TYPE-ALIST))
     ((SETQ TEMP-TEMP (NOT-TO-BE-REWRITTENP TERM ALIST))
      (REWRITE-SOLIDIFY TEMP-TEMP))
     (T (LET (ARGS FN TEMP)

        (* If we are inside of a defn, rewrite the args and then simplify the resulting term with lemmas etc.
        If we are not in a definition, we wish to avoid introducing too many IFs all at once and swamping CLAUSIFY.
        So rewrite the args until one of them gets an IF in it. After the first such IF, rewrite the args but if an IF shows up do not use
        the expansion -- use the result of just substituting alist into the arg.
        *)

              (SETQ ARGS (for ARG in (FARGS TERM) collect (REWRITE ARG ALIST TYPE-ALIST (QUOTE ?)
                                                                   (QUOTE ID)
                                                                   NIL)))
              (COND
                 ([AND (for ARG in ARGS always (QUOTEP ARG))
                       (SETQ FN (GETPROP (FFN-SYMB TERM)
                                         (QUOTE LISP-CODE)))
                       (NEQ (QUOTE *1*FAILED)
                            (SETQ TEMP (APPLY FN (for ARG in ARGS collect (CADR ARG]
                  (PUSH-LEMMA (FFN-SYMB TERM))
                  (LIST (QUOTE QUOTE)
                        TEMP))
                 (T                                              (* The use of FCONS-TERM below is justified by the
                                                                    immediately preceding computation.
                                                                    *)
                    (SETQ TEMP (REWRITE-TYPE-PRED (FCONS-TERM (FFN-SYMB TERM)
                                                              ARGS)))
                    (REWRITE-WITH-LEMMAS TEMP])
```

(REWRITE-FNCALL

```
[LAMBDA (*FNNAME* *ARGLIST*)                                     (* kbr: " 4-Jul-86 16:41")
  (LET (VALUE SDEFN (FNSTACK FNSTACK)
              *CONTROLLER-COMPLEXITIES*
              (LEMMA-STACK LEMMA-STACK)
              (LINEARIZE-ASSUMPTIONS-STACK LINEARIZE-ASSUMPTIONS-STACK)
              (*TYPE-ALIST* TYPE-ALIST)
              ANSWER)
       (SETQ SDEFN (GETPROP *FNNAME* (QUOTE SDEFN)))
       [SETQ ANSWER (COND
                       ((NULL SDEFN)
                        (REWRITE-SOLIDIFY (CONS-TERM *FNNAME* *ARGLIST*)))
                       ((OR (MEMB *FNNAME* FNSTACK)
                            (DISABLEDP *FNNAME*))
```

```
                        (REWRITE-SOLIDIFY (CONS-TERM *FNNAME* *ARGLIST*)))
                    (T [SETQ *CONTROLLER-COMPLEXITIES*
                        (for MASK in (GETPROP *FNNAME* (QUOTE CONTROLLER-POCKETS))
                           collect (for ARG in *ARGLIST* when (PROG1 (NOT (IEQP (LOGAND MASK 1)
                                                                                0))
                                                                     (SETQ MASK (LSH MASK -1)))
                                    sum (OR (QUOTEP ARG)
                                            (SETQ VALUE NIL))
                                        (MAX-FORM-COUNT ARG]
                       (SETQ FNSTACK (CONS *FNNAME* FNSTACK))
```

(* Add the name of the current fn to the FNSTACK so that when we see recursive calls in the body we won't be tempted to go into them. There is an odd aspect to the use of FNSTACK by this function.
Suppose that in the rewriting of the body of fn we apply a lemma and backwards chain to some hyp.
Suppose the hyp contains a call of fn. Then when we try to rewrite fn in the hyp we will think it is a recursive call and quit due to the (MEMB *FNNAME* FNSTACK) above. Once upon a time, when we did not preprocess the hyps of lemmas at all and did not EXPAND-BOOT-STRAP-NON-REC-FNS in defns this problem burned us on
(ZEROP expr) because inside the defn of ZEROP we saw (EQUAL expr 0) and we backward chained to something with a ZEROP hyp and shied away from it. This occurred while trying to use LITTLE-STEP under PRIME-KEY under QUOTIENT-DIVIDES in the proof of PRIME-LIST-TIMES-LIST --
the ZEROP we were expanding was that in the DIVIDES hyp of PRIME-KEY and the ZEROP we shied away from was that in PRIME in LITTLE-STEP. We implemented makeshift fix to that by not putting nonrec fns onto FNSTACK here.
But that does not prevent us from shying away from calls to recursive fns encountered in lemmas while somehow under the body of the fn. Worse, it turns out to be very expensive. Suppose we eliminate ZEROP by expanding it in preprocessing. Then PRIME-LIST-TIMES-LIST is proved whether we put nonrec fns onto the stack or not.
But if we do not, it takes 248K conses while if we do it takes 140K.
So we have gone back to putting everything on the stack and await the day that shying away from a spurious gets us.
*)

```
                    (PUSH-LEMMA-FRAME)
                    (PRINT-TO-DISPLAY *FNNAME* (QUOTE ?)
                           NIL)
                    (PUSH-LINEARIZE-ASSUMPTIONS-FRAME)  (* Rewrite the body of the definition *)
                    (SETQ VALUE (REWRITE (CADDR SDEFN)
                                         (for VAR in (CADR SDEFN) as VAL in *ARGLIST*
                                            collect (CONS VAR VAL))
                                         TYPE-ALIST OBJECTIVE ID-IFF T))
                    (COND
                       ((NULL (GETPROP *FNNAME* (QUOTE INDUCTION-MACHINE)))
```

(* We are dealing with a nonrec fn. If we are at the top level of the clause but the expanded body has too many IFs in it compared to the number of IFs in the args, we do not use the expanded body.
Because we know the IFs in the args will be classified out soon and we do not want to swamp CLAUSIFY by giving it too many at once. Otherwise we use the expanded body. *)

```
                        (COND
                           ((AND (for X in (CDR FNSTACK) never (GETPROP X (QUOTE INDUCTION-MACHINE)))
                                 (TOO-MANY-IFS *ARGLIST* VALUE))
                            (POP-LEMMA-FRAME)
                            (POP-LINEARIZE-ASSUMPTIONS-FRAME)
                            (REWRITE-SOLIDIFY (FCONS-TERM *FNNAME* *ARGLIST*)))
                           (T (for X in (POP-LINEARIZE-ASSUMPTIONS-FRAME) do (PUSH-LINEARIZE-ASSUMPTION
                                                                                X))
                              (PRINT-TO-DISPLAY *FNNAME* (QUOTE !)
                                     NIL)
                              (for X in (POP-LEMMA-FRAME) do (PUSH-LEMMA X))
                              (PUSH-LEMMA *FNNAME*)
                              VALUE)))
                       ((REWRITE-FNCALLP *FNNAME* VALUE)
                        (for X in (POP-LINEARIZE-ASSUMPTIONS-FRAME) do (PUSH-LINEARIZE-ASSUMPTION X))
                        (PRINT-TO-DISPLAY *FNNAME* (QUOTE !)
                               NIL)
                        (for X in (POP-LEMMA-FRAME) do (PUSH-LEMMA X))
                        (PUSH-LEMMA *FNNAME*)
                        VALUE)
                       (T (POP-LEMMA-FRAME)
                          (POP-LINEARIZE-ASSUMPTIONS-FRAME)
                          (REWRITE-SOLIDIFY (CONS-TERM *FNNAME* *ARGLIST*]
    (COND
       (NIL (NOT (EQUAL ANSWER (CONS *FNNAME* *ARGLIST*)))
            (SHOWPRINT (CONS *FNNAME* *ARGLIST*))
            (SHOWPRINT ANSWER)
            (\GETKEY)))
    ANSWER])
```

(REWRITE-FNCALLP

```
  [LAMBDA (FNNAME VALUE)                                    (* kbr: " 4-Jul-86 18:38")
```

(* A FNNAME call can be opened to give VALUE where FNNAME may be recursively defined.
Are all the FNNAME calls in VALUE better than the original FNNAME call? *)

```
    (COND
       ((VARIABLEP VALUE)
        T)
       ((FQUOTEP VALUE)
        T)
```

```
       [(EQ (FFN-SYMB VALUE)
            FNNAME)

          (* The recursive call is OK if (1) each arg of the call already occurs in some literal of CURRENT-CL or
          (2) the call itself occurs in CURRENT-SIMPLIFY-CL or (3) the actuals of the recursive call corresponding to the SUBSET of
          some JUSTIFICATION for the termination of FNNAME are overall less complex than those of the original call or
          (4) the actuals of the recursive call corresponding to the SUBSET of some JUSTIFICATION for the termination of FNNAME
          are constant and some actual not corresponding to a formal in the SUBSET is symbolically simpler now than before *)

          (AND [OR (for ARG in (FARGS VALUE) always (for LIT in CURRENT-CL thereis (DUMB-OCCUR ARG LIT)))
                   (for LIT in CURRENT-SIMPLIFY-CL thereis (DUMB-OCCUR VALUE LIT))
                   (for N in *CONTROLLER-COMPLEXITIES* as MASK in (GETPROP FNNAME (QUOTE CONTROLLER-POCKETS))
                      thereis (LESSP (for ARG in (FARGS VALUE) when (PROG1 (NOT (IEQP (LOGAND MASK 1)
                                                                                       0))
                                                                           (SETQ MASK (LSH MASK -1)))
                                     sum (MAX-FORM-COUNT ARG))
                                  N))
                   (for MASK in (GETPROP FNNAME (QUOTE CONTROLLER-POCKETS)) bind TEMP
                      thereis (PROGN (SETQ TEMP MASK)

          (* Is there a controller pocket such that all the controllers are constant and some non controller is symbolically simpler now
          than before? *)

                                (AND (for ARG in (FARGS VALUE)
                                        when (PROG1 (NOT (IEQP (LOGAND TEMP 1)
                                                               0))
                                                    (SETQ TEMP (LSH TEMP -1)))
                                        always (QUOTEP ARG))
                                     (for ARG1 in *ARGLIST* as ARG2 in (FARGS VALUE)
                                        thereis (AND (PROG1 (IEQP (LOGAND MASK 1)
                                                                  0)
                                                            (SETQ MASK (LSH MASK -1)))
                                                     (LESSP (MAX-FORM-COUNT ARG2)
                                                            (MAX-FORM-COUNT ARG1]
            (for ARG in (FARGS VALUE) always (REWRITE-FNCALLP FNNAME ARG]
       (T (for ARG in (FARGS VALUE) always (REWRITE-FNCALLP FNNAME ARG])
```

(REWRITE-IF

```
  [LAMBDA (TEST LEFT RIGHT TYPE-ALIST)                              (* kbr: "29-Jun-86 18:02")
                                                                    (* Rewrites the term (IF TEST LEFT RIGHT) *)

    [COND
       ((AND (NVARIABLEP TEST)
             (NOT (FQUOTEP TEST))
             (EQ (FFN-SYMB TEST)
                 (QUOTE IF))
             (EQUAL (FARGN TEST 2)
                    FALSE)
             (FALSE-NONFALSEP (FARGN TEST 3))
             (NOT DEFINITELY-FALSE))                                (* Note: FALSE-NONFALSEP sets DEFINITELY-FALSE *)
                                                                    (* Change (IF (IF P FALSE TRUE) LEFT RIGHT) to
                                                                    (IF P RIGHT LEFT) *)
        (swap LEFT RIGHT)
        (SETQ TEST (FARGN TEST 1]
    (ASSUME-TRUE-FALSE TEST)
    (COND
       (MUST-BE-TRUE (JUMPOUTP LEFT (REWRITE LEFT ALIST TYPE-ALIST OBJECTIVE ID-IFF DEFN-FLG)))
       (MUST-BE-FALSE (JUMPOUTP RIGHT (REWRITE RIGHT ALIST TYPE-ALIST OBJECTIVE ID-IFF DEFN-FLG)))
       (T (REWRITE-IF1 TEST (JUMPOUTP LEFT (LET (FALSE-TYPE-ALIST)
                                                (REWRITE LEFT ALIST TRUE-TYPE-ALIST OBJECTIVE ID-IFF DEFN-FLG)
                                                ))
                   (JUMPOUTP RIGHT (REWRITE RIGHT ALIST FALSE-TYPE-ALIST OBJECTIVE ID-IFF DEFN-FLG])
```

(REWRITE-IF1

```
  [LAMBDA (TEST LEFT RIGHT)                                         (* kbr: "29-Jun-86 18:02")
                                                                    (* Called by REWRITE-IF to rewrite
                                                                    (IF TEST LEFT RIGHT) *)

    (COND
       ((EQUAL LEFT RIGHT)                                          (* Change (IF TEST P P) to P *)
        LEFT)
       ((AND (EQUAL TEST LEFT)
             (FALSE-NONFALSEP RIGHT)
             DEFINITELY-FALSE)                                      (* Change (IF P P FALSE) to P *)
        TEST)
       ((AND (EQUAL TRUE LEFT)
             (FALSE-NONFALSEP RIGHT)
             DEFINITELY-FALSE
             (BOOLEAN TEST))                                        (* Change (IF TEST TRUE FALSE) to TEST if TEST is a
                                                                    boolean *)
        TEST)
       (T (FCONS-TERM* (QUOTE IF)
                  TEST LEFT RIGHT])
```

(REWRITE-LINEAR-CONCL

```
  [LAMBDA (CONCL)                                                   (* kbr: "19-Oct-85 16:31")
```

```
          (* We desire to rewrite the instantiated conclusion of linear lemmas before adding them to the linear pot.
          However, because all of the literals of the clause being proved are on the TYPE-ALIST as false, it is possible --
          say when proving an instance of an already proved linear lemma --
          to rewrite the conclusion to F! We could avoid this by either not putting the linear-like literals on the type alist in the first
          place, or by not rewriting the entire conclusion, just the args. We took the latter approach because it was simplest.
          It does suffer from the possibility that the whole (LESSP lhs rhs) of the conclusion might rewrite to something else, possibly
          a better LESSP. *)

    (LET (LHS RHS)
         (COND
            ((BM-MATCH CONCL (LESSP LHS RHS))
             (FCONS-TERM* (QUOTE LESSP)
                    (REWRITE LHS UNIFY-SUBST TYPE-ALIST (QUOTE ?)
                           (QUOTE ID)
                           NIL)
                    (REWRITE RHS UNIFY-SUBST TYPE-ALIST (QUOTE ?)
                           (QUOTE ID)
                           NIL)))
            [(BM-MATCH CONCL (NOT (LESSP LHS RHS)))
             (FCONS-TERM* (QUOTE NOT)
                    (FCONS-TERM* (QUOTE LESSP)
                           (REWRITE LHS UNIFY-SUBST TYPE-ALIST (QUOTE ?)
                                  (QUOTE ID)
                                  NIL)
                           (REWRITE RHS UNIFY-SUBST TYPE-ALIST (QUOTE ?)
                                  (QUOTE ID)
                                  NIL]
            (T (ERROR1 (PQUOTE (PROGN REWRITE-LINEAR-CONCL THOUGHT THAT ALL LINEAR LEMMAS HAD CONCLUSIONS WITH
                                      NLISTP LESSP !)))
                     NIL
                     (QUOTE HARD])
```

(REWRITE-SOLIDIFY

```
  [LAMBDA (TERM)                                                    (* kbr: "29-Jun-86 17:43")
                                                                    (* Rewrites TERM with the context supplied by TYPE-ALIST *)
    (LET (LIT TEMP LHS RHS)
         (COND
            ((QUOTEP TERM)
             TERM)
            ((AND (NVARIABLEP TERM)
                  (EQ (FFN-SYMB TERM)
                      (QUOTE IF)))                                  (* See the proof in JUMPOUTP. *)
             TERM)
            ((for PAIR in TYPE-ALIST thereis (AND (IEQP (CDR PAIR)
                                                        TYPE-SET-TRUE)
                                                  (BM-MATCH (CAR PAIR)
                                                         (EQUAL LHS RHS))
                                                  (EQUAL LHS TERM)))

             (* If TERM is equal to the LHS of a true equality then TERM rewrites to the RHS of the equality.
             *)

             RHS)
            ((AND (SETQ TEMP-TEMP (SASSOC TERM TYPE-ALIST))
                  (SETQ TEMP (OBJ-TABLE (CDR TEMP-TEMP)
                                        OBJECTIVE ID-IFF)))

             (* If the TERM is in the TYPE-ALIST as true or false then return true or false.
             *)

             TEMP)
            ((SETQ LIT (for LIT in LITS-THAT-MAY-BE-ASSUMED-FALSE when (COND
                                                                          ((EQUAL LIT TERM)
                                                                           (SETQ TEMP FALSE))
                                                                          ((COMPLEMENTARYP LIT TERM)
                                                                           (SETQ TEMP TRUE))
                                                                          (T NIL))
                                                                          do (RETURN LIT)))
             (COND
                ((OR (EQ ID-IFF (QUOTE IFF))
                     (EQ TEMP FALSE)
                     (BOOLEAN TERM))
                 (PUSH-LEMMA LIT)
                 TEMP)
                (T TERM)))
            (T TERM])
```

(REWRITE-TYPE-PRED

```
  [LAMBDA (TERM)                                                    (* kbr: "19-Oct-85 16:31")
    (LET (LHS RHS PAIR TYPE-SET)
         (COND
            ((OR (VARIABLEP TERM)
                 (FQUOTEP TERM))
             TERM)
```

```
            ((BM-MATCH TERM (EQUAL LHS RHS))
             (COND
                ((EQUAL LHS RHS)
                 TRUE)
                ((NOT-IDENT LHS RHS)
                 FALSE)
                ((AND (BOOLEAN LHS)
                      (EQUAL TRUE RHS))
                 LHS)
                ((AND (BOOLEAN RHS)
                      (EQUAL TRUE LHS))
                 RHS)
                ((BM-MATCH RHS (EQUAL & &))
                 (FCONS-TERM* (QUOTE IF)
                        RHS
                        (FCONS-TERM* (QUOTE EQUAL)
                               LHS TRUE)
                        (FCONS-TERM* (QUOTE IF)
                               LHS FALSE TRUE)))
                ((EQUAL LHS FALSE)
                 (FCONS-TERM* (QUOTE IF)
                        RHS FALSE TRUE))
                ((EQUAL RHS FALSE)
                 (FCONS-TERM* (QUOTE IF)
                        LHS FALSE TRUE))
                ((BM-MATCH LHS (EQUAL & &))
                 (FCONS-TERM* (QUOTE IF)
                        LHS
                        (FCONS-TERM* (QUOTE EQUAL)
                               RHS TRUE)
                        (FCONS-TERM* (QUOTE IF)
                               RHS FALSE TRUE)))
                ((AND (SETQ TYPE-SET (TYPE-SET LHS))
                      (for X in RECOGNIZER-ALIST thereis (IEQP TYPE-SET (CDR X)))
                      (IEQP TYPE-SET (TYPE-SET RHS))
                      (NOT (BTM-OBJECT-OF-TYPE-SET TYPE-SET)))
```

(* This piece of code was hacked together to test the idea that if you have an
(EQUAL lhs rhs) in which lhs and rhs have the same type -- and that type does not contain a btm object --
that you should rewrite it to T or F provided you can appropriately decide the equalities of the components.
Before attempting to add complete equality we did not do anything like this and relied solely on elim to do it for us.
In the first attempt to add it to rewrite we just rewrote all such (EQUAL lhs rhs) to the conjunction of the equalities of the components. That was unsatisfactory because it caused such equalities as
(EQUAL (ADDTOLIST X L) B) to be torn up all the time. That caused us to fail to prove thms like
SORT-OF-ORDERED-NUMBER-LIST because weak subgoals are pushed --
subgoals about (CAR (ADDTOLIST X L)) and (CDR (ADDTOLIST X L)) instead about
(ADDTOLIST X L) itself. If this piece of code survives it should be cleaned up.
Two problems. We repeatedly cons up the constant (EQUAL (CAR LHS)
(CAR RHS)) and we (RETURN TERM) which works only because we know this clause is the second to last one in the parent COND. *)

```
                (for DEST in (CDR (ASSOC (CAR (for X in SHELL-ALIST when (IEQP TYPE-SET (LOGBIT (CDR X)))
                                                 do (RETURN X)))
                                         SHELL-POCKETS))
                   do (SETQ TEMP-TEMP (REWRITE (FCONS-TERM* (QUOTE EQUAL)
                                                            (FCONS-TERM* DEST (QUOTE LHS))
                                                            (FCONS-TERM* DEST (QUOTE RHS)))
                                               (LIST (CONS (QUOTE LHS)
                                                           LHS)
                                                     (CONS (QUOTE RHS)
                                                           RHS))
                                               TYPE-ALIST
                                               (QUOTE ?)
                                               (QUOTE ID)
                                               NIL))
                      (COND
                         ((EQUAL TEMP-TEMP FALSE)
                          (RETURN FALSE))
                         ((NOT (EQUAL TEMP-TEMP TRUE))
                          (RETURN TERM)))
                   finally (RETURN TRUE)))
                (T TERM)))
            ((SETQ PAIR (ASSOC (FFN-SYMB TERM)
                               RECOGNIZER-ALIST))
             (SETQ TYPE-SET (TYPE-SET (FARGN TERM 1)))
             (COND
                ((LOGSUBSETP TYPE-SET (CDR PAIR))
                 TRUE)
                ((IEQP 0 (LOGAND TYPE-SET (CDR PAIR)))
                 FALSE)
                (T TERM)))
            (T TERM])
```

(REWRITE-WITH-LEMMAS

```
  [LAMBDA (TERM)                                          (* kbr: "19-Oct-85 16:31")
    (LET (REWRITTEN-TERM UNIFY-SUBST TEMP INST-HYP)
```

```
(COND
   ((VARIABLEP TERM)
    (REWRITE-SOLIDIFY TERM))
   ((FQUOTEP TERM)
    TERM)
   ((MEMB (FFN-SYMB TERM)
          FNS-TO-BE-IGNORED-BY-REWRITE)
    TERM)
   ((AND (OR (NEQ (FFN-SYMB TERM)
                  (QUOTE LESSP))
             (NOT (MEMB (QUOTE LESSP)
                        FNSTACK)))
         (REWRITE-WITH-LINEAR TERM)))
   ((for LEMMA in (GETPROP (FFN-SYMB TERM)
                           (QUOTE LEMMAS))
         unless (DISABLEDP (fetch (REWRITE-RULE NAME) of LEMMA))
         thereis (COND
                    ((META-LEMMAP LEMMA)
```

(* The conclusion is the name of a LISP fn to apply to the term being rewritten.
To add such lemma it must be the case that the LISP function return a TERMP such that in the current history
(EQUAL TERM val) can be proved. *)

```
                     (SETQ REWRITTEN-TERM (APPLY* (fetch (REWRITE-RULE CONCL) of LEMMA)
                                                  TERM))
                     (COND
                        ((EQUAL REWRITTEN-TERM TERM)
                         NIL)
                        (T
```

(* Because of the FORMP part of the correctness proof for user defined metafunctions we know REWRITTEN-TERM is a TERMP. However, we want all terms inside the theorem prover to be in quote normal form --
all explicit values be represented with QUOTE. We normalize REWRITTEN-TERM by applying the empty substitution to it. When we wrote the metapaper we were uncertain whether it was essential to the soundness of the theorem-prover that terms be in quote normal form -- however the theorem-prover could certainly be implemented so that it was not crucial so we left this issue out of the paper. We attempted to verify that the soundness of the current implementation did not depend upon terms being in quote normal form, but we got very weary, particularly because one of us could never remember what it was that we were trying to prove. We did learn that some parts of the theorem prover that used functions such as OCCUR would be heuristically inaccurate if terms were not in normal form.
We never discovered any situation in which terms not being in normal form would cause unsoundness but we did not get past the C's in an alphabetical scan. Instead, we gave up the search and decided to require that terms be in normal form throughout the theorem-prover. We still have not yet completed a pass through the theorem-prover checking that normalcy is preserved, but we believe that we were thorough in the initial *1*-reformulation of the theorem-prover --
never constructing a term except through CONS-TERM (unless we really knew what we were doing, such as consing up an IF term in rewrite)%. Our confidence that we were thorough during the *1*-reformulation is based upon the existence of a comment in CONS-TERM claiming that every term had to be in normal form.
*)

```
                           (PUSH-LEMMA (fetch (REWRITE-RULE NAME) of LEMMA))
                           (SETQ REWRITTEN-TERM (REWRITE (SUBLIS-VAR NIL REWRITTEN-TERM)
                                                         NIL TYPE-ALIST OBJECTIVE ID-IFF DEFN-FLG))
                           T)))
                  ((EQ (FFN-SYMB (fetch (REWRITE-RULE CONCL) of LEMMA))
                       (QUOTE NOT))
                   (COND
                      ((AND (OR (NULL (fetch (REWRITE-RULE HYPS) of LEMMA))
                                (NEQ OBJECTIVE (QUOTE TRUE)))
                            (ONE-WAY-UNIFY (FARGN (fetch (REWRITE-RULE CONCL) of LEMMA)
                                                  1)
                                           TERM)
                            (RELIEVE-HYPS (fetch (REWRITE-RULE HYPS) of LEMMA)
                                          (fetch (REWRITE-RULE NAME) of LEMMA)))
                       (PUSH-LEMMA (fetch (REWRITE-RULE NAME) of LEMMA))
                       (SETQ REWRITTEN-TERM FALSE)
                       T)
                      (T NIL)))
                  ((EQ (FFN-SYMB (fetch (REWRITE-RULE CONCL) of LEMMA))
                       (QUOTE EQUAL))
                   (COND
                      ((AND (OR (NULL (fetch (REWRITE-RULE HYPS) of LEMMA))
                                (NEQ OBJECTIVE (QUOTE TRUE))
                                (NOT (EQUAL (FARGN (fetch (REWRITE-RULE CONCL) of LEMMA)
                                                   2)
                                            FALSE)))
                            [OR (NOT (MEMB (FFN-SYMB TERM)
                                           FNSTACK))
                                (NOT (FNNAMEP (FFN-SYMB TERM)
                                              (FARGN (fetch (REWRITE-RULE CONCL) of LEMMA)
                                                     2]
                            (ONE-WAY-UNIFY (FARGN (fetch (REWRITE-RULE CONCL) of LEMMA)
                                                  1)
                                           TERM)
                            (PROGN (SETQ TEMP COMMUTED-EQUALITY-FLG)
                                   T)
                            [for PAIR in (fetch (REWRITE-RULE LOOP-STOPPER) of LEMMA)
                               never (TERM-ORDER (CDR (ASSOC (CAR PAIR)
                                                             UNIFY-SUBST))
```

```
                                                (CDR (ASSOC (CDR PAIR)
                                                            UNIFY-SUBST]
                                                     (RELIEVE-HYPS (fetch (REWRITE-RULE HYPS) of LEMMA)
                                                                   (fetch (REWRITE-RULE NAME) of LEMMA)))
                                                (SETQ REWRITTEN-TERM (REWRITE (COND
                                                                                  (TEMP (COMMUTE-EQUALITIES
                                                                                            (FARGN (fetch (REWRITE-RULE CONCL)
                                                                                                          of LEMMA)
                                                                                                   2)))
                                                                                  (T (FARGN (fetch (REWRITE-RULE CONCL)
                                                                                                   of LEMMA)
                                                                                            2)))
                                                                              UNIFY-SUBST TYPE-ALIST OBJECTIVE ID-IFF DEFN-FLG))
                                                (PUSH-LEMMA (fetch (REWRITE-RULE NAME) of LEMMA))
                                                T)
                                               ((AND (OR (NULL (fetch (REWRITE-RULE HYPS) of LEMMA))
                                                         (NEQ OBJECTIVE (QUOTE FALSE)))
                                                     (EQ (FFN-SYMB TERM)
                                                         (QUOTE EQUAL))
                                                     (ONE-WAY-UNIFY (fetch (REWRITE-RULE CONCL) of LEMMA)
                                                                    TERM)
                                                     (RELIEVE-HYPS (fetch (REWRITE-RULE HYPS) of LEMMA)
                                                                   (fetch (REWRITE-RULE NAME) of LEMMA)))
                                                (PUSH-LEMMA (fetch (REWRITE-RULE NAME) of LEMMA))
                                                (SETQ REWRITTEN-TERM TRUE)
                                                T)
                                               (T NIL)))
                                          ((AND (OR (NULL (fetch (REWRITE-RULE HYPS) of LEMMA))
                                                    (NEQ OBJECTIVE (QUOTE FALSE)))
                                                (OR (EQ ID-IFF (QUOTE IFF))
                                                    (BOOLEAN TERM))
                                                (ONE-WAY-UNIFY (fetch (REWRITE-RULE CONCL) of LEMMA)
                                                               TERM))
                                           (COND
                                              ((RELIEVE-HYPS (fetch (REWRITE-RULE HYPS) of LEMMA)
                                                             (fetch (REWRITE-RULE NAME) of LEMMA))
                                               (PUSH-LEMMA (fetch (REWRITE-RULE NAME) of LEMMA))
                                               (SETQ REWRITTEN-TERM TRUE)
                                               T)
                                              (T NIL)))
                                          (T NIL)))
                           REWRITTEN-TERM)
                          ((MEMBER TERM EXPAND-LST)
```

(* If we have been told to expand this term, do it. We used to do this inside of REWRITE-FNCALL, but there to avoid jumping out when we hit unapproved recursive calls we just substituted the actuals into the body and returned that. This seems neater. *)

```
   (SETQ TEMP (GETPROP (FFN-SYMB TERM)
                       (QUOTE SDEFN)))
   (PUSH-LEMMA (FFN-SYMB TERM))
   (REWRITE (CADDR TEMP)
            (for V in (CADR TEMP) as X in (FARGS TERM) collect (CONS V X))
            TYPE-ALIST OBJECTIVE ID-IFF DEFN-FLG))
  (T (SETQ TEMP (REWRITE-FNCALL (FFN-SYMB TERM)
                                (FARGS TERM)))
     (COND
        ((EQUAL TEMP TERM)
         TERM)
        ((CONTAINS-REWRITEABLE-CALLP (FFN-SYMB TERM)
                                     TEMP)
         (REWRITE TEMP NIL TYPE-ALIST OBJECTIVE ID-IFF DEFN-FLG))
        (T TEMP])
```

(REWRITE-WITH-LINEAR

```
[LAMBDA (TERM)                                                  (* kbr: "19-Oct-85 16:31")
  (PROG (ANS TEMP)
        (SETQ TEMP TERM)
        (BM-MATCH TEMP (NOT TEMP))                              (* TEMP is the atom of TERM. *)
        [COND
           ([AND (NOT (BM-MATCH TEMP (LESSP & &)))
                 (NOT (BM-MATCH TEMP (EQUAL & &]
            NIL)
           ((EQ OBJECTIVE (QUOTE ?))
```

(* We tried rewriting with linear under the objective ?, and it cost us 4 million conses over a proveall, so we stopped rewriting with linear under the objective ?. We found that too restrictive, and experimented with the idea of only rewriting with linear under ? when ANCESTORS is nonNIL, i.e., when we are working on a term that may appear as part of the simplification of the theorem as opposed to a term that appears while rewriting the hypothesis of a rewrite rule.
That cost us 5 times more conses on the theorem it was designed to prove! So we have abandoned linear under ? altogether, again. Here, however is the most recent experimental code:
(COND ((AND (NULL ANCESTORS) (EQ (ADD-TERM-TO-POT-LST TERM SIMPLIFY-CLAUSE-POT-LST NIL NIL)
(QUOTE CONTRADICTION))) (SETQ ANS TRUE) (GO WIN)))
(COND ((AND (NULL ANCESTORS) (EQ (ADD-TERM-TO-POT-LST TERM SIMPLIFY-CLAUSE-POT-LST T NIL)
(QUOTE CONTRADICTION))) (SETQ ANS FALSE) (GO WIN))) *)

```
             NIL)
            [(EQ OBJECTIVE (QUOTE TRUE))
             (COND
                ((EQ (ADD-TERM-TO-POT-LST TERM SIMPLIFY-CLAUSE-POT-LST NIL NIL)
                     (QUOTE CONTRADICTION))
                 (SETQ ANS TRUE)
                 (GO WIN]
            (T (COND
                  ((EQ (ADD-TERM-TO-POT-LST TERM SIMPLIFY-CLAUSE-POT-LST T NIL)
                       (QUOTE CONTRADICTION))
                   (SETQ ANS FALSE)
                   (GO WIN]
         (RETURN NIL)
      WIN (for X in LEMMAS-USED-BY-LINEAR do (PUSH-LEMMA X))
         (PUSH-LEMMA (QUOTE ZERO))
         (for X in LINEAR-ASSUMPTIONS do (PUSH-LINEARIZE-ASSUMPTION X))
         (RETURN ANS])
```

(RPLACAI

```
  [LAMBDA (LIST I X)                                        (* kbr: "19-Oct-85 16:31")
    (COND
       ((IEQP I 1)
        (RPLACA (OR LIST (CONS NIL NIL))
                X))
       (T (RPLACD (OR LIST (CONS NIL NIL))
                  (RPLACAI (CDR LIST)
                           (SUB1 I)
                           X])

)

(RPAQQ CODE-S-ZCOMS
       ((* CODE-S-Z *)
        (FNS S SARGS SCONS-TERM SCRUNCH SCRUNCH-CLAUSE SCRUNCH-CLAUSE-SET SEARCH-GROUND-UNITS
             SEQUENTIAL-DIFFERENCE SET-DIFF SET-DIFF-N SET-EQUAL SET-SIMPLIFY-CLAUSE-POT-LST SETTLED-DOWN-CLAUSE
             SETTLED-DOWN-SENT SETUP SETUP-META-NAMES SHELL-CONSTRUCTORP SHELL-DESTRUCTOR-NESTP SHELL-OCCUR
             SHELL-OCCUR1 SHELLP SIMPLIFY-CLAUSE SIMPLIFY-CLAUSE-MAXIMALLY SIMPLIFY-CLAUSE-MAXIMALLY1
             SIMPLIFY-CLAUSE0 SIMPLIFY-CLAUSE1 SIMPLIFY-LOOP SIMPLIFY-SENT SINGLETON-CONSTRUCTOR-TO-RECOGNIZER
             SKO-DEST-NESTP SOME-SUBTERM-WORSE-THAN-OR-EQUAL SORT-DESTRUCTOR-CANDIDATES SOUND-IND-PRIN-MASK
             STACK-DEPTH START-STATS STOP-STATS STORE-SENT STRIP-BRANCHES STRIP-BRANCHES1 SUB-SEQUENCEP SUBBAGP
             SUBLIS-EXPR SUBLIS-EXPR1 SUBLIS-VAR SUBLIS-VAR-LST SUB-PAIR-EXPR SUB-PAIR-EXPR-LST SUB-PAIR-EXPR1
             SUB-PAIR-VAR SUB-PAIR-VAR-LST SUBST-EXPR SUBST-EXPR-ERROR1 SUBST-EXPR-LST SUBST-EXPR1 SUBST-FN
             SUBST-VAR SUBST-VAR-LST BM-SUBST SUBSUMES SUBSUMES-REWRITE-RULE SUBSUMES1 SUBSUMES11
             SUM-STATS-ALIST TABULATE TERM-ORDER TERMINATION-MACHINE TP-EXPLODEN1 TP-GETCHARN1 TP-IMPLODE1
             TO-BE-IGNOREDP TOO-MANY-IFS TOP-FNNAME TOTAL-FUNCTIONP TRANSITIVE-CLOSURE TRANSLATE
             TRANSLATE-TO-LISP TREE-DEPENDENTS TRIVIAL-POLYP TRIVIAL-POLYP1 TRUE-POLYP TYPE-ALIST-CLAUSE
             TYPE-PRESCRIPTION-LEMMAP TYPE-SET TYPE-SET2 UBT UNBREAK-LEMMA UNCHANGING-VARS UNCHANGING-VARS1
             UNDO-BACK-THROUGH UNDO-NAME UNION-EQUAL UNPRETTYIFY VARIANTP WORSE-THAN WORSE-THAN-OR-EQUAL WRAPUP
             XXXJOIN ZERO-POLY)))


          (* * CODE-S-Z *)

(DEFINEQ

(S
  [LAMBDA (VAR VAL)                                         (* kbr: "19-Oct-85 16:31")
    (COND
       ([NOT (ERRSET (SETQ TEMP-TEMP (TRANSLATE VAR]
        NIL)
       ((OR (NEQ VAR TEMP-TEMP)
            (NOT (VARIABLEP VAR)))
        (QUOTE (NOT VARIABLEP)))
       ([NOT (ERRSET (SETQ VAL (TRANSLATE VAL]
        NIL)
       ((NOT (QUOTEP VAL))
        (QUOTE (NOT QUOTEP)))
       (T [SETQ TEMP-TEMP (OR (ASSOC VAR R-ALIST)
                              (CAR (SETQ R-ALIST (CONS (CONS VAR VAL)
                                                       R-ALIST]
          (RPLACD TEMP-TEMP (CADR VAL))
          VAR])

(SARGS
  [LAMBDA (TERM)                                            (* kbr: "19-Oct-85 16:31")
    (COND
       ((NEQ (CAR TERM)
             (QUOTE QUOTE))
        (CDR TERM))
       [(LITATOM (CADR TERM))
        (COND
           ((EQ (CADR TERM)
                *1*T)
            NIL)
           ((EQ (CADR TERM)
```

```
               *1*F)
            NIL)
           (T (LIST (LIST (QUOTE QUOTE)
                          (DTACK-0-ON-END (CHCON (CADR TERM]
      [(FIXP (CADR TERM))
       (COND
          [(LESSP (CADR TERM)
                  0)
           (LIST (LIST (QUOTE QUOTE)
                       (MINUS (CADR TERM]
          ((EQUAL (CADR TERM)
                  0)
           NIL)
          (T (LIST (LIST (QUOTE QUOTE)
                         (SUB1 (CADR TERM]
      ((EQ (CAR (CADR TERM))
           *1*SHELL-QUOTE-MARK)
       (for X in (CDDR (CADR TERM)) collect (LIST (QUOTE QUOTE)
                                                  X)))
      (T (LIST (LIST (QUOTE QUOTE)
                     (CAR (CADR TERM)))
               (LIST (QUOTE QUOTE)
                     (CDR (CADR TERM])
```

(SCONS-TERM

```
[LAMBDA (FN ARGS)                                          (* kbr: "19-Oct-85 16:31")
  (COND
     [(EQ FN (QUOTE EQUAL))
      (COND
         ((EQUAL (CAR ARGS)
                 (CADR ARGS))
          TRUE)
         ((AND (QUOTEP (CAR ARGS))
               (QUOTEP (CADR ARGS)))
          FALSE)
         (T (CONS (QUOTE EQUAL)
                  ARGS]
     (T (CONS-TERM FN ARGS])
```

(SCRUNCH

```
[LAMBDA (L)                                                (* kbr: " 4-Jul-86 18:10")
                                                           (* Setifies list L *)
  (for TAIL on L unless (MEMBER (CAR TAIL)
                                (CDR TAIL))
     collect (CAR TAIL])
```

(SCRUNCH-CLAUSE

```
[LAMBDA (CL)                                               (* kbr: "19-Oct-85 16:31")
  (for TAIL on CL unless (OR (AND (FALSE-NONFALSEP (CAR TAIL))
                                  DEFINITELY-FALSE)
                             (MEMBER (CAR TAIL)
                                     (CDR TAIL)))
     collect (CAR TAIL])
```

(SCRUNCH-CLAUSE-SET

```
[LAMBDA (CLAUSES)                                          (* kbr: "19-Oct-85 16:31")
  (TRANSITIVE-CLOSURE (for CL in CLAUSES collect (SCRUNCH-CLAUSE CL))
         (FUNCTION (LAMBDA (CL1 CL2)
                     (COND
                        ((SUBSETP CL1 CL2)
                         CL1)
                        ((SUBSETP CL2 CL1)
                         CL2)
                        (T NIL])
```

(SEARCH-GROUND-UNITS

```
[LAMBDA (HYP)                                              (* kbr: "19-Oct-85 16:31")
                                                           (* Like LOOKUP-HYP except looks through ground unit
                                                           REWRITE lemmas. *)
  (PROG (TERM FN REWRITE-RULE)
        [COND
           [(BM-MATCH HYP (NOT TERM))
            (COND
               ((VARIABLEP TERM)
                (RETURN NIL))
               ((FQUOTEP TERM)
                (RETURN (EQUAL TERM FALSE)))
               (T (SETQ FN (FFN-SYMB TERM]
           ((VARIABLEP HYP)
            (RETURN NIL))
           [(FQUOTEP HYP)
            (RETURN (NOT (EQUAL HYP FALSE]
```

```
            (T (SETQ FN (FFN-SYMB HYP]
         (COND
            ((SETQ REWRITE-RULE (for REWRITE-RULE in (GET1 FN (QUOTE LEMMAS))
                                   when (AND (NOT (DISABLEDP (fetch (REWRITE-RULE NAME) of REWRITE-RULE)))
                                             (NOT (META-LEMMAP REWRITE-RULE))
                                             (NOT (fetch (REWRITE-RULE HYPS) of REWRITE-RULE))
                                             (NOT (FREE-VARSP (fetch (REWRITE-RULE CONCL) of REWRITE-RULE)
                                                              NIL))
                                             (ONE-WAY-UNIFY1 HYP (fetch (REWRITE-RULE CONCL) of REWRITE-RULE)))
                                   do (RETURN REWRITE-RULE)))
             (PUSH-LEMMA (fetch (REWRITE-RULE NAME) of REWRITE-RULE))
             (RETURN T))
            (T (RETURN NIL])
```

(SEQUENTIAL-DIFFERENCE

```
  [LAMBDA (SMALLER LARGER)                                    (* kbr: "19-Oct-85 16:31")
    (COND
       ((NLISTP SMALLER)
        LARGER)
       ((NLISTP LARGER)
        (QUOTE NOT-RELATED))
       ((EQUAL (CAR SMALLER)
               (CAR LARGER))
        (SEQUENTIAL-DIFFERENCE (CDR SMALLER)
               (CDR LARGER)))
       (T (SETQ TEMP-TEMP (SEQUENTIAL-DIFFERENCE SMALLER (CDR LARGER)))
          (COND
             ((EQ TEMP-TEMP (QUOTE NOT-RELATED))
              (QUOTE NOT-RELATED))
             (T (CONS (CAR LARGER)
                      TEMP-TEMP])
```

(SET-DIFF

```
  [LAMBDA (X Y)                                               (* kbr: "19-Oct-85 16:31")
    (for ELE in X unless (MEMBER ELE Y) collect ELE])
```

(SET-DIFF-N

```
  [LAMBDA (BIG LITTLE N)                                      (* kbr: "19-Oct-85 16:31")
    (COND
       ((ZEROP N)
        NIL)
       ((NLISTP BIG)
        (ERROR1 (PQUOTE (PROGN SET-DIFF-N CALLED WITH INAPPROPRIATE ARGUMENTS %.))
               (BINDINGS)
               (QUOTE HARD)))
       ((MEMB (CAR BIG)
              LITTLE)
        (SET-DIFF-N (CDR BIG)
              LITTLE N))
       (T (CONS (CAR BIG)
                (SET-DIFF-N (CDR BIG)
                       LITTLE
                       (SUB1 N])
```

(SET-EQUAL

```
  [LAMBDA (X Y)                                               (* kbr: "19-Oct-85 16:31")
    (AND (SUBSETP X Y)
         (SUBSETP Y X])
```

(SET-SIMPLIFY-CLAUSE-POT-LST

```
  [LAMBDA (CL HEURISTIC-TYPE-ALIST)                           (* kbr: "19-Oct-85 16:31")

          (* We use the same basic pot list for all the calls REWRITE for a given clause.
          However, to keep from biting our tail, we must know which literals each poly descends from and avoid the polys descending
          from the negation of our current lit. In order to keep track of which literals are being used we set TYPE-ALIST to NIL before
          setting up the pot list, and use the special hacks LITS-THAT-MAY-BE-ASSUMED-FALSE and HEURISTIC-TYPE-ALIST.
          The pot list we thus construct is immediately tested against CONTRADICTION to see if CL is a consequence of linear.
          However, the failure to use everything we know has burned us here.
          In particular, the type alist might contain an equality that could be used as a rewrite rule to help us establish the hypothesis
          of some needed lemma. Imagine for example that the clause contains b=a and p
          (a) as hyps and we need to prove p (b) to get some lemma. We try to handle this as follows.
          After setting up SIMPLIFY-CLAUSE-POT-LST -- the pot list we will use subsequently and which has all the dependencies
          carefully tracked -- we go at the pot list again with the ALL-NEW-FLG of ADD-TERMS-TO-POT-LST set to T.
          This causes us to treat every addend in the pot list as new and reconsider the adding of all the lemmas.
          If this produces CONTRADICTION, we win. If not, we pretend we did not do it --
          since the resulting pot list has hidden dependencies in it. *)

    (LET ((LITS-THAT-MAY-BE-ASSUMED-FALSE CL)
          (TYPE-ALIST NIL))
         (SETQ SIMPLIFY-CLAUSE-POT-LST (ADD-TERMS-TO-POT-LST CL NIL NIL NIL))
         [COND
            ((NEQ SIMPLIFY-CLAUSE-POT-LST (QUOTE CONTRADICTION))
```

```
            (SETQ TYPE-ALIST HEURISTIC-TYPE-ALIST)
            (COND
               ((EQ (ADD-TERMS-TO-POT-LST NIL SIMPLIFY-CLAUSE-POT-LST NIL T)
                    (QUOTE CONTRADICTION))
                (SETQ SIMPLIFY-CLAUSE-POT-LST (QUOTE CONTRADICTION]
        NIL])
```

(SETTLED-DOWN-CLAUSE

```
[LAMBDA (CL HIST)                                          (* kbr: "19-Oct-85 16:31")
  (COND
     ((ASSOC (QUOTE SETTLED-DOWN-CLAUSE)
             HIST)
      NIL)
     (T (SETQ PROCESS-HIST NIL)
        (SETQ PROCESS-CLAUSES (LIST CL))
        T])
```

(SETTLED-DOWN-SENT

```
[LAMBDA (CL HIST)                                          (* kbr: "19-Oct-85 16:31")
  (EXECUTE (QUOTE SETTLED-DOWN-CLAUSE)
           CL HIST (QUOTE SIMPLIFY-SENT)
           (QUOTE ELIMINATE-DESTRUCTORS-SENT])
```

(SETUP

```
[LAMBDA (FORM CLAUSES LEMMAS)                              (* kbr: "19-Oct-85 16:31")
  (SETQ ORIGTHM FORM)
  [COND
     ((NOT (MEMBER ORIGTHM FAILED-THMS))
      (SETQ FAILED-THMS (CONS ORIGTHM FAILED-THMS]
  (SETQ EXPAND-LST HINTED-EXPANSIONS)
  (SETQ TERMS-TO-BE-IGNORED-BY-REWRITE NIL)
  (SETQ INDUCTION-HYP-TERMS NIL)
  (SETQ INDUCTION-CONCL-TERMS NIL)
  (SETQ ALL-LEMMAS-USED LEMMAS)
  (SETQ STACK NIL)
  (SETQ FNSTACK NIL)
  (SETQ LAST-PRINT-CLAUSES NIL)
  (SETQ TYPE-ALIST NIL)
  (SETQ LITS-THAT-MAY-BE-ASSUMED-FALSE NIL)
  (SETQ CURRENT-LIT 0)
  (SETQ CURRENT-ATM 0)
  (SETQ ANCESTORS NIL)
  (INIT-LEMMA-STACK)
  (INIT-LINEARIZE-ASSUMPTIONS-STACK)
  (SETQ LAST-PRINEVAL-CHAR NIL)
  (RANDOM-INITIALIZATION ORIGTHM)
  (IO (QUOTE SETUP)
      (LIST ORIGTHM)
      NIL CLAUSES LEMMAS])
```

(SETUP-META-NAMES

```
[LAMBDA NIL                                                (* kbr: "19-Oct-85 16:31")
  (ADD-FACT (QUOTE MEANING)
            (QUOTE LEMMAS)
            (create REWRITE-RULE
                    NAME _ (QUOTE MEANING)
                    CONCL _ (QUOTE MEANING-SIMPLIFIER)))
  (ADD-FACT (QUOTE FORMP)
            (QUOTE LEMMAS)
            (create REWRITE-RULE
                    NAME _ (QUOTE FORMP)
                    CONCL _ (QUOTE FORMP-SIMPLIFIER])
```

(SHELL-CONSTRUCTORP

```
[LAMBDA (TERM)                                             (* kbr: "19-Oct-85 16:31")
  (COND
     ((VARIABLEP TERM)
      NIL)
     (T (ASSOC (FN-SYMB TERM)
               SHELL-ALIST])
```

(SHELL-DESTRUCTOR-NESTP

```
[LAMBDA (VAR TERM)                                         (* kbr: "19-Oct-85 16:31")
  (COND
     ((VARIABLEP TERM)
      (EQ VAR TERM))
     ((FQUOTEP TERM)
      NIL)
     (T (AND (for POCKET in SHELL-POCKETS thereis (MEMB (FFN-SYMB TERM)
                                                       (CDR POCKET)))
             (SHELL-DESTRUCTOR-NESTP VAR (FARGN TERM 1])
```

(SHELL-OCCUR

```
[LAMBDA (TERM1 TERM2)                                   (* kbr: "19-Oct-85 16:31")
```

(* Returns T if TERM1 properly occurs in a nest of shells TERM2.
That is whether TERM1 occurs as an arg at some depth in the shell TERM2, and that the chain of shells from the occurrence to TERM1 all the way up to the top of TERM2 is properly typed.
See the comment in SHELL-OCCUR1. Does not bother to do anything if TERM1 is a SHELLP, because (assuming the terms are coming from EQUAL expressions) the two shells would be either different and we wouldn't be here, or the same, in which case they would be rewritten. At the moment the only fn to call SHELL-OCCUR is NOT-IDENT and we only use NOT-IDENT to decide EQUALs or else one of the two terms is FALSE.
*)

```
  (LET (TYPE-SET-TERM1)
       (COND
          ((SHELLP TERM1)
           NIL)
          ((VARIABLEP TERM2)
           NIL)
          ((FQUOTEP TERM2)
           NIL)
          [(ASSOC (FFN-SYMB TERM2)
                  SHELL-ALIST)
           (SETQ TYPE-SET-TERM1 (TYPE-SET TERM1))
           (for ARG in (FARGS TERM2) as TR in (GET1 (FFN-SYMB TERM2)
                                                    (QUOTE TYPE-RESTRICTIONS))
              thereis (AND (SETQ TEMP-TEMP (SHELL-OCCUR1 TERM1 ARG))
                           (LOGSUBSETP TEMP-TEMP (fetch (TYPE-RESTRICTION TYPE-SET) of TR]
          (T NIL]))
```

(SHELL-OCCUR1

```
[LAMBDA (TERM1 TERM2)                                   (* kbr: "19-Oct-85 16:31")
```

(* This function wants to see whether TERM1 occurs as an arg to a shell in TERM2.
However, because of type restrictions, one must not be fooled into thinking that, for example,
(ADD1 0) occurs inside of (ADD1 (CONS (ADD1 0) NIL)) despite the fact that it occurs as an arg to a shell.
The basic idea is that TERM1 must either be TERM2 or else must shell-occur inside the shell TERM2 --
in a spot of the right type. Thus, one way to compute it would be to see if TERM1 shell-occurred in an arg position of shell TERM2 and if so to then determine if the typeset of the arg was suitable.
However, that would involve either a general purpose call on typeset or else looking ahead to see whether the arg in which TERM1 occurred was itself a shell -- in which case its typeset is just on its type-prescription --
or was a TERM1 occurrence itself -- in which case a full blown typeset is necessary.
Rather than do it that way we have fixed SHELL-OCCUR1 so that it returns the typeset of TERM2 if an occurrence was found, and otherwise NIL. *)

```
  (COND
     ((EQUAL TERM1 TERM2)
      TYPE-SET-TERM1)
     ((VARIABLEP TERM2)
      NIL)
     ((FQUOTEP TERM2)
      NIL)
     [[AND (ASSOC (FFN-SYMB TERM2)
                  SHELL-ALIST)
           (for ARG in (FARGS TERM2) as TR in (GET1 (FFN-SYMB TERM2)
                                                    (QUOTE TYPE-RESTRICTIONS))
              thereis (AND (SETQ TEMP-TEMP (SHELL-OCCUR1 TERM1 ARG))
                           (LOGSUBSETP TEMP-TEMP (fetch (TYPE-RESTRICTION TYPE-SET) of TR]
      (CAR (TYPE-PRESCRIPTION (FFN-SYMB TERM2]
     (T NIL])
```

(SHELLP

```
[LAMBDA (TERM)                                          (* kbr: "19-Oct-85 16:31")
  (COND
     ((VARIABLEP TERM)
      NIL)
     ((FQUOTEP TERM)
      T)
     (T (OR (MEMB (FFN-SYMB TERM)
                  *1*BTM-OBJECTS)
            (ASSOC (FFN-SYMB TERM)
                   SHELL-ALIST])
```

(SIMPLIFY-CLAUSE

```
[LAMBDA (CURRENT-SIMPLIFY-CL HIST)                      (* kbr: "19-Oct-85 22:43")
```

(* If T is returned, then the conjunction of PROCESS-CLAUSES implies CURRENT-SIMPLIFY-CL.
Equivalently, if T is returned, then under the assumption that CURRENT-SIMPLIFY-CL is F, CURRENT-SIMPLIFY-CL is equivalent to the conjunction of PROCESS-CLAUSES. Note that PROCESS-CLAUSES may be the facetious answer F, i.e., false generalization may and does happen. We know such tail biting can occur through use of linear arithmetic.
We are uncertain whether it can occur without use of linear arithmetic.
To make it happen with linear we just need two slightly different versions of the same inequality literal.
The poly arising from the second is used to rewrite the first to false and the poly arising from the first --
which is still in the pot list -- is used to rewrite the second to false.

```
          LITS-TO-BE-IGNORED-BY-LINEAR actually prevents this direct example from working --
          the poly arising from the first is ignored after its literal has been rewritten to false.
          To overcome this minor obstacle, it is necessary to cause the first literal to be rewritten to something that will prove to be
          false eventually but isn't syntactically F. *)

    (LET (ANS (TERMS-TO-BE-IGNORED-BY-REWRITE TERMS-TO-BE-IGNORED-BY-REWRITE)
              (EXPAND-LST EXPAND-LST))
         (PROG NIL
               [COND
                  [(SETQ TEMP-TEMP (ASSOC (QUOTE SETTLED-DOWN-CLAUSE)
                                          HIST))

          (* The clause has settled down under rewriting with the INDUCTION-HYP-TERMS ignored and the
          INDUCTION-CONCL-TERMS forcibly expanded. In general then we now want to stop treating these terms specially and
          continue simplifying. However, there is a special case that will save a little time.
          Suppose that the clause just settled down -- that is, the most recent HIST entry is the settled mark.
          And suppose that none of the specially treated terms occurs in the clause we're to simplify.
          Then we needn't simplify it again. The first supposition is important.
          Imagine that the clause settled down long ago and we have done much since then.
          *)

                   (COND
                      ((AND (EQ TEMP-TEMP (CAR HIST))
                            (for TERM in INDUCTION-HYP-TERMS never (DUMB-OCCUR-LST TERM CURRENT-SIMPLIFY-CL)))

          (* Since we know the INDUCTION-CONCL-TERMS couldn't occur in the clause --
          they would have been expanded -- it suffices to check for just the hyp terms.
          This test should speed up base cases and the preinduction simplification at least.
          *)

                       (RETURN NIL]
                  (T

          (* The clause has not yet settled down, so arrange to ignore INDUCTION-HYP-TERMS during rewriting and to expand
          without question INDUCTION-CONCL-TERMS. *)

                     (SETQ TERMS-TO-BE-IGNORED-BY-REWRITE (APPEND INDUCTION-HYP-TERMS
                                                                  TERMS-TO-BE-IGNORED-BY-REWRITE))
                     (SETQ EXPAND-LST (APPEND INDUCTION-CONCL-TERMS EXPAND-LST]
               (INIT-LEMMA-STACK)
               (PUSH-LEMMA-FRAME)
               (SETQ PROCESS-CLAUSES (SIMPLIFY-CLAUSE0 CURRENT-SIMPLIFY-CL HIST))
               (SETQ PROCESS-HIST (for X in (POP-LEMMA-FRAME) unless (AND (LISTP X)
                                                                          (NLISTP (CAR X)))
                                         collect X))

          (* The lemmas ignored are really literals from LITS-THAT-MAY-BE-ASSUMED-FALSE that get put in by
          REWRITE-SOLIDIFY. The identifying test for these literals is not a simple LISTP because
          PROCESS-EQUATIONAL-POLYS puts in some LISTP elements to encode its additions to the clause and we must
          preserve them. *)

               (for X in PROCESS-HIST unless (OR (LISTP X)
                                                 (MEMB X ALL-LEMMAS-USED))
                  do (SETQ ALL-LEMMAS-USED (CONS X ALL-LEMMAS-USED)))
               (RETURN (NOT (AND (IEQP (LENGTH PROCESS-CLAUSES)
                                       1)
                                 (EQUAL (CAR PROCESS-CLAUSES)
                                        CURRENT-SIMPLIFY-CL])
```

(SIMPLIFY-CLAUSE-MAXIMALLY

```
  [LAMBDA (CURRENT-CL)                                                    (* kbr: "19-Oct-85 16:31")
    (LET (SIMPLIFY-CLAUSE-MAXIMALLY-CLAUSES SIMPLIFY-CLAUSE-MAXIMALLY-HIST)
         (SIMPLIFY-CLAUSE-MAXIMALLY1 CURRENT-CL)
         (SETQ PROCESS-HIST SIMPLIFY-CLAUSE-MAXIMALLY-HIST)
         (SETQ PROCESS-CLAUSES SIMPLIFY-CLAUSE-MAXIMALLY-CLAUSES)
         (NOT (EQUAL PROCESS-CLAUSES (LIST CURRENT-CL])
```

(SIMPLIFY-CLAUSE-MAXIMALLY1

```
  [LAMBDA (CL)                                                            (* kbr: "19-Oct-85 16:31")
    (COND
       ((SIMPLIFY-CLAUSE CL NIL)
        (for X in PROCESS-HIST unless (OR (LISTP X)
                                          (MEMB X SIMPLIFY-CLAUSE-MAXIMALLY-HIST))
           do (SETQ SIMPLIFY-CLAUSE-MAXIMALLY-HIST (CONS X SIMPLIFY-CLAUSE-MAXIMALLY-HIST)))
        (for CL in PROCESS-CLAUSES do (SIMPLIFY-CLAUSE-MAXIMALLY1 CL)))
       (T (SETQ SIMPLIFY-CLAUSE-MAXIMALLY-CLAUSES (CONS CL SIMPLIFY-CLAUSE-MAXIMALLY-CLAUSES])
```

(SIMPLIFY-CLAUSE0

```
  [LAMBDA (CL HIST)                                                       (* kbr: " 6-Jul-86 09:45")
                                                                          (* Called by SIMPLIFY-CLAUSE.
                                                                          *)
    (PROG (TYPE-ALIST SIMPLIFY-CLAUSE-POT-LST CLS NEG-HYPS)
          (SETQ CL (REMOVE-TRIVIAL-EQUATIONS CL))
          (SETQ TYPE-ALIST (TYPE-ALIST-CLAUSE CL))
```

```
          (COND
             ((EQ (QUOTE CONTRADICTION)
                  TYPE-ALIST)
              (RETURN NIL)))
          (SET-SIMPLIFY-CLAUSE-POT-LST CL TYPE-ALIST)
          [COND
             ((EQ SIMPLIFY-CLAUSE-POT-LST (QUOTE CONTRADICTION))
              (SETQ CLS NIL))
             (T (SETQ CLS (LIST (PROCESS-EQUATIONAL-POLYS CL HIST SIMPLIFY-CLAUSE-POT-LST]
          (COND
             ((NOT (AND (IEQP (LENGTH CLS)
                              1)
                        (EQUAL (CAR CLS)
                               CL)))
              (PUSH-LEMMA (QUOTE ZERO))
              (for X in LEMMAS-USED-BY-LINEAR do (PUSH-LEMMA X))
              (SETQ LINEAR-ASSUMPTIONS (for HYP in LINEAR-ASSUMPTIONS unless (for LIT in CL
                                                                                thereis (COMPLEMENTARYP HYP LIT))
                                                collect HYP))
              (SETQ NEG-HYPS (for HYP in LINEAR-ASSUMPTIONS collect (DUMB-NEGATE-LIT HYP)))
              (SETQ CLS (for CL in CLS collect (DISJOIN-CLAUSES NEG-HYPS CL)))
              (for TERM in LINEAR-ASSUMPTIONS do (SETQ CLS (CONS (CONS TERM CL)
                                                                 CLS)))
              (RETURN CLS))
             (T (RETURN (SIMPLIFY-CLAUSE1 CL NIL NIL 1])
```

(SIMPLIFY-CLAUSE1

```
[LAMBDA (TAIL NEW-CLAUSE LITS-TO-BE-IGNORED-BY-LINEAR I)        (* kbr: " 6-Jul-86 09:46")
```

(* Called by SIMPLIFY-CLAUSE0. Returns a list of clauses whose conjunction is equivalent to the clause CL formed by appending TAIL to NEW-CLAUSE under the hypothesis of the polys in SIMPLIFY-CLAUSE-POT-LST and under the hypothesis that CL is false. *)

```
   (PROG (VAL SEGS TYPE-ALIST NEG-HYPS CURRENT-LIT CURRENT-ATM BRANCHES)
          (COND
             ((NULL TAIL)
              (RETURN (LIST NEW-CLAUSE)))
             (T (PRINT-TO-DISPLAY (QUOTE SIMPLIFY-CLAUSE)
                                  I NIL)
                (SETQ CURRENT-LIT (SETQ CURRENT-ATM (CAR TAIL)))
                (BM-MATCH CURRENT-ATM (NOT CURRENT-ATM))
                (SETQ LITS-TO-BE-IGNORED-BY-LINEAR (CONS CURRENT-LIT LITS-TO-BE-IGNORED-BY-LINEAR))
                (SETQ FNSTACK NIL)
                (SETQ TYPE-ALIST (TYPE-ALIST-CLAUSE NEW-CLAUSE))
                (COND
                   ((EQ TYPE-ALIST (QUOTE CONTRADICTION))
                    (RETURN NIL)))
                (SETQ TYPE-ALIST (TYPE-ALIST-CLAUSE (CDR TAIL)))
                (COND
                   ((EQ TYPE-ALIST (QUOTE CONTRADICTION))
                    (RETURN NIL)))
                (INIT-LINEARIZE-ASSUMPTIONS-STACK)
                (PUSH-LINEARIZE-ASSUMPTIONS-FRAME)
                (SETQ VAL (REWRITE CURRENT-ATM NIL TYPE-ALIST (QUOTE ?)
                                   (QUOTE IFF)
                                   NIL))
                [COND
                   ((NEQ CURRENT-LIT CURRENT-ATM)
                    (SETQ VAL (NEGATE-LIT VAL]
                (SETQ LINEAR-ASSUMPTIONS (POP-LINEARIZE-ASSUMPTIONS-FRAME))
                (SETQ NEG-HYPS (for HYP in LINEAR-ASSUMPTIONS collect (NEGATE-LIT HYP)))
                (SETQ BRANCHES (CLAUSIFY VAL))
                [SETQ SEGS (CONJOIN-CLAUSE-SETS (for SEG in BRANCHES collect (DISJOIN-CLAUSES NEG-HYPS SEG))
                                                (for HYP in LINEAR-ASSUMPTIONS bind (CL _ (ADD-LITERAL (PEGATE-LIT CURRENT-LIT)
                                                                                                       NIL NIL))
                                                      collect (ADD-LITERAL HYP CL NIL]
                (RETURN (for SEG in SEGS join (SIMPLIFY-CLAUSE1 (CDR TAIL)
                                                                (APPEND NEW-CLAUSE SEG)
                                                                (COND
                                                                   ((EQUAL BRANCHES (QUOTE (NIL)))
                                                                    LITS-TO-BE-IGNORED-BY-LINEAR)
                                                                   (T (CDR LITS-TO-BE-IGNORED-BY-LINEAR)))
                                                                (ADD1 I])
```

(SIMPLIFY-LOOP

```
[LAMBDA (CLAUSES)                                               (* kbr: "20-Oct-85 15:36")
```

(* This function just serves as a target for the RETFROM in STORE-SENT in the event that we are working on the original input and find that we have split it into more than one goal and want to back up and use induction on the input term. *)

```
   (for CURRENT-CL in CLAUSES do (SIMPLIFY-SENT CURRENT-CL NIL])
```

(SIMPLIFY-SENT

```
[LAMBDA (CL HIST)                                                    (* kbr: "19-Oct-85 16:31")
  (EXECUTE (QUOTE SIMPLIFY-CLAUSE)
         CL HIST (QUOTE SIMPLIFY-SENT)
         (QUOTE SETTLED-DOWN-SENT])
```

(SINGLETON-CONSTRUCTOR-TO-RECOGNIZER

```
[LAMBDA (FNNAME)                                                     (* kbr: "19-Oct-85 16:31")
  (COND
     ((SETQ TEMP-TEMP (ASSOC FNNAME SHELL-ALIST))
      (SETQ TEMP-TEMP (LSH 1 (CDR TEMP-TEMP)))
      (COND
         [(MEMBER TEMP-TEMP SINGLETON-TYPE-SETS)
          (CAR (for PAIR in RECOGNIZER-ALIST when (EQUAL TEMP-TEMP (CDR PAIR)) do (RETURN PAIR]
         (T NIL)))
     (T NIL])
```

(SKO-DEST-NESTP

```
[LAMBDA (TERM DEEPFLG)                                               (* kbr: "19-Oct-85 16:31")
  (COND
     ((VARIABLEP TERM)
      T)
     ((FQUOTEP TERM)
      NIL)
     [[AND (SETQ TEMP-TEMP (GET1 (FFN-SYMB TERM)
                                 (QUOTE ELIMINATE-DESTRUCTORS-SEQ)))
           (NOT (DISABLEDP (fetch (REWRITE-RULE NAME) of TEMP-TEMP]
      (COND
         (DEEPFLG (for X in (FARGS TERM) always (SKO-DEST-NESTP X DEEPFLG)))
         (T (for X in (FARGS TERM) always (VARIABLEP X]
     (T NIL])
```

(SOME-SUBTERM-WORSE-THAN-OR-EQUAL

```
[LAMBDA (TERM1 TERM2)                                                (* kbr: "19-Oct-85 16:31")

        (* Returns T if some subterm of TERM1 is WORSE-THAN or EQUAL to TERM2 itself.
        *)

  (COND
     ((VARIABLEP TERM1)
      (EQ TERM1 TERM2))
     ((OR (EQUAL TERM1 TERM2)
          (QUICK-WORSE-THAN TERM1 TERM2))
      T)
     ((FQUOTEP TERM1)
      NIL)
     (T (for ARG in (FARGS TERM1) thereis (SOME-SUBTERM-WORSE-THAN-OR-EQUAL ARG TERM2])
```

(SORT-DESTRUCTOR-CANDIDATES

```
[LAMBDA (LST)                                                        (* kbr: "22-Oct-85 15:37")

        (* Each element of LST is a list of NVARIABLEP nonQUOTEP terms.
        We sort them into descending order according to the sum of the level numbers of the fn symbols of the terms in the CDR of
        each element. INTERLISP's SORT is apparently nonstable and frequently
        (perhaps always) reverses elements of equal weight. Zetalisp sort is stable.
        We found three occasions in the rsa and wilson proofs when this difference bit us and caused a different elimination to be
        chosen first. The first two times we fixed it by letting it do the new elim and just seeing that the appropriate lemmas were
        available to handle the new goals. But on the third time we decided simply to REVERSE the input list to mimic INTERLISP's
        sort, just so we could get on with reproducing the old proofs on the new machine.
        *)

  (SORT (REVERSE LST)
        (FUNCTION (LAMBDA (X Y)
                    (GREATERP (for TERM in (CDR X) sum (GET-LEVEL-NO (FFN-SYMB TERM)))
                              (for TERM in (CDR Y) sum (GET-LEVEL-NO (FFN-SYMB TERM])
```

(SOUND-IND-PRIN-MASK

```
[LAMBDA (TERM JUSTIFICATION FORMALS QUICK-BLOCK-INFO)                (* kbr: "19-Oct-85 16:31")

        (* TERM is a term we are considering doing induction for. JUSTIFICATION is one of the justifications associated with the
        function symbol of TERM. FORMALS is the formals list of the fn and QUICK-BLOCK-INFO is the obvious.
        JUSTIFICATION and the machine for fn describe an induction.
        We wish to determine, in the terminology of ACL, whether the induction applies to TERM.
        If so we return a mask indicating how to build the substitutions for the induction from TERM and the machine for fn.
        Otherwise we return NIL. Let the changeables be those actuals of TERM that are in the measured subset of
        JUSTIFICATION and that sometimes change in the recursion. Let the unchangeables be all of the variables occurring in
        measured actuals that never change in recursion. The induction applies if changeables is a sequence of distinct variable
        names and has an empty intersection with unchangeables. If the induction is applicable then the substitutions should
        substitute for the changeables just as the recursion would, and hold each unchangeable fixed --
        i.e., substitute each for itself. With such substitutions it is possible to prove the measure lemmas analogous to those proved
        in JUSTIFICATION, except that the measure is obtained by instantiating the measure term used in the justification by the
        measured actuals in unchanging slots. Actual variables that are neither among the changeables or unchangeables may be
        substituted for arbitrarily. If the induction is applicable we return a mask with as many elements as there are actuals.
        For each actual the mask contains either CHANGEABLE, UNCHANGEABLE, or NIL.
```

CHANGEABLE means the actual should be instantiated as specified in the recursion.
UNCHANGEABLE means each var in the actual should be held fixed.
NIL means that the corresponding substitution pairs in the machine for the function should be ignored.
Abstractly, this function builds the mask by first putting either CHANGEABLE or UNCHANGEABLE in each measured slot.
It then fills in the remaining slots from the left so as to permit the actual to be instantiated or held fixed as desired by the recursion, provided that in so doing it does not permit substitutions for previously allocated actuals.
*)

```
(PROG (UNCHANGEABLES SUBSET CHANGEABLES)
      (SETQ SUBSET (fetch (JUSTIFICATION SUBSET) of JUSTIFICATION))
      (SETQ UNCHANGEABLES (for ACTUAL in (FARGS TERM) as VAR in FORMALS as Q in QUICK-BLOCK-INFO bind LOOP-ANS
                             when (AND (MEMB VAR SUBSET)
                                       (EQ Q (QUOTE UNCHANGING)))
                             do (SETQ LOOP-ANS (UNIONQ (ALL-VARS ACTUAL)
                                                       LOOP-ANS))
                             finally (RETURN LOOP-ANS)))
      (SETQ CHANGEABLES (for ACTUAL in (FARGS TERM) as VAR in FORMALS as Q in QUICK-BLOCK-INFO
                           when (AND (MEMB VAR SUBSET)
                                     (NEQ Q (QUOTE UNCHANGING)))
                           collect ACTUAL))
      (COND
         ((OR (NOT (NO-DUPLICATESP CHANGEABLES))
              (for X in CHANGEABLES thereis (NVARIABLEP X))
              (INTERSECTP CHANGEABLES UNCHANGEABLES))
          (RETURN NIL)))
      (RETURN (for ACTUAL in (FARGS TERM) as Q in QUICK-BLOCK-INFO as VAR in FORMALS
                 collect (COND
                           [(MEMB VAR SUBSET)
                            (COND
                               ((EQ Q (QUOTE UNCHANGING))
                                (QUOTE UNCHANGEABLE))
                               (T (QUOTE CHANGEABLE]
                           [(AND (VARIABLEP ACTUAL)
                                 (EQ Q (QUOTE UNCHANGING)))
                            (COND
                               ((MEMB ACTUAL CHANGEABLES)
                                NIL)
                               (T (SETQ UNCHANGEABLES (ADD-TO-SET ACTUAL UNCHANGEABLES))
                                  (QUOTE UNCHANGEABLE]
                           ((AND (VARIABLEP ACTUAL)
                                 (NOT (MEMB ACTUAL CHANGEABLES))
                                 (NOT (MEMB ACTUAL UNCHANGEABLES)))
                            (SETQ CHANGEABLES (CONS ACTUAL CHANGEABLES))
                            (QUOTE CHANGEABLE))
                           (T NIL])
```

(STACK-DEPTH

```
[LAMBDA (STK)                                                    (* kbr: "19-Oct-85 21:59")
  (ADD1 (LENGTH STK])
```

(START-STATS

```
[LAMBDA NIL                                                      (* kbr: "19-Oct-85 16:31")
  (SETQ ELAPSEDTHMTIME (TIME-IN-60THS))
  (SETQ IOTHMTIME 0])
```

(STOP-STATS

```
[LAMBDA NIL                                                      (* kbr: "19-Oct-85 16:31")
  (PRINT-STATS (QUOTIENT (FLOAT (DIFFERENCE (DIFFERENCE (TIME-IN-60THS)
                                                        ELAPSEDTHMTIME)
                                            IOTHMTIME))
                         60.0)
               (QUOTIENT (FLOAT IOTHMTIME)
                         60.0)
               PROVE-FILE])
```

(STORE-SENT

```
[LAMBDA (CL HIST)                                                (* kbr: "20-Oct-85 15:36")
  (LET (CL-SET)
       (COND
          ((NULL CL)
           (IO (QUOTE STORE-SENT)
               CL HIST NIL (LIST (GET-STACK-NAME STACK)))
           (WRAPUP NIL))
          (DO-NOT-USE-INDUCTION-FLG (IO (QUOTE STORE-SENT)
                                        CL HIST NIL (LIST (GET-STACK-NAME STACK)
                                                          (QUOTE QUIT)))
                 (WRAPUP NIL))
          ([AND (NOT (AND IN-PROVE-LEMMA-FLG (ASSOC (QUOTE INDUCT)
                                                    HINTS)))
                (OR [AND (NULL STACK)
                         (for X in HIST thereis (NOT (MEMB (CAR X)
                                                           (QUOTE (SETTLED-DOWN-CLAUSE SIMPLIFY-CLAUSE SETUP]
                    (AND STACK (NOT (ASSOC (QUOTE BEING-PROVED)
```

```
                                        STACK]
```

(* Abort and push the input clause to work on if (a) this is the first time we've ever pushed anything and we've done anything to the input other than simplify it, or (b) we have not yet gone into the first induction for the original conjecture but have already pushed one simplified subgoal. *)

```
   (SETQ STACK NIL)
   (SETQ CL-SET (CNF-DNF THM (QUOTE C)))
```

(* Once upon a time we backed up to the output of PREPROCESS in PROVE.
However, PREPROCESS -- and CLAUSIFY-INPUT -- applies unconditional rewrite rules and we want the ability as users to type in exactly what the system inducts on. The theorem that PREPROCESS screwed us on was HACK1 when it distributed TIMES and GCD. *)

```
   (IO (QUOTE STORE-SENT)
       CL NIL NIL (LIST (GET-STACK-NAME STACK)
                        CL-SET))
   (PUSH-CLAUSE-SET CL-SET)
   (RETFROM (QUOTE SIMPLIFY-LOOP)
          NIL))
  (T (SETQ CL-SET (LIST CL))
     (IO (QUOTE STORE-SENT)
         CL HIST NIL (LIST (GET-STACK-NAME STACK)))
     (PUSH-CLAUSE-SET CL-SET])
```

(STRIP-BRANCHES

```
[LAMBDA (TERM)                                          (* kbr: "19-Oct-85 16:31")
  (LET (CL)
       (for PAIR in (COND
                      ((BM-MATCH TERM (NOT TERM))
                       (STRIP-BRANCHES1 TERM T T))
                      (T (STRIP-BRANCHES1 TERM T NIL)))
         unless (EQUAL (SETQ CL (ADD-LITERAL (PEGATE-LIT (CAR PAIR))
                                             (CDR PAIR)
                                             T))
                       TRUE-CLAUSE)
         collect CL])
```

(STRIP-BRANCHES1

```
[LAMBDA (TERM TOPFLG NEGATE-FLG)                        (* kbr: "19-Oct-85 16:31")
  (LET (ANS1 ANS2 ANS3 ANS LST NEW-CL)
       (COND
          ((VARIABLEP TERM)
           (LIST (CONS (COND
                         (NEGATE-FLG (NEGATE-LIT TERM))
                         (T TERM))
                       NIL)))
          [(FQUOTEP TERM)
           (COND
              (TOPFLG (COND
                        [(EQUAL TERM FALSE)
                         (COND
                            (NEGATE-FLG NIL)
                            (T (LIST (CONS FALSE NIL]
                        (NEGATE-FLG (LIST (CONS FALSE NIL)))
                        (T NIL)))
              (NEGATE-FLG (LIST (CONS (COND
                                        ((EQUAL TERM FALSE)
                                         TRUE)
                                        (T FALSE))
                                      NIL)))
              (T (LIST (CONS TERM NIL]
          ((EQ (FFN-SYMB TERM)
               (QUOTE IF))
           (COND
              ([AND TOPFLG (OR (AND (NOT NEGATE-FLG)
                                    (EQUAL (FARGN TERM 3)
                                           FALSE))
                               (AND NEGATE-FLG (EQUAL (FARGN TERM 3)
                                                      TRUE]
               (APPEND (for PAIR in (STRIP-BRANCHES1 (FARGN TERM 1)
                                                     TOPFLG NIL)
                          unless (EQUAL (SETQ NEW-CL (ADD-LITERAL (PEGATE-LIT (CAR PAIR))
                                                                  (CDR PAIR)
                                                                  T))
                                        TRUE-CLAUSE)
                          collect (CONS FALSE NEW-CL))
                       (STRIP-BRANCHES1 (FARGN TERM 2)
                                        TOPFLG NEGATE-FLG)))
              ([AND TOPFLG (OR (AND (NOT NEGATE-FLG)
                                    (EQUAL (FARGN TERM 2)
                                           FALSE))
                               (AND NEGATE-FLG (EQUAL (FARGN TERM 2)
                                                      TRUE]
               (APPEND (for PAIR in (STRIP-BRANCHES1 (FARGN TERM 1)
```

```
                                    TOPFLG T)
                          unless (EQUAL (SETQ NEW-CL (ADD-LITERAL (PEGATE-LIT (CAR PAIR))
                                                                  (CDR PAIR)
                                                                  T))
                                        TRUE-CLAUSE)
                                 collect (CONS FALSE NEW-CL))
                                (STRIP-BRANCHES1 (FARGN TERM 3)
                                   TOPFLG NEGATE-FLG)))
                       (T (SETQ ANS1 (STRIP-BRANCHES1 (FARGN TERM 1)
                                        NIL NIL))
                          (SETQ ANS2 (STRIP-BRANCHES1 (FARGN TERM 2)
                                        TOPFLG NEGATE-FLG))
                          (SETQ ANS3 (STRIP-BRANCHES1 (FARGN TERM 3)
                                        TOPFLG NEGATE-FLG))
                          [for PAIR in ANS1
                             do (for PAIR2 in ANS2 unless (EQUAL [CDR (SETQ ANS (CONS (CAR PAIR2)
                                                                                      (DISJOIN-CLAUSES
                                                                                       (CDR PAIR)
                                                                                       (ADD-LITERAL (NEGATE-LIT
                                                                                                     (CAR PAIR))
                                                                                              (CDR PAIR2)
                                                                                              NIL]
                                                                                      TRUE-CLAUSE)
                                                 do (SETQ LST (CONS ANS LST)))
                                        (for PAIR3 in ANS3 unless (EQUAL [CDR (SETQ ANS (CONS (CAR PAIR3)
                                                                                              (DISJOIN-CLAUSES
                                                                                               (CDR PAIR)
                                                                                               (ADD-LITERAL (PEGATE-LIT
                                                                                                             (CAR PAIR))
                                                                                                      (CDR PAIR3)
                                                                                                      NIL]
                                                                                              TRUE-CLAUSE)
                                                 do (SETQ LST (CONS ANS LST]
                             LST)))
                       (T (for PICK in (ALL-PICKS (for ARG in (FARGS TERM) collect (STRIP-BRANCHES1 ARG NIL NIL)))
                             collect (CONS [COND
                                              [NEGATE-FLG (DUMB-NEGATE-LIT (SCONS-TERM (FFN-SYMB TERM)
                                                                                       (for PAIR in PICK collect (CAR PAIR]
                                              (T (SCONS-TERM (FFN-SYMB TERM)
                                                             (for PAIR in PICK collect (CAR PAIR]
                                           (for PAIR in PICK bind ANS until (EQUAL ANS TRUE-CLAUSE)
                                              do (SETQ ANS (DISJOIN-CLAUSES (CDR PAIR)
                                                                            ANS))
                                              finally (RETURN ANS])
```

(SUB-SEQUENCEP

```
[LAMBDA (SMALLER LARGER)                                    (* kbr: "19-Oct-85 16:31")
  (COND
     ((NLISTP SMALLER)
      T)
     ((NLISTP LARGER)
      NIL)
     ((EQUAL (CAR SMALLER)
             (CAR LARGER))
      (SUB-SEQUENCEP (CDR SMALLER)
             (CDR LARGER)))
     (T (SUB-SEQUENCEP SMALLER (CDR LARGER])
```

(SUBBAGP

```
[LAMBDA (BAG1 BAG2)                                         (* kbr: "19-Oct-85 16:31")
  (COND
     ((NLISTP BAG1)
      T)
     ((NLISTP BAG2)
      NIL)
     ((MEMBER (CAR BAG1)
              BAG2)
      (SUBBAGP (CDR BAG1)
             (DELETE1 (CAR BAG1)
                    BAG2)))
     (T NIL])
```

(SUBLIS-EXPR

```
[LAMBDA (ALIST FORM)                                        (* kbr: "19-Oct-85 16:31")
  [for PAIR in ALIST do (COND
                          ((QUOTEP (CAR PAIR))
                           (SUBST-EXPR-ERROR1 (CAR PAIR]
  (SUBLIS-EXPR1 ALIST FORM])
```

(SUBLIS-EXPR1

```
[LAMBDA (ALIST FORM)                                        (* kbr: "19-Oct-85 16:31")
  (COND
     ((SETQ TEMP-TEMP (SASSOC FORM ALIST))
```

```
        (CDR TEMP-TEMP))
       ((VARIABLEP FORM)
        FORM)
       ((FQUOTEP FORM)
        FORM)
       (T (CONS-TERM (FFN-SYMB FORM)
                 (for ARG in (FARGS FORM) collect (SUBLIS-EXPR1 ALIST ARG])
```

(SUBLIS-VAR

```
[LAMBDA (ALIST FORM)                                                    (* kbr: "19-Oct-85 16:31")
```

(* In REWRITE-WITH-LEMMAS we use this function with the NIL alist to put FORM into quote normal form. Do not optimize this function for the NIL alist. *)

```
  (COND
     ((VARIABLEP FORM)
      (COND
         ((SETQ TEMP-TEMP (ASSOC FORM ALIST))
          (CDR TEMP-TEMP))
         (T FORM)))
     ((FQUOTEP FORM)
      FORM)
     (T (CONS-TERM (FFN-SYMB FORM)
                 (for ARG in (FARGS FORM) collect (SUBLIS-VAR ALIST ARG])
```

(SUBLIS-VAR-LST

```
[LAMBDA (ALIST TERMLST)                                                 (* kbr: "19-Oct-85 16:31")
  (for TERM in TERMLST collect (SUBLIS-VAR ALIST TERM])
```

(SUB-PAIR-EXPR

```
[LAMBDA (OLDLST NEWLST TERM)                                            (* kbr: "19-Oct-85 16:31")
  [for X in OLDLST do (COND
                        ((QUOTEP X)
                         (SUBST-EXPR-ERROR1 X]
  (SUB-PAIR-EXPR1 OLDLST NEWLST TERM])
```

(SUB-PAIR-EXPR-LST

```
[LAMBDA (OLDLST NEWLST LST)                                             (* kbr: "19-Oct-85 16:31")
  (for X in LST collect (SUB-PAIR-EXPR OLDLST NEWLST X])
```

(SUB-PAIR-EXPR1

```
[LAMBDA (OLDLST NEWLST TERM)                                            (* kbr: "19-Oct-85 16:31")
  (COND
     ((for OLD1 in OLDLST as NEW1 in NEWLST thereis (COND
                                                      ((EQUAL OLD1 TERM)
                                                       (SETQ TEMP-TEMP NEW1)
                                                       T)
                                                      (T NIL)))
      TEMP-TEMP)
     ((VARIABLEP TERM)
      TERM)
     ((FQUOTEP TERM)
      TERM)
     (T (CONS-TERM (FFN-SYMB TERM)
                 (for ARG in (FARGS TERM) collect (SUB-PAIR-EXPR1 OLDLST NEWLST ARG])
```

(SUB-PAIR-VAR

```
[LAMBDA (OLDLST NEWLST TERM)                                            (* kbr: "19-Oct-85 16:31")
  (COND
     ((VARIABLEP TERM)
      (COND
         ((for OLD1 in OLDLST as NEW1 in NEWLST thereis (COND
                                                          ((EQ OLD1 TERM)
                                                           (SETQ TEMP-TEMP NEW1)
                                                           T)
                                                          (T NIL)))
          TEMP-TEMP)
         (T TERM)))
     ((FQUOTEP TERM)
      TERM)
     (T (CONS-TERM (FFN-SYMB TERM)
                 (for ARG in (FARGS TERM) collect (SUB-PAIR-VAR OLDLST NEWLST ARG])
```

(SUB-PAIR-VAR-LST

```
[LAMBDA (OLDLST NEWLST LST)                                             (* kbr: "19-Oct-85 16:31")
  (for X in LST collect (SUB-PAIR-VAR OLDLST NEWLST X])
```

(SUBST-EXPR

```
[LAMBDA (NEW OLD FORM)                                                  (* kbr: "19-Oct-85 16:31")
```

```
    (COND
       ((VARIABLEP OLD)
        (SUBST-VAR NEW OLD FORM))
       ((FQUOTEP OLD)
        (SUBST-EXPR-ERROR1 OLD))
       (T (SUBST-EXPR1 NEW OLD FORM])
```

(SUBST-EXPR-ERROR1

```
  [LAMBDA (OLD)                                                    (* kbr: "19-Oct-85 16:31")
    (ERROR1 (PQUOTE (PROGN ATTEMPT TO BM-SUBST FOR THE EXPLICIT CONSTANT (!PPR OLD NIL)
                           %. THE SUBSTITUTION FUNCTIONS WERE OPTIMIZED TO DISALLOW THIS %.))
            (BINDINGS (QUOTE OLD)
                  OLD)
            (QUOTE HARD])
```

(SUBST-EXPR-LST

```
  [LAMBDA (NEW OLD LST)                                            (* kbr: "19-Oct-85 16:31")
    (for X in LST collect (SUBST-EXPR NEW OLD X])
```

(SUBST-EXPR1

```
  [LAMBDA (NEW OLD FORM)                                           (* kbr: "19-Oct-85 16:31")
    (COND
       ((EQUAL OLD FORM)
        NEW)
       ((VARIABLEP FORM)
        FORM)
       ((FQUOTEP FORM)
        FORM)
       (T (CONS-TERM (FFN-SYMB FORM)
                 (for ARG in (FARGS FORM) collect (SUBST-EXPR1 NEW OLD ARG])
```

(SUBST-FN

```
  [LAMBDA (NEW OLD TERM)                                           (* kbr: "19-Oct-85 16:31")
```

(* Replaces calls of OLD with calls of NEW. Assumes both have same arity and that neither is a shell constructor or bottom object. *)

```
    (COND
       ((VARIABLEP TERM)
        TERM)
       ((FQUOTEP TERM)
        TERM)
       [(EQ OLD (FFN-SYMB TERM))
        (FCONS-TERM NEW (for ARG in (FARGS TERM) collect (SUBST-FN NEW OLD ARG]
       (T (FCONS-TERM (FFN-SYMB TERM)
                 (for ARG in (FARGS TERM) collect (SUBST-FN NEW OLD ARG])
```

(SUBST-VAR

```
  [LAMBDA (NEW OLD FORM)                                           (* kbr: "19-Oct-85 16:31")
    (COND
       ((VARIABLEP FORM)
        (COND
           ((EQ FORM OLD)
            NEW)
           (T FORM)))
       ((FQUOTEP FORM)
        FORM)
       (T (CONS-TERM (FFN-SYMB FORM)
                 (for ARG in (FARGS FORM) collect (SUBST-VAR NEW OLD ARG])
```

(SUBST-VAR-LST

```
  [LAMBDA (NEW OLD TERMLST)                                        (* kbr: "19-Oct-85 16:31")
    (for TERM in TERMLST collect (SUBST-VAR NEW OLD TERM])
```

(BM-SUBST

```
  [LAMBDA (NEW OLD FORM)                                           (* kbr: "19-Oct-85 16:31")
    (COND
       ((VARIABLEP OLD)
        (SUBST-VAR NEW OLD FORM))
       (T (SUBST-EXPR NEW OLD FORM])
```

(SUBSUMES

```
  [LAMBDA (CL1 CL2)                                                (* kbr: "19-Oct-85 16:31")
    (LET (UNIFY-SUBST)
         (SUBSUMES1 CL1])
```

(SUBSUMES-REWRITE-RULE

```
  [LAMBDA (REWRITE-RULE1 REWRITE-RULE2)                            (* kbr: "19-Oct-85 16:31")
```

```
    (LET (UNIFY-SUBST (CL2 (fetch (REWRITE-RULE HYPS) of REWRITE-RULE2)))
         (AND (ONE-WAY-UNIFY1 (fetch (REWRITE-RULE CONCL) of REWRITE-RULE1)
                              (fetch (REWRITE-RULE CONCL) of REWRITE-RULE2))
              (SUBSUMES1 (fetch (REWRITE-RULE HYPS) of REWRITE-RULE1])
```

(SUBSUMES1

```
  [LAMBDA (CL1)                                                    (* kbr: "19-Oct-85 16:31")
                                                                   (* Also called by SUBSUMES-SEQ.
                                                                   *)
    (COND
       ((NULL CL1)
        T)
       (T (for LIT in CL2 thereis (SUBSUMES11 LIT CL1 UNIFY-SUBST])
```

(SUBSUMES11

```
  [LAMBDA (LIT CL1 UNIFY-SUBST)                                    (* kbr: "19-Oct-85 16:31")
    (AND (ONE-WAY-UNIFY1 (CAR CL1)
                         LIT)
         (SUBSUMES1 (CDR CL1])
```

(SUM-STATS-ALIST

```
  [LAMBDA (ALIST)                                                  (* kbr: "25-Oct-85 16:21")
    (PROG (CPU IO)
          (SETQ CPU 0)
          (SETQ IO 0)
          (for X in ALIST do (SETQ CPU (IPLUS (CADR X)
                                              CPU))
                             (SETQ IO (IPLUS (CADDR X)
                                             IO)))
          (RETURN (LIST CPU IO])
```

(TABULATE

```
  [LAMBDA (N FILE)                                                 (* kbr: "19-Oct-85 16:31")
    (ISPACES (IDIFFERENCE N (IPOSITION FILE NIL NIL))
             FILE])
```

(TERM-ORDER

```
  [LAMBDA (TERM1 TERM2)                                            (* kbr: "26-Oct-85 17:20")

          (* A simple -- or complete or total -- ordering is a relation satisfying: XrX.
          TERM-ORDER is a simple ordering on terms. (TERM-ORDER TERM1 TERM2) if and only if
          (a) the number of occurrences of variables in TERM1 is strictly less than the number in TERM2, or
          (b) the numbers of variable occurrences are equal and the FORM-COUNT of TERM1 is strictly less than that of TERM2, or
          (c) the numbers of variable occurrences are equal, the FORM-COUNTS are equal, and
          (LEXORDER TERM1 TERM2)%. Let (STRICT-TERM-ORDER X Y) be the LISP function defined as
          (AND (TERM-ORDER X Y) (NOT (EQUAL X Y)))%. For a fixed, finite set of function symbols and variable symbols
          STRICT-TERM-ORDER is well founded, as can be proved with the following lemma.
          Lemma. Suppose that M is a function whose range is well ordered by r and such that the inverse image of any member of
          the range is finite. Suppose that L is a total order. Define (LESSP ITIMES y) IEQP
          (OR (r (M ITIMES) (M y)) (AND (EQUAL (M ITIMES) (M y)) (L ITIMES y)
          (NOT (EQUAL ITIMES y))))%. ILESSP is a well-ordering. Proof.
          Suppose ... ILESSP t3 ILESSP t2 ILESSP t1 is an infinite descending sequence.
          ..., (M t3), (M t2), (M t1) is weakly descending but not infinitely descending and so has a least element.
          WLOG assume ... IEQP (M t3) IEQP (M t2) IEQP (M t1)%. By the finiteness of the inverse image of
          (M t1), { ..., t3, t2, t1} is a finite set, which has a least element under L, WLOG t27.
          But t28 L t27 and t28 /= t27 by t28 ILESSP t27, contradicting the minimality of t27.
          QED If (TERM-ORDER ITIMES y) and t2 results from replacing one occurrence of y with ITIMES in t1, then
          (TERM-ORDER t2 t1)%. Cases on why ITIMES is less than y. 1.0 If the number of occurrences of variables in ITIMES is
          strictly smaller than in y, then the number in t2 is strictly smaller than in t1.
          2.0 If the number of occurrences of variables in ITIMES is equal to the number in y but
          (FORM-COUNT ITIMES) is smaller than (FORM-COUNT y), then the number of occurrences in t1 is equal to the number in
          t2 but (FORM-COUNT t1) is less than (FORM-COUNT t2)%. 3.0 If the number of variable occurrences and parenthesis
          occurrences in ITIMES and y are the same, then (LEXORDER ITIMES y)%.
          (TERM-ORDER t2 t1) reduces to (LEXORDER t2 t1) because the number of variable and parenthesis occurrences in t2 and
          t1 are the same. The lexicographic scan of t1 and t2 will be all equals until ITIMES and y are hit.
          *)

    (LET (FORM-COUNT1 FORM-COUNT2 NUMBER-OF-VARIABLES1 NUMBER-OF-VARIABLES2)
                                                                   (* Side effect of FORM-COUNT is to set
                                                                   NUMBER-OF-VARIABLES. *)
         (SETQ FORM-COUNT1 (FORM-COUNT TERM1))
         (SETQ NUMBER-OF-VARIABLES1 NUMBER-OF-VARIABLES)
         (SETQ FORM-COUNT2 (FORM-COUNT TERM2))
         (SETQ NUMBER-OF-VARIABLES2 NUMBER-OF-VARIABLES)
         (COND
            ((LESSP NUMBER-OF-VARIABLES1 NUMBER-OF-VARIABLES2)
             T)
            ((LESSP NUMBER-OF-VARIABLES2 NUMBER-OF-VARIABLES1)
             NIL)
            ((LESSP FORM-COUNT1 FORM-COUNT2)
             T)
            ((LESSP FORM-COUNT2 FORM-COUNT1)
```

```
            NIL)
           (T (LEXORDER TERM1 TERM2])
```

(TERMINATION-MACHINE

```
[LAMBDA (FNNAME TERM TESTS)                                    (* kbr: "19-Oct-85 16:31")

        (* This function builds a list of TESTS-AND-CASE representing the function FNNAME with body TERM.
        For each call of FNNAME in body, a TESTS-AND-CASE is returned whose TESTS are all the tests that CASE is the arglist
        of the call. If a rules b, then a governs b but not vice versa. For example, in
        (if (g (if a b c)) d e), a governs b but does not rule b. The reason for taking this weaker notion of governance is that we can
        show easily that the TESTS-AND-CASEs are together sufficient to imply the TESTS-AND-CASES generated by
        INDUCTION-MACHINE. *)

   (COND
      ((OR (VARIABLEP TERM)
           (FQUOTEP TERM))
       NIL)
      [(EQ (FFN-SYMB TERM)
           (QUOTE IF))
       (NCONC (for ARGLIST in (ALL-ARGLISTS FNNAME (FARGN TERM 1))
                 collect (create TESTS-AND-CASE
                                 TESTS _ TESTS
                                 CASE _ ARGLIST))
              [TERMINATION-MACHINE FNNAME (FARGN TERM 2)
                     (APPEND TESTS (LIST (FARGN TERM 1]
              (TERMINATION-MACHINE FNNAME (FARGN TERM 3)
                     (APPEND TESTS (LIST (NEGATE-LIT (FARGN TERM 1]
      (T (for ARGLIST in (ALL-ARGLISTS FNNAME TERM) collect (create TESTS-AND-CASE
                                                                    TESTS _ TESTS
                                                                    CASE _ ARGLIST])
```

(TP-EXPLODEN1

```
[LAMBDA (SYM)                                                  (* kbr: "19-Oct-85 16:31")
   (for N in (OUR-EXPLODEN SYM)
      collect (COND
                ((OR (IEQP N #/-)
                     (AND (ILEQ #/0 N)
                          (ILEQ N #/9)))
                 N)
                ((AND (ILEQ #/A N)
                      (ILEQ N #/Z))
                 (IDIFFERENCE N 32.0))
                (T (ERROR1 (PQUOTE (PROGN QUOTED LITERAL ATOMS MUST BE IN LOWER CASE AND (!PPR X NIL)
                                                                                          IS NOT %.))
                           (BINDINGS (QUOTE X)
                                  SYM)
                           (QUOTE SOFT])
```

(TP-GETCHARN1

```
[LAMBDA (SYM N)                                                (* kbr: "19-Oct-85 16:31")
   (LET ((A (OUR-GETCHARN SYM N)))
        (COND
           ((OR (IEQP A #/-)
                (AND (ILEQ #/0 A)
                     (ILEQ A #/9)))
            A)
           ((AND (ILEQ #/A A)
                 (ILEQ A #/Z))
            (IDIFFERENCE A 32.0))
           (T (ERROR1 (PQUOTE (PROGN QUOTED LITERAL ATOMS MUST BE IN LOWER CASE AND (!PPR X NIL)
                                                                                    IS NOT %.))
                      (BINDINGS (QUOTE X)
                             SYM)
                      (QUOTE HARD])
```

(TP-IMPLODE1

```
[LAMBDA (L)                                                    (* kbr: "19-Oct-85 16:31")
   (OUR-IMPLODE (for N in L
                   collect (COND
                             ((OR (IEQP (OUR-GETCHARN N 1)
                                        #/-)
                                  (AND (ILEQ #/0 (OUR-GETCHARN N 1))
                                       (ILEQ (OUR-GETCHARN N 1)
                                             #/9)))
                              (OUR-GETCHARN N 1))
                             ((AND (ILEQ #/A (OUR-GETCHARN N 1))
                                   (ILEQ (OUR-GETCHARN N 1)
                                         #/Z))
                              (IPLUS (OUR-GETCHARN N 1)
                                     32.0))
                             (T (ERROR1 (PQUOTE (PROGN QUOTED LITERAL ATOMS MUST BE IN LOWER CASE
                                                       AND (!PPR X NIL)
                                                       IS NOT %.))
```

```
                                        (BINDINGS (QUOTE X)
                                                  (OUR-IMPLODE L))
                                        (QUOTE HARD])
```

(TO-BE-IGNOREDP

```
[LAMBDA (POLY)                                                  (* kbr: "19-Oct-85 16:31")
  (LET (LEMMAS LITS)
       (SETQ LEMMAS (fetch (POLY LEMMAS) of POLY))
       (SETQ LITS (fetch (POLY LITERALS) of POLY))
       (for LIT in LITS-TO-BE-IGNORED-BY-LINEAR thereis (OR (MEMB LIT LEMMAS)
                                                            (MEMB LIT LITS])
```

(TOO-MANY-IFS

```
[LAMBDA (ARGS VAL)                                              (* kbr: "20-Oct-85 19:31")
```

(* Let ARGS be the list of actuals to a nonrec fn. Let VAL be the rewritten body.
We wish to determine whether the expansion of the fn call introduces too many IFs all at once.
Our motivation comes from an example like (M2 (ZTAK & & &) (ZTAK & & &)
(ZTAK & & &)) where the careless opening up of everybody produces a formula with several hundred IFs in it because of M2's duplication of the IFs coming from the simplification of the ZTAKs.
My first thought was to never expand a nonrec fn -- at the top level of the clause --
if it had some IFs in its args and to wait till CLAUSIFY has cleaned things up.
That slowed a proveall down by a factor of 2 -- and by a factor of 13 in PRIME-LIST-TIMES-LIST --
because of the ridiculously slow expansion of such basic nonrec fns as AND, OR, NOT, and NLISTP.
I have been thinking about the problem and have thought of the following ideas.
None except the final one have been implemented or tested. I thought of permitting the expansion if VAL had fewer IFs than ARGS but that is obviously bad because it does not permit the fn to introduce any IFs of its own, e.g., as in AND.
So I have decided to just prohibit the duplication of IF-containing-args in VAL.
That is, I do not want to expand the fn if the expansion causes the duplication of some arg containing an IF.
Of course, it could be that an IF-containing-arg does not occur in VAL only because it has been rewritten by some rewrite rule to some other term, possibly containing even more IFs, but I have decided to ignore that and blame that problem on the process that permitted the introduction of those IFs. So when I say an arg is duplicated in VAL I really mean the arg literally OCCURs twice. Then it occurred to me that if arg1 and arg2 both contained IFs and arg1 was duplicated in VAL but arg2 did not occur at all, then perhaps one should permit the expansion if the number of IFs in the arg1 occurrences are less than the number in the arg1 plus arg2. So that is what I have implemented.
This function computes (GREATERP (FOR ARG IN ARGS SUM
(* (COUNT-IFS ARG) *)) (FOR ARG IN ARGS SUM (COUNT-IFS ARG))) but does it slightly more efficiently by observing that if no IFs occur in any arg then there is no point in doing the OCCUR-CNTs and that once the left hand side has been pushed beyond the right there is no point in continuing. *)

```
  (LET (RHS LHS)
       (SETQ RHS (for ARG in ARGS sum (COUNT-IFS ARG)))
       (SETQ LHS 0)
       (COND
          ((ZEROP RHS)
           NIL)
          (T (for ARG in ARGS when [NOT (ZEROP (SETQ TEMP-TEMP (COUNT-IFS ARG]
                thereis (PROGN
```

(* The WHEN clause above just takes advantage of the fact that if X is 0 then X*Y is 0 and Y need not be computed.
*)

```
                                (GREATERP (SETQ LHS (PLUS (TIMES TEMP-TEMP (OCCUR-CNT ARG VAL))
                                                          LHS))
                                          RHS])
```

(TOP-FNNAME

```
[LAMBDA (CONCL)                                                 (* kbr: "19-Oct-85 16:31")
  (OR (BM-MATCH CONCL (NOT CONCL))
      (BM-MATCH CONCL (EQUAL CONCL &)))
  (COND
     ((VARIABLEP CONCL)
      NIL)
     (T (FN-SYMB CONCL])
```

(TOTAL-FUNCTIONP

```
[LAMBDA (FNNAME)                                                (* kbr: "19-Oct-85 16:31")
  (LET (TEMP)
       (SETQ TEMP (GET1 FNNAME (QUOTE JUSTIFICATIONS)))
       (NOT (AND (IEQP (LENGTH TEMP)
                       1)
                 (NULL (fetch (JUSTIFICATION RELATION) of (CAR TEMP)))
                 (NOT (DISABLEDP FNNAME])
```

(TRANSITIVE-CLOSURE

```
[LAMBDA (SET PRED)                                              (* kbr: "26-Oct-85 14:02")
```

(* Compares all pairs x,y of distinct occurrences of from the bag SET with
(PRED ITIMES y) and if PRED returns non-NIL, ITIMES and y are removed from SET and the result of PRED is inserted.
This operation is repeated until no changes occur. CAUTION: It must be the case that
(PRED ITIMES y) IEQP (PRED y ITIMES) . *)

```
    (LET (ALIVE NEW RESULT)
         (SETQ ALIVE (for X in SET collect (CONS X T)))
         (SETQ NEW (COPYLIST ALIVE))
         (while NEW unless (AND (CDR (CAR NEW))
                                (for TAIL on ALIVE when [PROG NIL
                                                            LOOP
                                                                [COND
                                                                   ((NULL (CDR (CAR TAIL)))
                                                                    (COND
                                                                       ((NULL (CDR TAIL))
                                                                        (RETURN NIL))
                                                                       (T (RPLACA TAIL (CADR TAIL))
                                                                          (RPLACD TAIL (CDDR TAIL))
                                                                          (GO LOOP]
                                                                (RETURN (COND
                                                                           ((EQ (CAR TAIL)
                                                                                (CAR NEW))
                                                                            NIL)
                                                                           ([SETQ RESULT (APPLY* PRED
                                                                                                 (CAR (CAR TAIL))
                                                                                                 (CAR (CAR NEW]
                                                                            (SETQ RESULT (CONS RESULT T))
                                                                            (RPLACD (CAR TAIL)
                                                                                   NIL)
                                                                            (RPLACA TAIL RESULT)
                                                                            (RPLACD (CAR NEW)
                                                                                    NIL)
                                                                            (RPLACA NEW RESULT)
                                                                            T)
                                                                           (T NIL]
                                do (RETURN TAIL)))
           do (SETQ NEW (CDR NEW)))
         (for PAIR in ALIVE when (CDR PAIR) collect (CAR PAIR])
```

(TRANSLATE

```
  [LAMBDA (X)                                                    (* kbr: "26-Oct-85 17:19")
    (COND
       [(NLISTP X)
        (COND
           ((FIXP X)
            (LIST (QUOTE QUOTE)
                  X))
           ((LITATOM X)
            (COND
               ((EQ X T)
                TRUE)
               ((EQ X (QUOTE F))
                FALSE)
               ((EQ X NIL)
                (QUOTE (QUOTE NIL)))
               ((ILLEGAL-NAME X)
                (ERROR1 (PQUOTE (PROGN (!PPR X NIL)
                                       IS AN ILLEGAL VARIABLE NAME %.))
                        (BINDINGS (QUOTE X)
                                  X)
                        (QUOTE SOFT)))
               (T X)))
           (T (ERROR1 (PQUOTE (PROGN UNRECOGNIZED SYNTAX: (!PPR X NIL)))
                      (BINDINGS (QUOTE X)
                                X)
                      (QUOTE SOFT]
       ((NOT (LISTP X))
        (ERROR1 (PQUOTE (PROGN NO HUNKS PLEASE: (!PPR X NIL)))
                (BINDINGS (QUOTE X)
                          X)
                (QUOTE SOFT)))
       ((CDR (LAST X))
        (ERROR1 (PQUOTE (PROGN CONTRARY TO THE RULES OF WELL-FORMEDNESS , THE LAST CDR OF (!PPR X NIL)
                               IS NON-NIL))
                (BINDINGS (QUOTE X)
                          X)
                (QUOTE SOFT)))
       ((NOT (LITATOM (CAR X)))
        (ERROR1 (PQUOTE (PROGN FUNCTION SYMBOLS MUST BE LISP LITERAL ATOMS AND (!PPR (CAR X)
                                                                                     NIL)
                                                                          IS NOT
                               !))
                (BINDINGS (QUOTE X)
                          X)
                (QUOTE SOFT)))
       ((NOT (LITATOM (CAR X)))
        (ERROR1 (PQUOTE (PROGN (!PPR (CAR X)
                                     NIL)
                               IS NOT INTERNED IN THE RIGHT PLACES !))
                (BINDINGS (QUOTE X)
                          X)
```

```
        (QUOTE SOFT)))
      [(PROPERTYLESS-SYMBOLP (CAR X))
       (COND
          ((EQ (CAR X)
               (QUOTE QUOTE))
           (COND
              ([NOT (IEQP 1 (LENGTH (CDR X]
               (ERROR1 (PQUOTE (PROGN QUOTE MUST BE GIVEN EXACTLY ONE ARGUMENT %. IN (!PPR X NIL)
                                      IT IS GIVEN THE WRONG NUMBER OF ARGUMENTS %.))
                       (BINDINGS (QUOTE X)
                              X)
                       (QUOTE SOFT)))
              ((NOT (EVG (CADR X)))
               (ERROR1 (PQUOTE (PROGN THE OBJECT QUOTED IN THE EXPRESSION (!PPR X NIL)
                                      DOES NOT REPRESENT AN EXPLICIT VALUE TERM))
                       (BINDINGS (QUOTE X)
                              X)
                       (QUOTE SOFT)))
              (T X)))
          ((MEMB (CAR X)
                 (QUOTE (NIL T F)))
           (ERROR1 (PQUOTE (PROGN (!PPR (CAR X)
                                        NIL)
                                  IS AN ILLEGAL FUNCTION SYMBOL %.))
                   (BINDINGS (QUOTE X)
                          X)
                   (QUOTE SOFT)))
          [(EQ (CAR X)
               (QUOTE LIST))
           (COND
              ((NULL (CDR X))
               (TRANSLATE NIL))
              (T (XXXJOIN (QUOTE CONS)
                          (NCONC1 (for ARG in (CDR X) collect (TRANSLATE ARG))
                                  (TRANSLATE NIL]
          [(CAR-CDRP (CAR X))
           (COND
              [(IEQP (LENGTH (CDR X))
                     1)
               (FIXCAR-CDR (LIST (CAR X)
                                 (TRANSLATE (CADR X]
              (T (ERROR1 (PQUOTE (PROGN (!PPR (CAR X)
                                              NIL)
                                        IS A RESERVED ABBREVIATION FOR A CAR-CDR NEST
                                          AND MUST BE GIVEN EXACTLY ONE ARGUMENT %.))
                         (BINDINGS (QUOTE X)
                                X)
                         (QUOTE SOFT]
          (T (ERROR1 (PQUOTE (PROGN PROPERTYLESS-SYMBOLP AND TRANSLATE DO NOT AGREE ON (!PPR (CAR X)
                                                                                                NIL)
                                    %.))
                     (BINDINGS (QUOTE X)
                            X)
                     (QUOTE HARD]
      [(NULL (ARITY (CAR X)))
       (COND
          (IN-BOOT-STRAP-FLG (ERROR1 (PQUOTE (PROGN (!PPR (CAR X)
                                                          NIL)
                                                    HAS BEEN ENCOUNTERED AS AN UNDEFINED FUNCTION BY
                                                    TRANSLATE %. YOU SHOULD ADD IT TO THE BINDING OF
                                                    ARITY-ALIST IN BOOT-STRAP IF YOU WISH TO SUPPRESS THIS
                                                    MESSAGE
                                                !))
                                     (BINDINGS (QUOTE X)
                                            X)
                                     (QUOTE WARNING)))
          (T (ERROR1 (PQUOTE (PROGN THE FUNCTION (!PPR (CAR X)
                                                       NIL)
                                    IS UNKNOWN %. PLEASE DELETE ALL REFERENCES TO IT , DEFINE IT OR DECLARE
                                    IT AS AN UNDEFINED FUNCTION %.))
                     (BINDINGS (QUOTE X)
                            X)
                     (QUOTE SOFT]
      [(AND (MEMB (CAR X)
                  (QUOTE (AND OR PLUS TIMES)))
            (IGREATERP (LENGTH (CDR X))
                       2))
       (XXXJOIN (CAR X)
                (for ARG in (CDR X) collect (TRANSLATE ARG]
      ([NOT (IEQP (LENGTH (CDR X))
                  (ARITY (CAR X]
       (ERROR1 (PQUOTE (PROGN THE FUNCTION SYMBOL (!PPR (CAR X)
                                                        NIL)
                              TAKES EXACTLY (@ N)
                              ARGUMENTS %. IN (!PPR X NIL)
                              IT IS GIVEN THE WRONG NUMBER OF ARGUMENTS %.))
               (BINDINGS (QUOTE X)
```

```
                    X
                    (QUOTE N)
                    (ARITY (CAR X)))
              (QUOTE SOFT)))
       [(MEMB (CAR X)
              BOOT-STRAP-MACRO-FNS)
        (SUB-PAIR-VAR (CADR (GET1 (CAR X)
                                  (QUOTE SDEFN)))
              (for ARG in (CDR X) collect (TRANSLATE ARG))
              (CADDR (GET1 (CAR X)
                           (QUOTE SDEFN]
       (T (CONS-TERM (CAR X)
                 (for ARG in (CDR X) collect (TRANSLATE ARG])
```

(TRANSLATE-TO-LISP

```
  [LAMBDA (X)                                                    (* kbr: "19-Oct-85 16:31")
    (LET (ANS TIME)
         (SETQ TIME (TIME-IN-60THS))
         (SETQ ALL-LEMMAS-USED NIL)
         [SETQ ANS (PRETTYIFY-LISP (OPTIMIZE-COMMON-SUBTERMS (ONEIFY X NIL]
         (SETQ TRANSLATE-TO-LISP-TIME (PLUS (DIFFERENCE (TIME-IN-60THS)
                                                        TIME)
                                            TRANSLATE-TO-LISP-TIME))
         ANS])
```

(TREE-DEPENDENTS

```
  [LAMBDA (NAME)                                                 (* kbr: "19-Oct-85 16:31")
    (CONS NAME (for X in (GET1 NAME (QUOTE IMMEDIATE-DEPENDENTS0)) bind LOOP-ANS
                  do (SETQ LOOP-ANS (UNIONQ (TREE-DEPENDENTS X)
                                            LOOP-ANS))
                  finally (RETURN LOOP-ANS])
```

(TRIVIAL-POLYP

```
  [LAMBDA (POLY)                                                 (* kbr: "19-Oct-85 16:31")
    (OR (TRIVIAL-POLYP1 POLY (QUOTE POSITIVE))
        (TRIVIAL-POLYP1 POLY (QUOTE NEGATIVE])
```

(TRIVIAL-POLYP1

```
  [LAMBDA (POLY PARITY)                                          (* kbr: "19-Oct-85 16:31")
    (PROG (WINNING-PAIR COEF)
          (COND
             [(EQ PARITY (QUOTE POSITIVE))
              (COND
                 ([AND (LESSP (fetch (POLY CONSTANT) of POLY)
                              1)
                       (IEQP 1 (for PAIR in (fetch (POLY ALIST) of POLY) count (GREATERP (CDR PAIR)
                                                                                         0]
                  (SETQ WINNING-PAIR (for PAIR in (fetch (POLY ALIST) of POLY) when (GREATERP (CDR PAIR)
                                                                                              0)
                                        do (RETURN PAIR)))
                  (SETQ COEF (CDR WINNING-PAIR)))
                 (T (RETURN NIL]
             [[AND (GREATERP (fetch (POLY CONSTANT) of POLY)
                             -1)
                   (IEQP 1 (for PAIR in (fetch (POLY ALIST) of POLY) count (LESSP (CDR PAIR)
                                                                                  0]
              (SETQ WINNING-PAIR (for PAIR in (fetch (POLY ALIST) of POLY) when (LESSP (CDR PAIR)
                                                                                       0)
                                    do (RETURN PAIR)))
              (SETQ COEF (MINUS (CDR WINNING-PAIR]
             (T (RETURN NIL)))
          (COND
             [[AND [NOT (BM-MATCH (fetch (POLY LITERALS) of POLY)
                                  (LIST (NOT (EQUAL & &]
                   (EQUAL 0 (REMAINDER (fetch (POLY CONSTANT) of POLY)
                                       COEF))
                   (for PAIR in (fetch (POLY ALIST) of POLY) always (EQUAL 0 (REMAINDER (CDR PAIR)
                                                                                        COEF]
```

(* We know that the polys in this pot list were formed from the current CL with the ADD-TERMS-TO-POT-LST FLG=NIL. That is, the literals of the clause were stored by LINEARIZE with their original parities, even though the poly was generated from their negations. *)

```
              (RETURN (CONS (CONS (CAR WINNING-PAIR)
                                  (COND
                                     ((EQ PARITY (QUOTE POSITIVE))
                                      1)
                                     (T -1)))
                            (create POLY
                                    CONSTANT _ (QUOTIENT (fetch (POLY CONSTANT) of POLY)
                                                         COEF)
                                    ALIST _ (for PAIR in (fetch (POLY ALIST) of POLY)
                                               collect (CONS (CAR PAIR)
```

```
                                                    (QUOTIENT (CDR PAIR)
                                                              COEF)))
                                    ASSUMPTIONS _ (fetch (POLY ASSUMPTIONS) of POLY)
                                    LITERALS _ (fetch (POLY LITERALS) of POLY)
                                    LEMMAS _ (fetch (POLY LEMMAS) of POLY]
          (T (RETURN NIL])
```

(TRUE-POLYP

```
[LAMBDA (POLY)                                                    (* kbr: "19-Oct-85 16:31")
  (AND (LESSEQP (fetch (POLY CONSTANT) of POLY)
                0)
       (for PAIR in (fetch (POLY ALIST) of POLY) always (LESSEQP (CDR PAIR)
                                                                 0])
```

(TYPE-ALIST-CLAUSE

```
[LAMBDA (CL)                                                      (* kbr: "19-Oct-85 16:31")
  (LET ((TYPE-ALIST TYPE-ALIST))
       [for LIT in CL while (NEQ TYPE-ALIST (QUOTE CONTRADICTION)) do (ASSUME-TRUE-FALSE LIT)
                                                                      (COND
                                                                         (MUST-BE-TRUE (SETQ TYPE-ALIST
                                                                                             (QUOTE CONTRADICTION)))
                                                                         (MUST-BE-FALSE NIL)
                                                                         (T (SETQ TYPE-ALIST FALSE-TYPE-ALIST]
       TYPE-ALIST])
```

(TYPE-PRESCRIPTION-LEMMAP

```
[LAMBDA (NAME)                                                    (* kbr: "19-Oct-85 16:31")
  (LET (ATM)
       (COND
          ([for TUPLE in (GET1 NAME (QUOTE LOCAL-UNDO-TUPLES)) thereis (BM-MATCH TUPLE (CONS (QUOTE
                                                                                                TYPE-PRESCRIPTION-LST
                                                                                                      )
                                                                                              (CONS ATM &]
           ATM)
          (T NIL])
```

(TYPE-SET

```
[LAMBDA (TERM)                                                    (* kbr: "19-Oct-85 16:31")
  (LET (PAIR TYPE-ARG1 TYPE-ARG2 ARG1 ARG2)
       (COND
          ((SETQ TEMP-TEMP (SASSOC TERM TYPE-ALIST))
           (CDR TEMP-TEMP))
          ((VARIABLEP TERM)
           TYPE-SET-UNKNOWN)
          [(FQUOTEP TERM)
           (CAR (TYPE-PRESCRIPTION (FN-SYMB0 (CADR TERM]
          ((SETQ PAIR (ASSOC (FFN-SYMB TERM)
                             RECOGNIZER-ALIST))
           (SETQ TYPE-ARG1 (TYPE-SET (FARGN TERM 1)))
           (COND
              ((IEQP 0 (LOGAND TYPE-ARG1 (CDR PAIR)))
               TYPE-SET-FALSE)
              ((LOGSUBSETP TYPE-ARG1 (CDR PAIR))
               TYPE-SET-TRUE)
              (T TYPE-SET-BOOLEAN)))
          ((BM-MATCH TERM (EQUAL ARG1 ARG2))
           (SETQ TYPE-ARG1 (TYPE-SET ARG1))
           (SETQ TYPE-ARG2 (TYPE-SET ARG2))
           (COND
              ((IEQP 0 (LOGAND TYPE-ARG1 TYPE-ARG2))
               TYPE-SET-FALSE)
              ((AND (IEQP TYPE-ARG1 TYPE-ARG2)
                    (MEMBER TYPE-ARG1 SINGLETON-TYPE-SETS))
               TYPE-SET-TRUE)
              (T TYPE-SET-BOOLEAN)))
          ((BM-MATCH TERM (NOT ARG1))
           (SETQ TYPE-ARG1 (TYPE-SET ARG1))
           (COND
              ((IEQP TYPE-ARG1 TYPE-SET-FALSE)
               TYPE-SET-TRUE)
              ((NOT (LOGSUBSETP TYPE-SET-FALSE TYPE-ARG1))
               TYPE-SET-FALSE)
              (T TYPE-SET-BOOLEAN)))
          [(EQ (FFN-SYMB TERM)
               (QUOTE IF))
           (ASSUME-TRUE-FALSE (FARGN TERM 1))
           (COND
              (MUST-BE-TRUE (TYPE-SET (FARGN TERM 2)))
              (MUST-BE-FALSE (TYPE-SET (FARGN TERM 3)))
              (T (LOGOR (TYPE-SET2 (FARGN TERM 2)
                                   TRUE-TYPE-ALIST)
                        (TYPE-SET2 (FARGN TERM 3)
                                   FALSE-TYPE-ALIST]
```

```
            [(SETQ TEMP-TEMP (TYPE-PRESCRIPTION (FFN-SYMB TERM)))
             (LOGOR (CAR TEMP-TEMP)
                    (for ARG in (FARGS TERM) as FLG in (CDR TEMP-TEMP) bind (LOOP-ANS _ 0) when FLG
                       do (SETQ LOOP-ANS (LOGOR LOOP-ANS (TYPE-SET ARG))) finally (RETURN LOOP-ANS]
            (T TYPE-SET-UNKNOWN])
```

(TYPE-SET2

```
[LAMBDA (TERM TYPE-ALIST)                                   (* kbr: "19-Oct-85 16:31")
```

(* This is like TYPE-SET, only it lets you specify the local TYPE-ALIST and protects the FALSE-TYPE-ALIST for you. *)

```
   (LET (FALSE-TYPE-ALIST)
        (TYPE-SET TERM])
```

(UBT

```
[LAMBDA (NAME)                                              (* kbr: "19-Oct-85 16:31")
  (UNDO-BACK-THROUGH NAME)
  NAME])
```

(UNBREAK-LEMMA

```
[LAMBDA (NAME)                                              (* kbr: "19-Oct-85 16:31")
  (COND
     ((NULL NAME)
      (SETQ BROKEN-LEMMAS NIL))
     (T (SETQ BROKEN-LEMMAS (REMOVE (ASSOC NAME BROKEN-LEMMAS)
                                    BROKEN-LEMMAS])
```

(UNCHANGING-VARS

```
[LAMBDA (TERM)                                              (* kbr: "19-Oct-85 16:31")
  (LET (ANS)
       (UNCHANGING-VARS1 (EXPAND-NON-REC-FNS TERM))
       ANS])
```

(UNCHANGING-VARS1

```
[LAMBDA (TERM)                                              (* kbr: "19-Oct-85 16:31")
  (COND
     ((VARIABLEP TERM)
      NIL)
     ((FQUOTEP TERM)
      NIL)
     (T (for ARG in (FARGS TERM) do (UNCHANGING-VARS1 ARG))
        (COND
           ([OR (MEMB (FFN-SYMB TERM)
                      *1*BTM-OBJECTS)
                (ASSOC (FFN-SYMB TERM)
                       RECOGNIZER-ALIST)
                (for X in SHELL-POCKETS thereis (MEMB (FFN-SYMB TERM)
                                                      X))
                (MEMB (FFN-SYMB TERM)
                      (QUOTE (IF EQUAL]
            NIL)
           ([AND (GET1 (FFN-SYMB TERM)
                       (QUOTE SDEFN))
                 (NOT (DISABLEDP (FFN-SYMB TERM]
            NIL)
           (T (for ARG in (FARGS TERM) when (VARIABLEP ARG) do (SETQ ANS (ADD-TO-SET ARG ANS])
```

(UNDO-BACK-THROUGH

```
[LAMBDA (NAME)                                              (* kbr: "19-Oct-85 16:31")
  (COND
     ((NOT (GET1 NAME (QUOTE EVENT)))
      (ERROR1 (PQUOTE (PROGN ATTEMPT TO UNDO A NONEVENT , (!PPR NAME NIL)
                             %.))
              (BINDINGS (QUOTE NAME)
                        NAME)
              (QUOTE SOFT)))
     (T (DREVERSE (while (AND (BOUNDP (QUOTE CHRONOLOGY))
                              (MEMB NAME CHRONOLOGY))
                     do (APPEND (UNDO-NAME (CAR CHRONOLOGY])
```

(UNDO-NAME

```
[LAMBDA (NAME)                                              (* kbr: "19-Oct-85 16:31")
  (LET (EVENTS)
       (COND
          ((NOT (GET1 NAME (QUOTE EVENT)))
           (ERROR1 (PQUOTE (PROGN ATTEMPT TO UNDO A NONEVENT , (!PPR NAME NIL)
                                  %.))
                   (BINDINGS (QUOTE NAME)
                             NAME)
```

```
                (QUOTE SOFT)))
            ((EQ NAME (QUOTE GROUND-ZERO))
             [SETQ EVENTS (DREVERSE (for X in CHRONOLOGY collect (GET1 X (QUOTE EVENT]
             (KILL-LIB)
             EVENTS)
            (T (SETQ EVENTS (REVERSE (DEPENDENTS-OF NAME)))
               (DREVERSE (for X in EVENTS collect (PROG1 (GET1 X (QUOTE EVENT))
                                                         (KILL-EVENT X])
```

(UNION-EQUAL

```
[LAMBDA (X Y)                                                    (* kbr: "19-Oct-85 16:31")
```

(* When we moved to the 3600 we replaced calls of INTERLISP's UNIONQ --
which uses EQUAL -- with our own UNION-EQUAL because Zetalisp's UNIONQ uses EQ.
Some calls of INTERLISP's UNIONQ were allowed to remain UNIONs because we could convince ourselves that only atoms were involved. However, on questionable cases we went ahead and used UNION-EQUAL.
Thus, some calls of UNION-EQUAL could be replaced by UNION.
The main place is when dealing with lemmas used, where inside the simpblock we permit listp names.
Seeing a call of UNION-EQUAL in such a situation is not to be taken as a claim that listp names are present we just didn't trace it out. *)

```
  (NCONC (for Z in X unless (MEMBER Z Y) collect Z)
         Y])
```

(UNPRETTYIFY

```
[LAMBDA (TERM)                                                   (* kbr: "19-Oct-85 16:31")
```

(* This function returns a list of pairs (hyps) such that the conjunction of all
(IMPLIES hyps concl) is equivalent to TERM. hyps is a list of hypotheses, implicitly conjoined.
concl does not begin with an AND or IMPLIES. *)

```
  (LET (C1 C2 HYP CONCL)
       (COND
          ((BM-MATCH TERM (AND C1 C2))
           (APPEND (UNPRETTYIFY C1)
                   (UNPRETTYIFY C2)))
          [(BM-MATCH TERM (IMPLIES HYP CONCL))
           (SETQ HYP (FLATTEN-ANDS-IN-LIT HYP))
           (for PAIR in (UNPRETTYIFY CONCL) collect (CONS (APPEND HYP (CAR PAIR))
                                                          (CDR PAIR]
          (T (LIST (CONS NIL TERM])
```

(VARIANTP

```
[LAMBDA (TERM1 TERM2)                                            (* kbr: "19-Oct-85 16:31")
  (AND (ONE-WAY-UNIFY TERM1 TERM2)
       (for PAIR in UNIFY-SUBST always (VARIABLEP (CDR PAIR)))
       (NO-DUPLICATESP (for PAIR in UNIFY-SUBST collect (CDR PAIR])
```

(WORSE-THAN

```
[LAMBDA (TERM1 TERM2)                                            (* kbr: "22-Oct-85 15:47")
                                                                 (* Is TERM1 syntactically worse than TERM2? *)
  (COND
     ((QUICK-WORSE-THAN TERM1 TERM2)
      T)
     ((VARIABLEP TERM1)
      NIL)
     ((FQUOTEP TERM1)
      NIL)
     (T (for ARG in (FARGS TERM1) thereis (SOME-SUBTERM-WORSE-THAN-OR-EQUAL ARG TERM2])
```

(WORSE-THAN-OR-EQUAL

```
[LAMBDA (TERM1 TERM2)                                            (* kbr: "19-Oct-85 16:31")
  (OR (EQUAL TERM1 TERM2)
      (WORSE-THAN TERM1 TERM2])
```

(WRAPUP

```
[LAMBDA (WON-FLG)                                                (* kbr: "19-Oct-85 20:15")
  [COND
     ((NEQ LEMMA-STACK ORIG-LEMMA-STACK)
      (ITERPRI T)
      (ERROR1 (PQUOTE (PROGN WRAPUP FOUND A NON-TRIVIAL LEMMA-STACK !))
              (BINDINGS)
              (QUOTE HARD]
  [COND
     ((NEQ LINEARIZE-ASSUMPTIONS-STACK ORIG-LINEARIZE-ASSUMPTIONS-STACK)
      (ITERPRI T)
      (ERROR1 (PQUOTE (PROGN WRAPUP FOUND A NON-TRIVIAL LINEARIZE-ASSUMPTIONS-STACK !))
              (BINDINGS)
              (QUOTE HARD]
  [COND
     (WON-FLG (SETQ FAILED-THMS (REMOVE ORIGTHM FAILED-THMS))
```

```
              (SETQ PROVED-THMS (CONS ORIGTHM PROVED-THMS]
    (IO (QUOTE FINISHED)
        NIL NIL NIL (LIST WON-FLG))
    (RETFROM (QUOTE PROVE)
           (COND
              (WON-FLG (QUOTE PROVED))
              (T NIL])
```

(**XXXJOIN**

```
  [LAMBDA (FN X)                                                    (* kbr: "19-Oct-85 16:31")
    (COND
       ((OR (NLISTP X)
            (NLISTP (CDR X)))
        (ERROR1 (PQUOTE (PROGN XXXJOIN MUST NOT BE CALLED ON A LIST WITH LESS THAN 2 ELEMENTS %.))
               NIL
               (QUOTE HARD)))
       ((NLISTP (CDDR X))
        (CONS-TERM FN X))
       (T (CONS-TERM FN (LIST (CAR X)
                              (XXXJOIN FN (CDR X])
```

(**ZERO-POLY**

```
  [LAMBDA (LIT)                                                     (* kbr: "19-Oct-85 16:31")
    (create POLY
           CONSTANT _ 0
           LITERALS _ (LIST LIT])

)

(RPAQQ EVENTSCOMS
       [(* EVENTS *)
        (FNS BOOT-STRAP ADD-AXIOM ADD-SHELL DCL DEFN DEFN& DISABLE ENABLE PROVE-LEMMA PROVE-LEMMA& REFLECT
             TOGGLE)
        (DECLARE: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS
               (ADDVARS (NLAMA TOGGLE REFLECT PROVE-LEMMA ENABLE DISABLE DEFN DCL ADD-SHELL ADD-AXIOM)
                      (NLAML)
                      (LAMA])


         (* * EVENTS *)

(DEFINEQ
```

(**BOOT-STRAP**

```
  [LAMBDA NIL                                                       (* kbr: "19-Oct-85 18:14")
    (LET ((IN-BOOT-STRAP-FLG T))
         (COND
            ((NOT IN-REDO-UNDONE-EVENTS-FLG)
             (CAR (REDO-UNDONE-EVENTS (LIST (LIST (QUOTE BOOT-STRAP)))
                         T
                         (QUOTE C)
                         NIL T T)))
            (T (LET [MAIN-EVENT-NAME (ARITY-ALIST (QUOTE ((NOT . 1)
                                                          (AND . 2)
                                                          (OR . 2)
                                                          (IMPLIES . 2)
                                                          (LESSP . 2)
                                                          (PLUS . 2]
                     (BOOT-STRAP0)
                     (CREATE-EVENT (QUOTE GROUND-ZERO)
                            (LIST (QUOTE BOOT-STRAP)))
                     (ADD-FACT (QUOTE IF)
                            (QUOTE LISP-CODE)
                            (QUOTE *1*IF))
                     (ADD-FACT (QUOTE EQUAL)
                            (QUOTE LISP-CODE)
                            (QUOTE *1*EQUAL))
                     [ADD-FACT (QUOTE IF)
                            (QUOTE TYPE-PRESCRIPTION-LST)
                            (CONS (QUOTE GROUND-ZERO)
                                  (QUOTE (0 NIL T T]
                     [ADD-FACT (QUOTE EQUAL)
                            (QUOTE TYPE-PRESCRIPTION-LST)
                            (CONS (QUOTE GROUND-ZERO)
                                  (CONS TYPE-SET-BOOLEAN (QUOTE (NIL NIL]
                     (ADD-FACT (QUOTE COUNT)
                            (QUOTE LISP-CODE)
                            (QUOTE *1*COUNT))
                     [ADD-FACT (QUOTE COUNT)
                            (QUOTE TYPE-PRESCRIPTION-LST)
                            (CONS (QUOTE GROUND-ZERO)
                                  (CONS TYPE-SET-NUMBERS (QUOTE (NIL]
                     (for INSTR in BOOT-STRAP-INSTRS do (APPLY (CAR INSTR)
                                                               (CDR INSTR)))
                     (SETQ FAILED-THMS NIL)
```

```
                    (QUOTE GROUND-ZERO])
```

(**ADD-AXIOM**

```
  [NLAMBDA $FEXPR$                                                    (* kbr: "19-Oct-85 16:31")
    ([LAMBDA (NAME TYPES TERM)
       (LET ((IN-ADD-AXIOM-FLG T))
            (COND
               ((NOT IN-REDO-UNDONE-EVENTS-FLG)
                (CAR (REDO-UNDONE-EVENTS (LIST (LIST (QUOTE ADD-AXIOM)
                                                     NAME TYPES TERM))
                                         T
                                         (QUOTE C)
                                         NIL T T)))
               (T (LET (MAIN-EVENT-NAME)
                       (CHK-ACCEPTABLE-LEMMA NAME TYPES TERM)
                       (CREATE-EVENT NAME (LIST (QUOTE ADD-AXIOM)
                                                NAME TYPES TERM))
                       (ADD-FACT NIL (QUOTE NONCONSTRUCTIVE-AXIOM-NAMES)
                             NAME)
                       (ADD-LEMMA0 NAME TYPES TERM)
                       (DEPEND NAME (ALL-FNNAMES (TRANSLATE TERM)))
                       NAME]
     (pop $FEXPR$)
     (pop $FEXPR$)
     (pop $FEXPR$])
```

(**ADD-SHELL**

```
  [NLAMBDA $FEXPR$                                                    (* kbr: "19-Oct-85 16:31")
    ([LAMBDA (SHELL-NAME BTM-FN-SYMB RECOGNIZER DESTRUCTOR-TUPLES)
       (COND
          ((NOT IN-REDO-UNDONE-EVENTS-FLG)
           (CAR (REDO-UNDONE-EVENTS (LIST (LIST (QUOTE ADD-SHELL)
                                                SHELL-NAME BTM-FN-SYMB RECOGNIZER DESTRUCTOR-TUPLES))
                                    T
                                    (QUOTE C)
                                    NIL T T)))
          (T (LET (MAIN-EVENT-NAME)
                  (CHK-ACCEPTABLE-SHELL SHELL-NAME BTM-FN-SYMB RECOGNIZER DESTRUCTOR-TUPLES)
                  (CREATE-EVENT SHELL-NAME (LIST (QUOTE ADD-SHELL)
                                                 SHELL-NAME BTM-FN-SYMB RECOGNIZER DESTRUCTOR-TUPLES))
                  (ADD-SHELL0 SHELL-NAME BTM-FN-SYMB RECOGNIZER DESTRUCTOR-TUPLES)
                  [DEPEND SHELL-NAME (SET-DIFF (UNIONQ (for X in DESTRUCTOR-TUPLES bind LOOP-ANS
                                                          do (SETQ LOOP-ANS (UNIONQ (CDR (CADR X))
                                                                                    LOOP-ANS))
                                                          finally (RETURN LOOP-ANS))
                                                       (for X in DESTRUCTOR-TUPLES bind LOOP-ANS
                                                          do (SETQ LOOP-ANS (ADD-TO-SET (CADDR X)
                                                                                        LOOP-ANS))
                                                          finally (RETURN LOOP-ANS)))
                                               (COND
                                                  (BTM-FN-SYMB (LIST BTM-FN-SYMB RECOGNIZER))
                                                  (T (LIST RECOGNIZER]

          (* Make the shell depend on every fn used in the type restrictions and defaults except the BTM-FN-SYMB and
          RECOGNIZER of this type. *)

                  SHELL-NAME]
     (pop $FEXPR$)
     (pop $FEXPR$)
     (pop $FEXPR$)
     (pop $FEXPR$])
```

(**DCL**

```
  [NLAMBDA $FEXPR$                                                    (* kbr: "19-Oct-85 16:31")
    ([LAMBDA (NAME ARGS)
       (COND
          ((NOT IN-REDO-UNDONE-EVENTS-FLG)
           (CAR (REDO-UNDONE-EVENTS (LIST (LIST (QUOTE DCL)
                                                NAME ARGS))
                                    T
                                    (QUOTE C)
                                    NIL T T)))
          (T (LET (MAIN-EVENT-NAME)
                  (CHK-ACCEPTABLE-DCL NAME ARGS)
                  (CREATE-EVENT NAME (LIST (QUOTE DCL)
                                           NAME ARGS))
                  (DCL0 NAME ARGS)
                  NAME]
     (pop $FEXPR$)
     (pop $FEXPR$])
```

(**DEFN**

```
  [NLAMBDA $FEXPR$                                                    (* kbr: "19-Oct-85 16:31")
```

```
    ([LAMBDA (NAME ARGS BODY RELATION-MEASURE-LST)
       (COND
          ((NOT IN-REDO-UNDONE-EVENTS-FLG)
           (CAR (REDO-UNDONE-EVENTS (LIST (LIST (QUOTE DEFN)
                                                NAME ARGS BODY RELATION-MEASURE-LST))
                                    T
                                    (QUOTE C)
                                    NIL T T)))
          (T (LET (MAIN-EVENT-NAME)
                  (CHK-ACCEPTABLE-DEFN NAME ARGS BODY RELATION-MEASURE-LST)
                  [CREATE-EVENT NAME (COND
                                        (RELATION-MEASURE-LST (LIST (QUOTE DEFN)
                                                                    NAME ARGS BODY RELATION-MEASURE-LST))
                                        (T (LIST (QUOTE DEFN)
                                                 NAME ARGS BODY]
                  (DEFN0 NAME ARGS BODY RELATION-MEASURE-LST NIL)
                  [DEPEND NAME
                          (REMOVE NAME
                                  (UNIONQ (ALL-FNNAMES (TRANSLATE BODY))
                                          (UNIONQ ALL-LEMMAS-USED
                                                  (for TEMP in (GETPROP NAME (QUOTE JUSTIFICATIONS)) bind LOOP-ANS
                                                     do (SETQ LOOP-ANS
                                                              (UNIONQ [COND
                                                                         ((NULL (fetch (JUSTIFICATION RELATION)
                                                                                   of TEMP))
                                                                          NIL)
                                                                         (T (UNIONQ (ALL-FNNAMES (fetch (JUSTIFICATION
                                                                                                           MEASURE-TERM)
                                                                                                    of TEMP))
                                                                                    (ADD-TO-SET (fetch (JUSTIFICATION
                                                                                                        RELATION)
                                                                                                   of TEMP)
                                                                                                (fetch (JUSTIFICATION LEMMAS)
                                                                                                   of TEMP]
                                                                      LOOP-ANS))
                                                     finally (RETURN LOOP-ANS]
                  (PRINT-DEFN-MSG NAME ARGS)
                  (DEFN-WRAPUP (TOTAL-FUNCTIONP NAME))
                  (COND
                     ((TOTAL-FUNCTIONP NAME)
                      NAME)
                     (T NIL]
     (pop $FEXPR$)
     (pop $FEXPR$)
     (pop $FEXPR$)
     (pop $FEXPR$])
```

(DEFN&

```
  [LAMBDA (NAME)                                          (* kbr: "29-Jun-86 16:14")
    (PROG (FORM)
          (for PROP in LIB-PROPS do (REMPROP NAME PROP))
          (PUTD (PACK* (QUOTE *1*)
                       NAME)
                NIL)
          [SETQ FORM (CONS (QUOTE DEFN)
                           (CONS NAME (GETPROP NAME (QUOTE DEFN]
          (SHOWPRINT FORM)
          (EVAL FORM])
```

(DISABLE

```
  [NLAMBDA $FEXPR$                                        (* kbr: "19-Oct-85 16:31")
    ([LAMBDA (OLDNAME)
       (APPLY (FUNCTION TOGGLE)
              (LIST (MAKE-NEW-NAME)
                    OLDNAME T]
     (pop $FEXPR$])
```

(ENABLE

```
  [NLAMBDA $FEXPR$                                        (* kbr: "19-Oct-85 16:31")
    ([LAMBDA (OLDNAME)
       (APPLY (FUNCTION TOGGLE)
              (LIST (MAKE-NEW-NAME)
                    OLDNAME NIL]
     (pop $FEXPR$])
```

(PROVE-LEMMA

```
  [NLAMBDA $FEXPR$                                        (* kbr: "19-Apr-86 18:10")
    ([LAMBDA (NAME TYPES TERM HINTS)
       (LET ((IN-PROVE-LEMMA-FLG T))
            (COND
               ((NOT IN-REDO-UNDONE-EVENTS-FLG)
                (CAR (REDO-UNDONE-EVENTS (LIST (LIST (QUOTE PROVE-LEMMA)
                                                     NAME TYPES TERM HINTS))
```

```
                        T
                        (QUOTE C)
                        NIL T T)))
              (T (LET (PROVE-ANS MAIN-EVENT-NAME)
                      (CHK-ACCEPTABLE-LEMMA NAME TYPES TERM)
                      (CHK-ACCEPTABLE-HINTS HINTS)
                      (NLSETQ (PROGN
```

(* Before calling PROVE we call APPLY-HINTS. APPLY-HINTS sets some global variables that affect the theorem-prover. We enter an UNWIND-PROTECT here so that we can set those variables to their standard default values no matter how we exit PROVE. *)

```
                                    (SETQ PROVE-ANS (PROVE (APPLY-HINTS HINTS TERM)))
                                    [COND
                                       (PROVE-ANS [CREATE-EVENT NAME (COND
                                                                        (HINTS (LIST (QUOTE PROVE-LEMMA)
                                                                                     NAME TYPES TERM HINTS))
                                                                        (T (LIST (QUOTE PROVE-LEMMA)
                                                                                 NAME TYPES TERM]
                                                  (ADD-LEMMA0 NAME TYPES TERM)
                                                  (DEPEND NAME (UNIONQ ALL-LEMMAS-USED
                                                                       (UNIONQ (EXTRACT-DEPENDENCIES-FROM-HINTS
                                                                                  HINTS)
                                                                               (ALL-FNNAMES (TRANSLATE TERM]
                                    (COND
                                       (PROVE-ANS NAME)
                                       (T NIL)))
                                 (for X in HINT-VARIABLE-ALIST do (SET (CADR X)
                                                                       (CADDDR X]
     (pop $FEXPR$)
     (pop $FEXPR$)
     (pop $FEXPR$)
     (pop $FEXPR$])
```

(PROVE-LEMMA&

```
  [LAMBDA (NAME)                                                    (* kbr: "29-Jun-86 16:16")
    (PROG (FORM)
          [SETQ FORM (CONS (QUOTE PROVE-LEMMA)
                           (CONS NAME (GETPROP NAME (QUOTE PROVE-LEMMA]
          (SHOWPRINT FORM T)
          (EVAL FORM])
```

(REFLECT

```
  [NLAMBDA $FEXPR$                                                  (* kbr: "19-Oct-85 16:31")
    ([LAMBDA (NAME SATISFACTION-LEMMA-NAME RELATION-MEASURE-LST)
       (COND
          ((NOT IN-REDO-UNDONE-EVENTS-FLG)
           (CAR (REDO-UNDONE-EVENTS (LIST (LIST (QUOTE REFLECT)
                                                NAME SATISFACTION-LEMMA-NAME RELATION-MEASURE-LST))
                                    T
                                    (QUOTE C)
                                    NIL T T)))
          (T (LET (MAIN-EVENT-NAME)
                  (DEFN-SETUP (LIST (QUOTE REFLECT)
                                    NAME SATISFACTION-LEMMA-NAME RELATION-MEASURE-LST))
                  (CHK-ACCEPTABLE-REFLECT NAME SATISFACTION-LEMMA-NAME RELATION-MEASURE-LST)
                  [CREATE-EVENT NAME (COND
                                        (RELATION-MEASURE-LST (LIST (QUOTE REFLECT)
                                                                    NAME SATISFACTION-LEMMA-NAME
                                                                    RELATION-MEASURE-LST))
                                        (T (LIST (QUOTE REFLECT)
                                                 NAME SATISFACTION-LEMMA-NAME]
                  (REFLECT0 NAME SATISFACTION-LEMMA-NAME RELATION-MEASURE-LST NIL)
                  [DEPEND NAME
                          (REMOVE NAME
                                  (ADD-TO-SET
                                   SATISFACTION-LEMMA-NAME
                                   (UNION-EQUAL ALL-LEMMAS-USED
                                                (for TEMP in (GETPROP NAME (QUOTE JUSTIFICATIONS)) bind LOOP-ANS
                                                   do (SETQ LOOP-ANS (UNIONQ [COND
                                                                                ((NULL (fetch (JUSTIFICATION RELATION)
                                                                                          of TEMP))
                                                                                 NIL)
                                                                                (T (UNIONQ (ALL-FNNAMES
                                                                                              (fetch (JUSTIFICATION
                                                                                                        MEASURE-TERM)
                                                                                                 of TEMP))
                                                                                           (ADD-TO-SET
                                                                                            (fetch (JUSTIFICATION
                                                                                                      RELATION)
                                                                                               of TEMP)
                                                                                            (fetch (JUSTIFICATION LEMMAS)
                                                                                               of TEMP]
                                                                             LOOP-ANS))
                                                   finally (RETURN LOOP-ANS]
```

```
                  [PRINT-DEFN-MSG NAME (CADR (GETPROP NAME (QUOTE SDEFN]
                  (DEFN-WRAPUP (TOTAL-FUNCTIONP NAME))
                  (COND
                     ((TOTAL-FUNCTIONP NAME)
                      NAME)
                     (T NIL]
     (pop $FEXPR$)
     (pop $FEXPR$)
     (pop $FEXPR$])
```

(TOGGLE

```
  [NLAMBDA $FEXPR$                                                    (* kbr: "19-Oct-85 16:31")
    ([LAMBDA (NAME OLDNAME FLG)
       (COND
          ((NOT IN-REDO-UNDONE-EVENTS-FLG)
           (CAR (REDO-UNDONE-EVENTS (LIST (LIST (QUOTE TOGGLE)
                                                NAME OLDNAME FLG))
                        T
                        (QUOTE C)
                        NIL T T)))
          (T (LET (MAIN-EVENT-NAME)
                  (CHK-ACCEPTABLE-TOGGLE NAME OLDNAME FLG)
                  (CREATE-EVENT NAME (LIST (QUOTE TOGGLE)
                                           NAME OLDNAME FLG))
                  (ADD-FACT NIL (QUOTE DISABLED-LEMMAS)
                            (CONS OLDNAME (CONS NAME FLG)))
                  (DEPEND NAME (LIST (MAIN-EVENT-OF OLDNAME)))
                  NAME]
     (pop $FEXPR$)
     (pop $FEXPR$)
     (pop $FEXPR$])

)

(DECLARE: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS

(ADDTOVAR NLAMA TOGGLE REFLECT PROVE-LEMMA ENABLE DISABLE DEFN DCL ADD-SHELL ADD-AXIOM)

(ADDTOVAR NLAML )

(ADDTOVAR LAMA )
)

(RPAQQ GENFACTCOMS ((* GENFACT *)
                    (FNS GENERATE-ADD-FACT-PART GENERATE-ADD-SUB-FACT1 GENERATE-SUB-FACT-PART
                         GENERATE-UNDO-TUPLE-PART)))


          (* * GENFACT *)

(DEFINEQ
```

(GENERATE-ADD-FACT-PART

```
  [LAMBDA (ALIST)                                                     (* kbr: "29-Oct-85 13:51")
    (LET (!SINGLE-PROPS! !ADDITIVE-PROPS! !ADDITIVE-VARS! !SINGLE-VARS!)
         (SETQ !SINGLE-PROPS! (for X in ALIST when (AND (EQ (CADR X)
                                                            (QUOTE SINGLE))
                                                        (EQ (CADDR X)
                                                            (QUOTE PROPERTY)))
                                 collect (CAR X)))
         (SETQ !ADDITIVE-PROPS! (for X in ALIST when (AND (EQ (CADR X)
                                                              (QUOTE ADDITIVE))
                                                          (EQ (CADDR X)
                                                              (QUOTE PROPERTY)))
                                   collect (CAR X)))
         (SETQ !ADDITIVE-VARS! (for X in ALIST when (AND (EQ (CADR X)
                                                             (QUOTE ADDITIVE))
                                                         (EQ (CADDR X)
                                                             (QUOTE VARIABLE)))
                                  collect (CAR X)))
         (SETQ !SINGLE-VARS! (for X in ALIST when (AND (EQ (CADR X)
                                                           (QUOTE SINGLE))
                                                       (EQ (CADDR X)
                                                           (QUOTE VARIABLE)))
                                collect (CAR X)))
         (BQUOTE (PROGN [COND
                           ((NULL VAL)
                            (ERROR1 (PQUOTE (PROGN ATTEMPT TO DO AN ADD-FACT WITH VALUE (!PPR NIL NIL)
                                                   ON
                                                   (!PPR PROP NIL) AND (!PPR ATM NIL)
                                                                       %.))
                                    (BINDINGS (QUOTE PROP)
                                              PROP
                                              (QUOTE ATM)
                                              ATM)
                                    (QUOTE HARD]
```

```
                    (SELECTQ PROP
                         ((\, !SINGLE-PROPS!)
                              [COND
                                 ((GETPROP ATM PROP)
                                  (ERROR1 (PQUOTE (PROGN ATTEMPT TO SMASH EXISTING SINGLE PROPERTY FACT
                                                         HUNG UNDER (!PPR PROP NIL)
                                                         OF
                                                         (!PPR ATM NIL)
                                                         %.))
                                          (BINDINGS (QUOTE PROP)
                                                 PROP
                                                 (QUOTE ATM)
                                                 ATM)
                                          (QUOTE HARD]
                              (PUT1 ATM VAL PROP))
                         ((\, !ADDITIVE-PROPS!)
                              (PUT1 ATM (CONS VAL (GETPROP ATM PROP))
                                    PROP))
                         (DCELL (COND
                                   ((GETD ATM)
                                    (ERROR1 (PQUOTE (PROGN ATTEMPT TO SMASH EXISTING LISP DEFINITION CELL
                                                           OF THE FUNCTION (!PPR ATM NIL)
                                                           %.))
                                            (BINDINGS (QUOTE ATM)
                                                   ATM)
                                            (QUOTE HARD)))
                                   (T (PUTD1 ATM VAL))))
                         ((\, !ADDITIVE-VARS!)
                              (OR (NULL ATM)
                                  (ERROR1 (PQUOTE (PROGN ADD-SUB-FACT MUST NOT BE CALLED WITH PROP SET TO A
                                                         VARIABLE NAME WHILE ATM IS NON-NIL BECAUSE IT
                                                         CONFUSES THE UNDO INFORMATION %.))
                                          NIL
                                          (QUOTE HARD)))
                              (SET PROP (CONS VAL (EVALV PROP))))
                         ((\, !SINGLE-VARS!)
                              (OR (NULL ATM)
                                  (ERROR1 (PQUOTE (PROGN ADD-SUB-FACT MUST NOT BE CALLED WITH PROP SET TO A
                                                         VARIABLE NAME WHILE ATM IS NON-NIL BECAUSE IT
                                                         CONFUSES THE UNDO INFORMATION %.))
                                          NIL
                                          (QUOTE HARD)))
                              [COND
                                 ((BOUNDP PROP)
                                  (ERROR1 (PQUOTE (PROGN ATTEMPT TO SMASH EXISTING SINGLE VARIABLE
                                                         (\, (!PPR PROP NIL))
                                                         %.))
                                          (BINDINGS (QUOTE PROP)
                                                 PROP)
                                          (QUOTE HARD]
                              (SET PROP VAL))
                         (ERROR1 (PQUOTE (PROGN ADD-SUB-FACT HAS BEEN CALLED ON A PROPERTY
                                                OR VARIABLE NAME NAMELY (\, (!PPR PROP NIL))
                                                   THAT WAS NOT DECLARED
                                                !))
                                 (BINDINGS (QUOTE PROP)
                                        PROP)
                                 (QUOTE HARD])
```

(GENERATE-ADD-SUB-FACT1

```
[LAMBDA (ALIST)                                                    (* kbr: "24-Oct-85 16:35")
  (COND
     [[AND [for X in (QUOTE (IDATE SATELLITES MAIN-EVENT EVENT LOCAL-UNDO-TUPLES))
           always (AND (SETQ TEMP-TEMP (ASSOC X ALIST))
                       (BM-MATCH (CDR TEMP-TEMP)
                              (LIST (QUOTE HIDDEN)
                                    (QUOTE PROPERTY]
           (BM-MATCH (ASSOC (QUOTE CHRONOLOGY)
                            ALIST)
                  (LIST (QUOTE CHRONOLOGY)
                        (QUOTE HIDDEN)
                        (QUOTE VARIABLE)))
           (for X in ALIST never (AND (EQ (CADR X)
                                          (QUOTE HIDDEN))
                                      (NOT (MEMB (CAR X)
                                                 (QUOTE (IDATE SATELLITES MAIN-EVENT EVENT LOCAL-UNDO-TUPLES
                                                         CHRONOLOGY]
      (SUB-PAIR (QUOTE (!LIB-PROPS! !LIBVARS! !SUBTRACT-FACT! !UNDO-TUPLE! !ADD-FACT!))
             (LIST (DREVERSE (for X in ALIST when (EQ (CADDR X)
                                                      (QUOTE PROPERTY))
                                collect (CAR X)))
                   (for X in ALIST when (EQ (CADDR X)
                                            (QUOTE VARIABLE))
                      collect (CAR X))
                   (GENERATE-SUB-FACT-PART ALIST)
                   (GENERATE-UNDO-TUPLE-PART ALIST)
```

```
                    (GENERATE-ADD-FACT-PART ALIST))
               (QUOTE (COND (INIT (INIT-LIB (QUOTE !LIB-PROPS!)
                                            (QUOTE !LIBVARS!))))
                            (TUPLE !SUBTRACT-FACT!)
                            (T [COND ([OR (EQ MAIN-EVENT-NAME (QUOTE GROUND-ZERO))
                                          (AND [OR (EQ MAIN-EVENT-NAME ATM)
                                                   (AND ATM (EQ MAIN-EVENT-NAME (GETPROP ATM (QUOTE MAIN-EVENT]
                                               (NEQ PROP (QUOTE DCELL]
                                     NIL)
                                    (T (PUT1 MAIN-EVENT-NAME (CONS !UNDO-TUPLE! (GETPROP MAIN-EVENT-NAME
                                                                                         (QUOTE LOCAL-UNDO-TUPLES
                                                                                                ))))
                                                (QUOTE LOCAL-UNDO-TUPLES]
                                !ADD-FACT!]
          (T (ERROR (QUOTE (THE USER MUST DECLARE ALL THE BUILT-IN EVENT LEVEL PROPERTIES AND VARIABLES AS HIDDEN
                            AND MUST NOT DECLARE ANY OTHER HIDDEN DATA.])
```

(GENERATE-SUB-FACT-PART

```
[LAMBDA (ALIST)                                                    (* kbr: "20-Apr-86 18:39")
  (SUBST (CONS (QUOTE SELECTQ)
               (CONS (QUOTE PROP)
                     (NCONC1 (for X in ALIST when (EQ (CADR X)
                                                      (QUOTE ADDITIVE))
                                 collect (LIST (CAR X)
                                               (CADDDR X)))
                             NIL)))
         (QUOTE !VAL-NAME!)
         (QUOTE (LET (ATM PROP VAL-NAME VAL TEMP)
                     [COND ((NLISTP TUPLE)
                            (SETQ PROP TUPLE)
                            (SET PROP NIL))
                           [(NLISTP (CDR TUPLE))
                            (SETQ PROP (CAR TUPLE))
                            (SETQ ATM (CDR TUPLE))
                            (COND ((EQ PROP (QUOTE DCELL))
                                   (PUTD1 ATM NIL))
                                  (T (PUTPROP ATM PROP NIL]
                           (T (SETQ PROP (CAR TUPLE))
                              (SETQ ATM (CADR TUPLE))
                              (SETQ VAL-NAME (CDDR TUPLE))
                              (* In the following (and in the LET above)
                                 TEMP was introduced to skirt a bug in the Release 5.0 compiler. *)
                              (SETQ TEMP (FOR VAL IN (COND ((NULL ATM)
                                                            (EVALV PROP))
                                                           (T (GETPROP ATM PROP)))
                                            WHEN
                                            (EQUAL !VAL-NAME! VAL-NAME)
                                            DO
                                            (RETURN VAL)))
                              [COND ((NULL TEMP)
                                     (ERROR1 (PQUOTE (PROGN IN UNDOING AN ADDITIVE ADD-FACT ON
                                                            (!PPR ATM NIL)
                                                            AND
                                                            (!PPR PROP NIL)
                                                            THE VALUE TO BE REMOVED WAS NOT FOUND %.))
                                             (BINDINGS (QUOTE PROP)
                                                       PROP
                                                       (QUOTE ATM)
                                                       ATM)
                                             (QUOTE WARNING]
                              (COND [(NULL ATM)
                                     (SET PROP (REMOVE1 TEMP (EVALV PROP]
                                    (T (PUTPROP ATM PROP (REMOVE1 TEMP (GETPROP ATM PROP]
                     NIL])
```

(GENERATE-UNDO-TUPLE-PART

```
[LAMBDA (ALIST)                                                    (* kbr: "19-Oct-85 16:31")
  (LET (!ADDITIVE! !---ADDITIVE-LST---! !SINGLE-VARS!)
       [SETQ !ADDITIVE! (QUOTE (!ADDITIVE-TYPE! (CONS PROP (CONS ATM !VAL-NAME!]
       (SETQ !---ADDITIVE-LST---! (for X in ALIST when (EQ (CADR X)
                                                           (QUOTE ADDITIVE))
                                     collect (SUB-PAIR (QUOTE (!ADDITIVE-TYPE! !VAL-NAME!))
                                                       (LIST (CAR X)
                                                             (CADDDR X))
                                                       !ADDITIVE!)))
       (SETQ !SINGLE-VARS! (for X in ALIST when (AND (EQ (CADR X)
                                                         (QUOTE SINGLE))
                                                     (EQ (CADDR X)
                                                         (QUOTE VARIABLE)))
                              collect (CAR X)))
       (BQUOTE (SELECTQ PROP
                        (\,@ !---ADDITIVE-LST---!)
                        ((\, !SINGLE-VARS!)
                            PROP)
                        (DCELL (CONS (QUOTE DCELL)
```

```
                                ATM))
                    (CONS PROP ATM])

)

(RPAQQ IOCOMS ((* IO *)
               (FNS !CLAUSE-SET !CLAUSE EQUALITY-HYP-NO GET-SCHEMA-MEASURE-RELATION IO IO1
                    JUSTIFICATION-SENTENCE !LIST MAPRINEVAL NOTICE-CLAUSE PEVAL PEVAL-APPLY PEVALV PLURALP
                    !PPR-LIST !PPR PRIN5* PRINEVAL PRINEVAL1 PRINT-DEFN-MSG TH-IFY UN-NOTICE-CLAUSE)))


          (* * IO *)

(DEFINEQ
```

(!CLAUSE-SET

```
  [LAMBDA (CL-SET INDENT)                                          (* kbr: "19-Oct-85 16:31")
    (LET ((*INDENT* (OR INDENT *INDENT*)))
         (SETQ LAST-CLAUSE CL-SET)
         (PPRINDENT [COND
                        ((NULL CL-SET)
                         TRUE)
                        ((NULL (CDR CL-SET))
                         (PRETTYIFY-CLAUSE (CAR CL-SET)))
                        (T (CONS (QUOTE AND)
                                 (for CL in CL-SET collect (PRETTYIFY-CLAUSE CL]
               (COND
                  ((IEQP 0 *INDENT*)
                   0)
                  (T (ADD1 *INDENT*)))
               1 *FILE*)
         (SETQ LAST-PRINEVAL-CHAR NIL)
         NIL])
```

(!CLAUSE

```
  [LAMBDA (CL INDENT)                                              (* kbr: "19-Oct-85 16:31")
    (LET ((*INDENT* (OR INDENT *INDENT*)))
         (SETQ LAST-CLAUSE CL)
         (PPRINDENT (PRETTYIFY-CLAUSE CL)
                (COND
                   ((IEQP 0 *INDENT*)
                    0)
                   (T (ADD1 *INDENT*)))
                1 *FILE*)
         (SETQ LAST-PRINEVAL-CHAR NIL)
         NIL])
```

(EQUALITY-HYP-NO

```
  [LAMBDA (TERM CL)                                                (* kbr: "19-Oct-85 16:31")
    (LET (HYPS)
         [SETQ HYPS (for LIT in CL count (BM-MATCH LIT (NOT (EQUAL & &]
         (COND
            ((IEQP HYPS 1)
             NIL)
            (T (ADD1 (for LIT in CL until (EQUAL LIT TERM) count (BM-MATCH LIT (NOT (EQUAL & &])
```

(GET-SCHEMA-MEASURE-RELATION

```
  [LAMBDA (CANDIDATE CL-SET)                                       (* kbr: "19-Oct-85 16:31")

          (* Returns a list of three things. A schematic formula, using p applied to all the vars in CL-SET, showing the induction in
          CANDIDATE a measure term, indicating what decreases and the well-founded relation.
          *)

    (LET (TERM MEASURE-ARGS FORMALS SCHEMA MEASURE RELATION)
         (SETQ TERM (fetch (CANDIDATE INDUCTION-TERM) of CANDIDATE))
         [SETQ FORMALS (CADR (GETPROP (FFN-SYMB TERM)
                                      (QUOTE SDEFN]
         (SETQ MEASURE (fetch (JUSTIFICATION MEASURE-TERM) of (fetch (CANDIDATE JUSTIFICATION) of CANDIDATE)))
                                                                   (* We must instantiate the measure term with the actuals.
                                                                   *)
         (SETQ MEASURE-ARGS (ALL-VARS MEASURE))
         (SETQ MEASURE (COND
                          (MEASURE (SUB-PAIR-VAR-LST MEASURE-ARGS (FILTER-ARGS MEASURE-ARGS FORMALS (FARGS
                                                                                                     TERM))
                                                     MEASURE))
                          (T NIL)))
         (SETQ RELATION (fetch (JUSTIFICATION RELATION) of (fetch (CANDIDATE JUSTIFICATION) of CANDIDATE)))
         [SETQ SCHEMA
          (CONS (QUOTE AND)
                (for CL
                   in [IND-FORMULA (fetch (CANDIDATE TESTS-AND-ALISTS-LST) of CANDIDATE)
                              NIL
                              (LIST (LIST (CONS (QUOTE P)
                                                (REVERSE (ALL-VARS-LST (REVERSE (APPLY (FUNCTION APPEND)
```

```
                                                                                CL-SET]
                  collect (PRETTYIFY-CLAUSE CL]
        (LIST SCHEMA MEASURE RELATION])
```

(**IO**

```
  [LAMBDA (PROCESS PARENT PARENT-HIST DESCENDANTS HIST-ENTRY)    (* kbr: "19-Oct-85 16:31")
    (LET (TIME)
         (SETQ TIME (TIME-IN-60THS))
         (APPLY IO-FN NIL)
         (SETQ IOTHMTIME (PLUS IOTHMTIME (DIFFERENCE (TIME-IN-60THS)
                                                     TIME)))
         NIL])
```

(**IO1**

```
  [LAMBDA NIL                                                    (* kbr: "19-Oct-85 16:31")
    (PROG (SO-NEXT-CONSIDER ACCUMS CROSS DEFNS DIR ELIM-LEMMAS GEN-LEMMAS HIGH-CNT INDENT KEEP LEMMAS LST MASS
                MERGED-CAND-CNT N NAME NAMES OBVIOUS RAW-CAND-CNT SKOS TERM1 TERM2 TERMS WINNING-CAND WON-FLG
                VETO-CNT BROTHER-NO MAX MEASURE RELATION SCHEMA FAVORED-CNT HYP-NO FLG *NOPOINT)
          (SETQ *NOPOINT T)
          [SETQ SO-NEXT-CONSIDER (PQUOTE (PROGN (COND
                                                    ((EQ LAST-PROCESS (QUOTE POP))
                                                     %.))
                                                   // // // /#
                                                   (COND
                                                      ((NOT (EQ LAST-PROCESS (QUOTE STORE-SENT)))
                                                       (? (SO NEXT CONSIDER)
                                                          (SO LET US TURN OUR ATTENTION TO)
                                                          (SO WE NOW RETURN TO))
                                                       : // // (!CLAUSE-SET CL-SET INDENT)
                                                       (? (, // // NAMED)
                                                          (, // // WHICH WE NAMED)
                                                          (, // // WHICH IS FORMULA))
                                                       (!PPR (CAR HIST-ENTRY)
                                                             NIL)
                                                       ABOVE %.)
                                                      [(AND (IEQP (LENGTH CL-SET)
                                                                  1)
                                                            (EQ LAST-CLAUSE (CAR CL-SET]
                                                      (T SO NOW LET US (? (CONSIDER)
                                                                          (RETURN TO))
                                                         : // // (!CLAUSE-SET CL-SET NIL)
                                                         (? (, // // NAMED)
                                                            (, // // WHICH WE NAMED)
                                                            (%. // // WE NAMED THIS)
                                                            (%. // // WE GAVE THIS THE NAME))
                                                         (!PPR (CAR HIST-ENTRY)
                                                               NIL)
                                                         (? (ABOVE)
                                                            NIL)
                                                         %.]
          [COND
             ((EQ PROCESS (QUOTE SETUP))
              (COND
                 ((NOT (OPENP PROVE-FILE))
                  (SETQ PROVE-FILE NIL)))
              (SETQ CLAUSE-ALIST NIL)
              (SETQ LAST-PROCESS (QUOTE SETUP))
              (SETQ LAST-PRINEVAL-CHAR (QUOTE %.))
              (NOTICE-CLAUSE PARENT 0 (LIST NIL)))
             ((EQ PROCESS (QUOTE SETTLED-DOWN-CLAUSE))
              (RETURN NIL))
             ((EQ PROCESS (QUOTE INDUCT))
              (COND
                 [(AND (NOT LEFTMARGINCHAR)
                       (EQ PARENT LAST-CLAUSE))
                  (SETQ TEMP-TEMP (UN-NOTICE-CLAUSE LAST-CLAUSE))
                  (SETQ CLAUSE-ALIST NIL)
                  (COND
                     ((AND (FIXP (CADR TEMP-TEMP))
                           (LESSP (CADR TEMP-TEMP)
                                  16))
                      (NOTICE-CLAUSE LAST-CLAUSE (CADR TEMP-TEMP)
                             (LIST NIL)))
                     (T (NOTICE-CLAUSE (CAR TEMP-TEMP)
                                0
                                (LIST NIL]
                 (T (SETQ CLAUSE-ALIST NIL)
                    (NOTICE-CLAUSE PARENT 0 (LIST NIL]
          [SETQ TEMP-TEMP (COND
                             ([AND PARENT (NOT (MEMB PROCESS (QUOTE (POP SUBSUMED-ABOVE SUBSUMED-BY-PARENT
                                                                                SUBSUMED-BELOW]
                              (UN-NOTICE-CLAUSE PARENT))
                             (T (QUOTE (NIL 0 (NIL]
```

(* The BROTHER-NO of a clause is the case number for it. It is a list of numbers, to be printed in reverse order, separated by dots. If the CAR of the BROTHER-NO is NIL it means do not print it.

```
          *)

          (SETQ BROTHER-NO (OR (CADDR TEMP-TEMP)
                               (LIST NIL)))
          (SETQ INDENT (CADR TEMP-TEMP))
          (SETQ MAX (LENGTH DESCENDANTS))
          [for CL in DESCENDANTS as I from MAX by -1 do (NOTICE-CLAUSE CL (COND
                                                                           ((IEQP MAX 1)
                                                                            INDENT)
                                                                           (T (IPLUS TREE-INDENT INDENT)))
                                                                 (COND
                                                                    ((IEQP MAX 1)
                                                                     (COND
                                                                        ((CAR BROTHER-NO)
                                                                         (CONS NIL BROTHER-NO))
                                                                        (T BROTHER-NO)))
                                                                    ((CAR BROTHER-NO)
                                                                     (CONS I BROTHER-NO))
                                                                    (T (CONS I (CDR BROTHER-NO]
          [COND
             ((MEMB PROCESS EXECUTE-PROCESSES)
              (COND
                 ((EQ LAST-PROCESS (QUOTE SETUP))
                  (SETQ LAST-CLAUSE PARENT))
                 (T (ITERPRIN TREE-LINES PROVE-FILE)
                    (ISPACES (IDIFFERENCE INDENT TREE-INDENT)
                           PROVE-FILE)
                    [COND
                       ((AND (NOT (EQUAL INDENT 0))
                             (CAR BROTHER-NO))
                        (IPRINC "CASE" PROVE-FILE)
                        (for I in (REVERSE BROTHER-NO) do (IPRINC I PROVE-FILE)
                                                          (IPRINC "." PROVE-FILE]
                       (PRINEVAL (PQUOTE (!CLAUSE PARENT NIL))
                              (BINDINGS (QUOTE PARENT)
                                        PARENT)
                              (IPLUS 5 INDENT)
                              PROVE-FILE]
          (SELECTQ PROCESS
              (SIMPLIFY-CLAUSE
                   (SETQ FLG NIL)
                   (SETQ LEMMAS NIL)
                   (SETQ DEFNS NIL)
                   [for X in HIST-ENTRY do (COND
                                             ((LISTP X)
```

(* A LISTP entry here means that PROCESS-EQUATIONAL-POLYS added an equality to this clause.
The form of X in this case is ((FIND-EQUATIONAL-POLY lhs)), where lhs and rhs are the sides of the equation added.
In this case, ZERO is also a member of HIST-ENTRY and at the moment we will just ignore this opportunity to make the IO fancier. *)

```
                                              NIL)
                                             ((EQ X (QUOTE ZERO))
                                              (SETQ FLG T))
                                             ((GETPROP X (QUOTE TYPE-PRESCRIPTION-LST))
                                              (SETQ DEFNS (CONS X DEFNS)))
                                             (T (SETQ LEMMAS (CONS X LEMMAS]
                   (COND
                      ((AND (EQ LAST-PROCESS (QUOTE SETUP))
                            (IEQP (LENGTH DESCENDANTS)
                                  1)
                            (NOT LEMMAS)
                            [for X in DEFNS always (MEMB X (QUOTE (AND OR NOT IMPLIES]
                            (NOT FLG))                              (* pretend nothing happened in this case.
                                                                    *)
                       (RPLACA (CDR (ASSOC (CAR DESCENDANTS)
                                           CLAUSE-ALIST))
                               0)
                       (RETURN NIL))
                      (T (PRINEVAL (PQUOTE (PROGN [COND
                                                    ((EQ LAST-PROCESS (QUOTE SETUP))
                                                     THIS
                                                     (? (FORMULA)
                                                        (CONJECTURE)
                                                        NIL)
                                                     SIMPLIFIES)
                                                    (T (? (, // // WHICH (? (WE (@ FURTHER?)
                                                                                 SIMPLIFY)
                                                                            ((@ FURTHER?)
                                                                             SIMPLIFIES)))
                                                          (%. // // [COND
                                                                      ((EQ HIST-ENTRY NIL)
                                                                       (? NIL (OF COURSE ,)))
                                                                      (PARENT-HIST (? (BUT)
                                                                                      NIL
                                                                                      (HOWEVER]
                                                                  THIS
```

```
                                                (? (SIMPLIFIES (@ FURTHER?))
                                                   ((@ FURTHER?)
                                                    SIMPLIFIES]
                                             [COND
                                                (FLG , USING LINEAR ARITHMETIC (COND
                                                                                  ((AND (NOT LEMMAS)
                                                                                        (NOT DEFNS))
                                                                                   ,]
                                             (COND
                                                (LEMMAS (COND
                                                           ((AND FLG (NOT DEFNS)) AND)
                                                           (T ,))
                                                        (? ((? (APPEALING TO)
                                                               (APPLYING)
                                                               (REWRITING WITH))
                                                            THE
                                                            (PLURAL? LEMMAS LEMMAS LEMMA))
                                                           (APPLYING)
                                                           (REWRITING WITH))
                                                        (!LIST LEMMAS)
                                                        ,))
                                             (COND
                                                (DEFNS (COND
                                                          ((OR FLG LEMMAS) AND)
                                                          (T ,))
                                                       (? (OPENING UP)
                                                          (EXPANDING)
                                                          (UNFOLDING))
                                                       (? (THE (PLURAL? DEFNS FUNCTIONS FUNCTION))
                                                          (THE (PLURAL? DEFNS DEFINITIONS DEFINITION)
                                                               OF)
                                                          NIL)
                                                       (!LIST DEFNS)
                                                       ,))
                                             (COND
                                                ((AND (NOT FLG)
                                                      (EQ LEMMAS NIL)
                                                      (EQ DEFNS NIL))
                                                 ,
                                                 (? (TRIVIALLY)
                                                    (CLEARLY)
                                                    (OBVIOUSLY))
                                                 ,))
                                             TO))
                                  (BINDINGS (QUOTE DEFNS)
                                         DEFNS
                                         (QUOTE LEMMAS)
                                         LEMMAS
                                         (QUOTE PARENT-HIST)
                                         PARENT-HIST
                                         (QUOTE HIST-ENTRY)
                                         HIST-ENTRY
                                         (QUOTE FURTHER?)
                                         (COND
                                            ((AND (NOT DESCENDANTS)
                                                  (IGREATERP (LENGTH PARENT-HIST)
                                                             5))
                                             (PQUOTE FINALLY))
                                            ((EQ (CAAR PARENT-HIST)
                                                 (QUOTE SIMPLIFY-CLAUSE))
                                             (PQUOTE AGAIN))
                                            ((ASSOC (QUOTE SIMPLIFY-CLAUSE)
                                                    PARENT-HIST)
                                             (PQUOTE FURTHER))
                                            (T NIL))
                                         (QUOTE LAST-PROCESS)
                                         LAST-PROCESS
                                         (QUOTE FLG)
                                         FLG)
                                  INDENT PROVE-FILE))))
(FERTILIZE-CLAUSE
     (BM-MATCH HIST-ENTRY (LIST MASS CROSS DIR TERM1 TERM2 KEEP))
     (SETQ HYP-NO (EQUALITY-HYP-NO (LIST (QUOTE NOT)
                                         (LIST (QUOTE EQUAL)
                                               TERM1 TERM2))
                                PARENT))
     (OR (EQ DIR (QUOTE LEFT-FOR-RIGHT))
         (swap TERM1 TERM2))
     (PRINEVAL (PQUOTE (PROGN %. // // WE (? NIL NIL (NOW))
                              USE THE (COND
                                         (HYP-NO (@ N))
                                         (T ABOVE))
                              EQUALITY HYPOTHESIS (COND
                                                     ((OR MASS (NOT CROSS))
                                                      BY SUBSTITUTING)
                                                     (T BY CROSS-FERTILIZING))
                              (!PPR TERM1 NIL)
```

```
                                        FOR
                                        (!PPR TERM2 NIL)
                                        (COND
                                           (KEEP AND KEEPING THE EQUALITY HYPOTHESIS)
                                           (T AND THROWING AWAY THE EQUALITY))
                                        %.))
                     (BINDINGS (QUOTE KEEP)
                               KEEP
                               (QUOTE TERM2)
                               TERM2
                               (QUOTE TERM1)
                               TERM1
                               (QUOTE CROSS)
                               CROSS
                               (QUOTE MASS)
                               MASS
                               (QUOTE N)
                               (TH-IFY HYP-NO)
                               (QUOTE HYP-NO)
                               HYP-NO)
                    INDENT PROVE-FILE))
          (ELIMINATE-DESTRUCTORS-CLAUSE
               (SETQ ELIM-LEMMAS NIL)
               (SETQ GEN-LEMMAS NIL)
               (for X in HIST-ENTRY
                  do (SETQ ELIM-LEMMAS (ADD-TO-SET (CAR X)
                                                   ELIM-LEMMAS))
                     (SETQ LST (CONS [LIST (QUOTE PROGN)
                                           (LIST (QUOTE !PPR)
                                                 (KWOTE (CAR (CDDDDR X)))
                                                 NIL)
                                           (PQUOTE BY)
                                           (LIST (QUOTE !PPR)
                                                 (KWOTE (CADR (CDDDDR X)))
                                                 NIL)
                                           (PQUOTE (PROGN TO ELIMINATE))
                                           (LIST (QUOTE !LIST)
                                                 (KWOTE (for D in (CADR X) collect (LIST (QUOTE !PPR)
                                                                                         (KWOTE D)
                                                                                         NIL]
                                     LST))
                     [COND
                        ((CADDR X)
                         (SETQ GEN-LEMMAS
                          (UNION-EQUAL (for TERM in (CADDR X) bind LOOP-ANS
                                        do (SETQ LOOP-ANS
                                                 (ADD-TO-SET (LIST (QUOTE PROGN)
                                                                   (PQUOTE (PROGN THE TYPE RESTRICTION LEMMA
                                                                                  NOTED WHEN))
                                                                   (FN-SYMB (ARGN TERM 1))
                                                                   (PQUOTE (PROGN WAS INTRODUCED)))
                                                             LOOP-ANS))
                                        finally (RETURN LOOP-ANS))
                                       GEN-LEMMAS]
                     (SETQ GEN-LEMMAS (UNION-EQUAL (CADDDR X)
                                                   GEN-LEMMAS)))
               (PRINEVAL [PQUOTE (PROGN %. // // (? (APPLYING)
                                                    (APPEALING TO))
                                        THE
                                        (PLURAL? ELIM-LEMMAS LEMMAS LEMMA)
                                        (!LIST ELIM-LEMMAS)
                                        ,
                                        (? (WE NOW)
                                           NIL)
                                        REPLACE
                                        (!LIST LST)
                                        %.
                                        (COND
                                           (GEN-LEMMAS WE (? (USE)
                                                             (RELY UPON)
                                                             (EMPLOY))
                                                  (!LIST GEN-LEMMAS)
                                                  TO
                                                  (? (CONSTRAIN)
                                                     (RESTRICT))
                                                  THE NEW (COND
                                                             ([OR (CDR ELIM-LEMMAS)
                                                                  (CDR (CAR (CDR (CAR HIST-ENTRY]
                                                              VARIABLES)
                                                             (T VARIABLE))
                                                  %.]
                     (BINDINGS (QUOTE HIST-ENTRY)
                               HIST-ENTRY
                               (QUOTE ELIM-LEMMAS)
                               ELIM-LEMMAS
                               (QUOTE GEN-LEMMAS)
                               GEN-LEMMAS
```

```
                        (QUOTE LST)
                        LST)
                INDENT PROVE-FILE))
        (GENERALIZE-CLAUSE
            (BM-MATCH HIST-ENTRY (LIST SKOS TERMS OBVIOUS LEMMAS))
            [SETQ LST (for TERM in TERMS as VAR in SKOS collect (LIST (QUOTE PROGN)
                                                                      (LIST (QUOTE !PPR)
                                                                            (KWOTE TERM)
                                                                            NIL)
                                                                      (PQUOTE BY)
                                                                      (LIST (QUOTE !PPR)
                                                                            (KWOTE VAR)
                                                                            NIL]
            [COND
               (OBVIOUS (SETQ LEMMAS
                              (UNION-EQUAL (for TERM in OBVIOUS bind LOOP-ANS
                                                do (SETQ LOOP-ANS
                                                         (ADD-TO-SET (LIST (QUOTE PROGN)
                                                                           (PQUOTE (PROGN THE TYPE RESTRICTION
                                                                                          LEMMA NOTED WHEN))
                                                                           (FN-SYMB (ARGN TERM 1))
                                                                           (PQUOTE (PROGN WAS INTRODUCED)))
                                                                     LOOP-ANS))
                                                finally (RETURN LOOP-ANS))
                                           LEMMAS]
            (PRINEVAL [PQUOTE (PROGN (? (, // // WHICH WE GENERALIZE BY)
                                        (%. // // WE WILL TRY TO PROVE THE ABOVE (? (FORMULA)
                                                                                    (CONJECTURE))
                                            BY GENERALIZING IT ,))
                                     REPLACING
                                     (!LIST LST)
                                     %.
                                     (COND
                                        (LEMMAS WE RESTRICT THE NEW (PLURAL? SKOS VARIABLES VARIABLE)
                                               BY
                                               (? (APPEALING TO)
                                                  (RECALLING))
                                               (!LIST LEMMAS)
                                               %.]
                    (BINDINGS (QUOTE LEMMAS)
                              LEMMAS
                              (QUOTE SKOS)
                              SKOS
                              (QUOTE LST)
                              LST)
                    INDENT PROVE-FILE))
        (ELIMINATE-IRRELEVANCE-CLAUSE
            (PRINEVAL (PQUOTE (? (, // // WHICH HAS (PLURAL? N (@ N)
                                                       AN)
                                    IRRELEVANT
                                    (PLURAL? N TERMS TERM)
                                    IN IT %. BY ELIMINATING (PLURAL? N (PROGN THESE TERMS)
                                                                       (PROGN THE TERM))
                                    WE GETPROP)
                                 (%. // // ELIMINATE (PLURAL? N NIL THE)
                                     IRRELEVANT
                                     (PLURAL? N TERMS TERM)
                                     %.)))
                    [BINDINGS (QUOTE N)
                              (IDIFFERENCE (LENGTH PARENT)
                                           (LENGTH (CAR DESCENDANTS]
                    INDENT PROVE-FILE))
        (STORE-SENT (NOTICE-CLAUSE PARENT INDENT BROTHER-NO)
                    [COND
                       ((AND PARENT (EQ LAST-PROCESS (QUOTE SETUP))
                             (CADR HIST-ENTRY))
                        (SETQ LAST-CLAUSE (CADR HIST-ENTRY))
                        (NOTICE-CLAUSE LAST-CLAUSE 0 (LIST NIL]
                    (PRINEVAL [PQUOTE (COND
                                         ((EQ PARENT NIL)
                                          (? (, // // WHICH MEANS THE PROOF ATTEMPT HAS)
                                             (%. // // WHY SAY MORE ?)
                                             (%. // // NEED WE GO ON ?)))
                                         ((EQ LAST-PROCESS (QUOTE SETUP))
                                          // /# (? (GIVE THE CONJECTURE THE NAME)
                                                 (NAME THE CONJECTURE)
                                                 (CALL THE CONJECTURE))
                                          (!PPR (CAR HIST-ENTRY)
                                                NIL)
                                          %.)
                                         ((EQ (CAR (CDR HIST-ENTRY))
                                              (QUOTE QUIT))
                                          , // // WHICH WE WOULD NORMALLY TRY TO PROVE BY INDUCTION %.
                                          BUT SINCE THE DO-NOT-USE-INDUCTION-FLG HAS BEEN SET BY THE
                                          USER , THE PROOF ATTEMPT HAS)
                                         ((CAR (CDR HIST-ENTRY))
                                          , // // WHICH WE WOULD (? (NORMALLY)
```

```
                                                                (USUALLY))
                                          PUSH AND WORK ON LATER BY INDUCTION %. BUT IF WE MUST USE
                                          INDUCTION TO PROVE THE INPUT CONJECTURE , WE PREFER TO INDUCT
                                          ON THE ORIGINAL FORMULATION OF THE PROBLEM %. THUS WE WILL
                                          DISREGARD ALL THAT WE HAVE PREVIOUSLY DONE , GIVE THE NAME
                                          (!PPR (CAR HIST-ENTRY)
                                                NIL)
                                          TO THE ORIGINAL INPUT , AND WORK ON IT %.)
                                         (T (? (, // // WHICH WE WILL (@ FINALLY?)
                                                  NAME
                                                  (!PPR (CAR HIST-ENTRY)
                                                        NIL)
                                                  %.)
                                               (%. // // (@ FINALLY?)
                                                   (? (GIVE THE ABOVE FORMULA THE NAME)
                                                      (NAME THE ABOVE SUBGOAL)
                                                      (CALL THE ABOVE CONJECTURE))
                                                   (!PPR (CAR HIST-ENTRY)
                                                         NIL)
                                                   %.]
                              (BINDINGS (QUOTE HIST-ENTRY)
                                        HIST-ENTRY
                                        (QUOTE LAST-PROCESS)
                                        LAST-PROCESS
                                        (QUOTE PARENT)
                                        PARENT
                                        (QUOTE FINALLY?)
                                        (COND
                                           ((IGREATERP (LENGTH PARENT-HIST)
                                                       5)
                                            (PQUOTE FINALLY))
                                           (T NIL)))
                                        INDENT PROVE-FILE))
(POP (PRINEVAL (PQUOTE (PROGN (COND
                                  ((EQ LAST-PROCESS (QUOTE POP))
                                   , WHICH (? (, IN TURN ,)
                                              (, CONSEQUENTLY ,)
                                              NIL)
                                   (? (ALSO)
                                      NIL))
                                  (T // // // /# THAT))
                               FINISHES THE PROOF OF (!PPR (CAR HIST-ENTRY)
                                                           NIL)))
                    (BINDINGS (QUOTE HIST-ENTRY)
                              HIST-ENTRY
                              (QUOTE LAST-PROCESS)
                              LAST-PROCESS)
                    0 PROVE-FILE))
(SUBSUMED-ABOVE
     (PRINEVAL (PQUOTE (PROGN (@ SO-NEXT-CONSIDER)
                              (? (HA !)
                                 (HOW NICE !)
                                 NIL NIL NIL)
                              THIS
                              (? (CONJECTURE)
                                 (FORMULA)
                                 (GOAL)
                                 NIL)
                              IS SUBSUMED BY THE (? ((? (LEMMA)
                                                        (THEOREM)
                                                        (GOAL))
                                                    WE NAMED (!PPR (CAR (CDR HIST-ENTRY))
                                                                   NIL)
                                                       AND PROVED ABOVE)
                                                    (PREVIOUSLY PROVED (!PPR (CAR (CDR HIST-ENTRY
                                                                                       ))
                                                                             NIL))
                                                    ((!PPR (CAR (CDR HIST-ENTRY))
                                                           NIL)
                                                     , WHICH WAS PROVED ABOVE))
                                 !))
                    (BINDINGS (QUOTE HIST-ENTRY)
                              HIST-ENTRY
                              (QUOTE SO-NEXT-CONSIDER)
                              SO-NEXT-CONSIDER
                              (QUOTE LAST-PROCESS)
                              LAST-PROCESS
                              (QUOTE CL-SET)
                              PARENT
                              (QUOTE LAST-CLAUSE)
                              LAST-CLAUSE
                              (QUOTE INDENT)
                              5)
                    0 PROVE-FILE))
(SUBSUMED-BY-PARENT
     (PRINEVAL (PQUOTE (PROGN (@ SO-NEXT-CONSIDER)
                              (? (OH NO !)
```

```
                                        (OOPS !)
                                        NIL)
                                     THIS FORMULA IS SUBSUMED BY ITS PARENT , (!PPR (CAR (CDR HIST-ENTRY
                                                                                         ))
                                                                                  NIL)
                                  ! (? (THAT MEANS WE WOULD LOOP IF WE TRIED TO PROVE IT BY INDUCTION %.)
                                     NIL NIL)))
                (BINDINGS (QUOTE HIST-ENTRY)
                       HIST-ENTRY
                       (QUOTE SO-NEXT-CONSIDER)
                       SO-NEXT-CONSIDER
                       (QUOTE LAST-PROCESS)
                       LAST-PROCESS
                       (QUOTE CL-SET)
                       PARENT
                       (QUOTE LAST-CLAUSE)
                       LAST-CLAUSE
                       (QUOTE INDENT)
                       5)
                0 PROVE-FILE))
        (SUBSUMED-BELOW
            (PRINEVAL (PQUOTE (PROGN (@ SO-NEXT-CONSIDER)
                                     (? (AH HA !)
                                        (WHAT LUCK !)
                                        (YOU PROBABLY DID NOT NOTICE , BUT)
                                        (BUT)
                                        NIL)
                                     THIS CONJECTURE IS SUBSUMED BY
                                     (? (ANOTHER SUBGOAL AWAITING OUR ATTENTION , NAMELY)
                                        (THE SUBGOAL WE NAMED)
                                        (FORMULA))
                                     (!PPR (CAR (CDR HIST-ENTRY))
                                           NIL)
                                     ABOVE %.))
                (BINDINGS (QUOTE HIST-ENTRY)
                       HIST-ENTRY
                       (QUOTE SO-NEXT-CONSIDER)
                       SO-NEXT-CONSIDER
                       (QUOTE LAST-PROCESS)
                       LAST-PROCESS
                       (QUOTE CL-SET)
                       PARENT
                       (QUOTE LAST-CLAUSE)
                       LAST-CLAUSE
                       (QUOTE INDENT)
                       5)
                0 PROVE-FILE))
        (INDUCT (BM-MATCH HIST-ENTRY (LIST NAME WINNING-CAND RAW-CAND-CNT MERGED-CAND-CNT VETO-CNT
                                           HIGH-CNT FAVORED-CNT))
                (COND
                   (WINNING-CAND
                    (SETQ FLG NIL)
                    (SETQ LEMMAS NIL)
                    (SETQ DEFNS NIL)
                    [for X in (fetch (JUSTIFICATION LEMMAS) of (fetch (CANDIDATE JUSTIFICATION) of
                                                                                          WINNING-CAND
                                                                                          ))
                       do (COND
                              ((EQ X (QUOTE ZERO))
                               (SETQ FLG T))
                              ((GETPROP X (QUOTE TYPE-PRESCRIPTION-LST))
                               (SETQ DEFNS (CONS X DEFNS)))
                              (T (SETQ LEMMAS (CONS X LEMMAS]
                    (BM-MATCH (GET-SCHEMA-MEASURE-RELATION WINNING-CAND PARENT)
                              (LIST SCHEMA MEASURE RELATION))
                    (SETQ ACCUMS (SET-DIFF (fetch (CANDIDATE CHANGED-VARS) of WINNING-CAND)
                                           (ALL-VARS MEASURE)))
                    (PRINEVAL [PQUOTE (PROGN (@ SO-NEXT-CONSIDER)
                                             (? (WE WILL TRY TO PROVE IT BY INDUCTION %.)
                                                (PERHAPS WE CAN PROVE IT BY INDUCTION %.)
                                                (LET US APPEAL TO THE INDUCTION PRINCIPLE %.)
                                                (WE WILL APPEAL TO INDUCTION %.))
                                             (COND
                                                [(NOT (IEQP RAW-CAND-CNT 1))
                                                 (? (THERE ARE (@ RAW-CAND-CNT)
                                                               PLAUSIBLE INDUCTIONS)
                                                    ((@ RAW-CAND-CNT)
                                                     INDUCTIONS ARE SUGGESTED BY TERMS IN THE CONJECTURE
                                                     )
                                                    (THE RECURSIVE TERMS IN THE CONJECTURE SUGGEST
                                                         (@ RAW-CAND-CNT)
                                                         INDUCTIONS))
                                                 (COND
                                                    ((IEQP RAW-CAND-CNT MERGED-CAND-CNT))
                                                    ((IEQP MERGED-CAND-CNT 1)
                                                     %. HOWEVER , THEY MERGE INTO ONE LIKELY CANDIDATE
                                                     INDUCTION %.)
```

```
                                    (T %. THEY MERGE INTO (@ MERGED-CAND-CNT)
                                       LIKELY CANDIDATE INDUCTIONS))
                                 (COND
                                    ((NOT (IEQP MERGED-CAND-CNT 1))
                                     (COND
                                        ((IEQP VETO-CNT 0)
                                         ,
                                         (COND
                                            ((IEQP MERGED-CAND-CNT 2)
                                             BOTH)
                                            (T ALL))
                                         OF WHICH ARE FLAWED %.)
                                        ((IEQP VETO-CNT MERGED-CAND-CNT)
                                         ,
                                         (COND
                                            ((IEQP VETO-CNT 2)
                                             BOTH)
                                            (T ALL))
                                         OF WHICH ARE UNFLAWED %.)
                                        ((IEQP VETO-CNT 1)
                                         %. HOWEVER , ONLY ONE IS UNFLAWED %.)
                                        (T , (@ VETO-CNT)
                                           OF WHICH ARE UNFLAWED %.))
                                     (COND
                                        ((NOT (IEQP VETO-CNT 1))
                                         (COND
                                            ((IEQP FAVORED-CNT 1)
                                             SO WE WILL CHOOSE THE ONE SUGGESTED BY THE
                                             LARGEST NUMBER OF NONPRIMITIVE RECURSIVE
                                             FUNCTIONS %.)
                                            (T (COND
                                                  ((NOT (IEQP FAVORED-CNT VETO-CNT))
                                                   WE LIMIT OUR CONSIDERATION TO THE
                                                   (@ FAVORED-CNT)
                                                   SUGGESTED BY THE LARGEST NUMBER OF
                                                   NONPRIMITIVE RECURSIVE FUNCTIONS IN
                                                   THE CONJECTURE %.))
                                               (COND
                                                  ((IEQP HIGH-CNT 1)
                                                   HOWEVER , ONE OF THESE IS MORE LIKELY
                                                   THAN THE (COND
                                                               ((IEQP FAVORED-CNT 2)
                                                                OTHER)
                                                               (T OTHERS))
                                                   %.)
                                                  (T SINCE (COND
                                                              ((IEQP HIGH-CNT
                                                                     FAVORED-CNT)
                                                               (COND
                                                                  ((IEQP HIGH-CNT 2)
                                                                   BOTH)
                                                                  (T ALL)))
                                                              (T (@ HIGH-CNT)))
                                                     OF THESE ARE EQUALLY LIKELY , WE
                                                     WILL CHOOSE ARBITRARILY %.]
                                    (T THERE IS ONLY ONE (? (PLAUSIBLE)
                                                            (SUGGESTED))
                                       INDUCTION %.))
                                 WE WILL INDUCT ACCORDING TO THE FOLLOWING SCHEME
                                 (!PPR SCHEMA (PQUOTE %.))
                                 (COND
                                    (MEASURE (@ JUSTIFICATION-SENTENCE)
                                           (PLURAL? TESTS-AND-ALISTS-LST EACH THE)
                                           INDUCTION STEP OF THE SCHEME %.
                                           (COND
                                              (ACCUMS NOTE , HOWEVER , THE INDUCTIVE
                                                     (COND
                                                        (INSTANCES? INSTANCES)
                                                        (T INSTANCE))
                                                    CHOSEN FOR (!PPR-LIST ACCUMS)
                                                    %.))
                                           THE ABOVE INDUCTION SCHEME (? (PRODUCES)
                                                                         (GENERATES)
                                                                         (LEADS TO)))
                                    (T THIS SCHEME IS JUSTIFIED BY THE ASSUMPTION THAT
                                       (!PPR (FN-SYMB TERM)
                                             NIL)
                                       IS TOTAL %.]
                           (BINDINGS (QUOTE ACCUMS)
                                  ACCUMS
                                  (QUOTE JUSTIFICATION-SENTENCE)
                                  (JUSTIFICATION-SENTENCE)
                                  (QUOTE RELATION)
                                  RELATION
                                  (QUOTE MEASURE)
                                  MEASURE
                                  (QUOTE LEMMAS)
```

```
                                    LEMMAS
                                    (QUOTE DEFNS)
                                    DEFNS
                                    (QUOTE FLG)
                                    FLG
                                    (QUOTE NUMBER)
                                    (LENGTH (fetch (JUSTIFICATION LEMMAS) of (fetch (CANDIDATE JUSTIFICATION)
                                                                                    of WINNING-CAND)))
                                    (QUOTE SCHEMA)
                                    SCHEMA
                                    (QUOTE FAVORED-CNT)
                                    FAVORED-CNT
                                    (QUOTE HIGH-CNT)
                                    HIGH-CNT
                                    (QUOTE MERGED-CAND-CNT)
                                    MERGED-CAND-CNT
                                    (QUOTE VETO-CNT)
                                    VETO-CNT
                                    (QUOTE RAW-CAND-CNT)
                                    RAW-CAND-CNT
                                    (QUOTE SO-NEXT-CONSIDER)
                                    SO-NEXT-CONSIDER
                                    (QUOTE TESTS-AND-ALISTS-LST)
                                    (fetch (CANDIDATE TESTS-AND-ALISTS-LST) of WINNING-CAND)
                                    (QUOTE INSTANCES?)
                                    [OR (CDR ACCUMS)
                                        (NOT (IEQP 1 (for TA in (fetch (CANDIDATE TESTS-AND-ALISTS-LST)
                                                                       of WINNING-CAND)
                                                                sum (LENGTH (fetch (TESTS-AND-ALISTS ALISTS)
                                                                                   of TA]
                                    (QUOTE TERM)
                                    (fetch (CANDIDATE INDUCTION-TERM) of WINNING-CAND)
                                    (QUOTE LAST-PROCESS)
                                    LAST-PROCESS
                                    (QUOTE CL-SET)
                                    PARENT
                                    (QUOTE LAST-CLAUSE)
                                    LAST-CLAUSE
                                    (QUOTE HIST-ENTRY)
                                    HIST-ENTRY
                                    (QUOTE INDENT)
                                    (IPLUS 5 INDENT))
                             INDENT PROVE-FILE))
                (T (PRINEVAL (PQUOTE (PROGN (@ SO-NEXT-CONSIDER)
                                            SINCE THERE IS NOTHING TO INDUCT UPON , THE PROOF HAS))
                             (BINDINGS (QUOTE SO-NEXT-CONSIDER)
                                    SO-NEXT-CONSIDER
                                    (QUOTE LAST-PROCESS)
                                    LAST-PROCESS
                                    (QUOTE CL-SET)
                                    PARENT
                                    (QUOTE LAST-CLAUSE)
                                    LAST-CLAUSE
                                    (QUOTE HIST-ENTRY)
                                    HIST-ENTRY
                                    (QUOTE INDENT)
                                    5)
                             0 PROVE-FILE))))
(SETUP (COND
          ([AND (IEQP (LENGTH DESCENDANTS)
                      1)
                (for X in HIST-ENTRY always (MEMB X (QUOTE (AND OR NOT IMPLIES]
           NIL)
          (T (PRINEVAL (PQUOTE (PROGN THIS (? (FORMULA)
                                              (CONJECTURE))
                                       CAN BE (COND
                                                 (HIST-ENTRY SIMPLIFIED , USING THE
                                                        (PLURAL? HIST-ENTRY ABBREVIATIONS
                                                                 ABBREVIATION)
                                                        (!LIST HIST-ENTRY)
                                                        ,)
                                                 (T PROPOSITIONALLY SIMPLIFIED))
                                       TO))
                       (BINDINGS (QUOTE HIST-ENTRY)
                              HIST-ENTRY)
                       INDENT PROVE-FILE))))
(FINISHED (BM-MATCH HIST-ENTRY (LIST WON-FLG))
          (PRINEVAL (PQUOTE (PROGN [COND
                                       [(EQ LAST-PROCESS (QUOTE POP))
                                        %.
                                        (COND
                                           (WON-FLG Q.E.D.)
                                           (T // // (@ FAILURE-MSG]
                                       (T // // (COND
                                                   ((EQ WON-FLG (QUOTE DEFN-OK)))
                                                   (WON-FLG Q.E.D.)
                                                   (T (@ FAILURE-MSG]
```

```
                                                    // //))
                                  (BINDINGS (QUOTE FAILURE-MSG)
                                        FAILURE-MSG
                                        (QUOTE WON-FLG)
                                        WON-FLG
                                        (QUOTE LAST-PROCESS)
                                        LAST-PROCESS)
                                  0 PROVE-FILE))
            (ERROR1 (PQUOTE (PROGN IO1 HAS BEEN GIVEN AN UNRECOGNIZED PROCESS NAMED (!PPR PROCESS NIL)
                                   %.))
                    (BINDINGS (QUOTE PROCESS)
                          PROCESS)
                    (QUOTE HARD)))
         [COND
            ([NOT (OR (MEMB PROCESS UN-PRODUCTIVE-PROCESSES)
                      (AND (EQ PROCESS (QUOTE INDUCT))
                           (NOT (CADR HIST-ENTRY)))
                      (AND (EQ PROCESS (QUOTE SETUP))
                           (IEQP (LENGTH DESCENDANTS)
                                 1)
                           (for X in HIST-ENTRY always (MEMB X (QUOTE (AND OR NOT IMPLIES]
             (SETQ N (LENGTH DESCENDANTS))
             (COND
                ((EQ LAST-PRINEVAL-CHAR (QUOTE %.))
                 (PRINEVAL (PQUOTE (? (WE THUS OBTAIN)
                                      (THE RESULT IS)
                                      (THIS PRODUCES)
                                      (THIS GENERATES)
                                      (WE WOULD THUS LIKE TO PROVE)
                                      (WE MUST THUS PROVE)))
                           (BINDINGS)
                           INDENT PROVE-FILE)))
             (COND
                ((NEQ LAST-PRINEVAL-CHAR (QUOTE :))
                 (PRINEVAL (PQUOTE (PROGN [COND
                                          ((EQUAL N 0)
                                           NIL)
                                          ((EQUAL N 1)
                                           (? (THE (? (NEW)
                                                      NIL)
                                                   (? (GOAL)
                                                      (CONJECTURE)
                                                      (FORMULA)))
                                              NIL NIL))
                                          (T (? ((@ N)
                                                 NEW
                                                 (? (GOALS)
                                                    (CONJECTURES)
                                                    (FORMULAS)))
                                                (THE FOLLOWING (@ N)
                                                     NEW
                                                     (? (GOALS)
                                                        (CONJECTURES)
                                                        (FORMULAS]
                                        :))
                           (BINDINGS (QUOTE N)
                                 N)
                           INDENT PROVE-FILE]
         (COND
            ((AND (NOT (MEMB PROCESS UN-PRODUCTIVE-PROCESSES))
                  (NOT DESCENDANTS))
             (ITERPRIN TREE-LINES PROVE-FILE)
             (PRINEVAL (PQUOTE (PROGN T %.))
                       (BINDINGS)
                       (IPLUS 6 INDENT)
                       PROVE-FILE)))
         (SETQ LAST-PROCESS (COND
                               ([AND (EQ PROCESS (QUOTE SETUP))
                                     (OR (NOT (IEQP (LENGTH DESCENDANTS)
                                                    1))
                                         (NOT (for X in HIST-ENTRY
                                                 always (MEMB X (QUOTE (AND OR NOT IMPLIES]
                                (QUOTE SETUP-AND-CLAUSIFY-INPUT))
                               (T PROCESS)))
         (RETURN NIL])
```

(JUSTIFICATION-SENTENCE

```
[LAMBDA NIL                                                    (* kbr: "19-Oct-85 16:31")
```

(* This fn returns a sentence to be fed to PRINEVAL. The BINDINGS must include FLG, LEMMAS, DEFNS, NUMBER, MEASURE, and RELATION. FLG is T or NIL indicating that linear arithmetic was used.
LEMMAS and DEFNS are the list of lemmas and definitions used.
NUMBER is the length of LEMMAS plus that of DEFNS plus 1 or 0 according to FLG.
MEASURE is a term and RELATION is a fn name. *)

```
   (PQUOTE (PROGN [COND
```

```
                     (FLG LINEAR ARITHMETIC (COND
                                              ((AND LEMMAS DEFNS)
                                               ,)
                                              ((OR LEMMAS DEFNS) AND]
                  [COND
                     (LEMMAS THE (PLURAL? LEMMAS LEMMAS LEMMA)
                             (!LIST LEMMAS)
                             (COND
                                ((AND FLG DEFNS)
                                 , AND)
                                (DEFNS AND]
                  (COND
                     (DEFNS THE (PLURAL? DEFNS DEFINITIONS DEFINITION)
                            OF
                            (!LIST DEFNS)))
                  [COND
                     ((OR FLG LEMMAS DEFNS)
                      [PLURAL? NUMBER [? (INFORM US)
                                         (ESTABLISH)
                                         (CAN BE USED TO (? (PROVE)
                                                            (SHOW)
                                                            (ESTABLISH]
                               (? (ESTABLISHES)
                                  (INFORMS US)
                                  (CAN BE USED TO (? (PROVE)
                                                     (SHOW)
                                                     (ESTABLISH]
                      THAT)
                     (T (? (IT IS OBVIOUS THAT)
                           (OBVIOUSLY)
                           (CLEARLY]
                  THE MEASURE (!PPR MEASURE NIL)
                  DECREASES ACCORDING TO THE WELL-FOUNDED RELATION (!PPR RELATION NIL)
                  IN])
```

(!LIST

```
[LAMBDA (*LST*)                                                    (* kbr: "19-Oct-85 16:31")
  (MAPRINEVAL *LST* *INDENT* *FILE* NIL NIL (PQUOTE ,)
         (COND
            ((CDDR *LST*)
             (PQUOTE (PROGN , AND)))
            (T (PQUOTE AND])
```

(MAPRINEVAL

```
[LAMBDA (*LST* *INDENT* *FILE* *LEFT* *RIGHT* *SEPR* *FINALSEPR*)
                                                                   (* kbr: "19-Oct-85 16:31")
  (AND *LEFT* (PRINEVAL1 *LEFT*))
  [COND
     ((LISTP *LST*)
      (COND
         [(CDR *LST*)
          (for TAIL on *LST* do (PRINEVAL1 (CAR TAIL))
                               (COND
                                  ((NULL (CDR TAIL))
                                   NIL)
                                  ((NULL (CDDR TAIL))
                                   (AND *FINALSEPR* (PRINEVAL1 *FINALSEPR*)))
                                  (T (AND *FINALSEPR* (PRINEVAL1 *SEPR*]
         (T (PRINEVAL1 (CAR *LST*]
  (AND *RIGHT* (PRINEVAL1 *RIGHT*])
```

(NOTICE-CLAUSE

```
[LAMBDA (CL COL BROTHER-NO)                                        (* kbr: "19-Oct-85 16:31")
  (CAR (SETQ CLAUSE-ALIST (CONS (LIST CL (OR COL 0)
                                      BROTHER-NO)
                                CLAUSE-ALIST])
```

(PEVAL

```
[LAMBDA (FORM)                                                     (* kbr: "19-Oct-85 16:31")
  (COND
     [(NLISTP FORM)
      (COND
         [(LITATOM FORM)
          (COND
             ((OR (EQ FORM NIL)
                  (EQ FORM T))
              FORM)
             (T (PEVALV FORM]
         ((NUMBERP FORM)
          FORM)
         (T (ERROR1 (PQUOTE (PROGN ILLEGAL PEVAL FORM , (!PPR TERM NIL)
                                   %.))
                    (BINDINGS (QUOTE TERM)
```

```
                          FORM)
                    (QUOTE HARD]
      ((OR (EQ (CAR FORM)
               (QUOTE PQUOTE))
           (EQ (CAR FORM)
               (QUOTE QUOTE)))
       (CADR FORM))
      [(MEMB (CAR FORM)
             PRINEVAL-FNS)
       (PEVAL-APPLY (CAR FORM)
              (for X in (CDR FORM) collect (PEVAL X]
      (T (ERROR1 (PQUOTE (PROGN ILLEGAL PEVAL FORM , (!PPR TERM NIL)
                                %.))
                 (BINDINGS (QUOTE TERM)
                       FORM)
                 (QUOTE HARD])
```

(PEVAL-APPLY

```
[LAMBDA (FN ARGS)                                          (* kbr: "19-Oct-85 16:31")
  (SELECTQ FN
       (AND [COND
                ((NULL ARGS)
                 T)
                ((MEMB NIL ARGS)
                 NIL)
                (T (CAR (LAST ARGS])
       (OR (for X in ARGS thereis X))
       (FN-SYMB (FN-SYMB (CAR ARGS)))
       (FFN-SYMB (FFN-SYMB (CAR ARGS)))
       (ARGN (ARGN (CAR ARGS)
                   (CADR ARGS)))
       (FARGN (FARGN (CAR ARGS)
                     (CADR ARGS)))
       (SARGS (SARGS (CAR ARGS)))
       (FARGS (FARGS (CAR ARGS)))
       (QUOTEP (QUOTEP (CAR ARGS)))
       (FQUOTEP (FQUOTEP (CAR ARGS)))
       (APPLY FN ARGS])
```

(PEVALV

```
[LAMBDA (X)                                                (* kbr: "19-Oct-85 18:25")
  (LET (TEMP)
       (COND
          ((SETQ TEMP (ASSOC X *ALIST*))
           (CDR TEMP))
          (T (ERROR1 (PQUOTE (PROGN (!PPR X NIL)
                                    IS AN UNBOUND NLISTP IN PRINEVAL !))
                     (LIST (CONS (QUOTE X)
                                 X))
                     (QUOTE HARD])
```

(PLURALP

```
[LAMBDA (X)                                                (* kbr: "19-Oct-85 16:31")
  (NOT (OR (EQUAL X 1)
           (AND (LISTP X)
                (NLISTP (CDR X])
```

(!PPR-LIST

```
[LAMBDA (*LST*)                                            (* kbr: "19-Oct-85 16:31")
  (MAPRINEVAL (for X in *LST* collect (LIST (QUOTE !PPR)
                                            (KWOTE X)
                                            NIL))
        *INDENT* *FILE* NIL NIL (PQUOTE ,)
        (COND
           ((CDDR *LST*)
            (PQUOTE (PROGN , AND)))
           (T (PQUOTE AND])
```

(!PPR

```
[LAMBDA (X PUNCT)                                          (* kbr: "19-Oct-85 16:31")
  [LET (NCHARS)
       (SETQ X (EXPAND-PPR-MACROS X))
       (SETQ NCHARS (NCHARS X))
       [COND
          [(IGREATERP (IPLUS 2 (MAX (IPOSITION *FILE* NIL NIL)
                                    *INDENT*)
                              NCHARS)
                 (LINEL *FILE*))
           (COND
              ((AND (ILEQ (IPLUS *INDENT* NCHARS)
                          (LINEL *FILE*))
                    (ILESSP NCHARS 25))
```

```
                (ITERPRI *FILE*)
                (ISPACES *INDENT* *FILE*)
                (IPRINC X *FILE*)
                (AND PUNCT (PRINEVAL1 PUNCT)))
               (T (PRINEVAL1 (PQUOTE (PROGN : //)))
                  (PPRINDENT X (IPLUS *INDENT* 6)
                         (COND
                            (PUNCT (NCHARS PUNCT))
                            (T 0))
                         *FILE*)
                  (AND PUNCT (PRINEVAL1 PUNCT))
                  (ITERPRI *FILE*]
            (T (ISPACES (IDIFFERENCE *INDENT* (IPOSITION *FILE* NIL NIL))
                     *FILE*)
               (OR (IEQP (IPOSITION *FILE* NIL NIL)
                         *INDENT*)
                   (ISPACES 1 *FILE*))
               (IPRINC X *FILE*)
               (AND PUNCT (PRINEVAL1 PUNCT]
         (OR PUNCT (SETQ LAST-PRINEVAL-CHAR (COND
                                              ((LISTP X)
                                               (QUOTE ")"))
                                              (T (QUOTE X]
   NIL])
```

(PRIN5*

```
  [LAMBDA (X)                                                    (* kbr: "19-Oct-85 17:23")
    (LET (SPACES (*NOPOINT T))
         (SETQ SPACES (COND
                         ((IEQP 0 (IPOSITION *FILE* NIL NIL))
                          0)
                         ((EQ LAST-PRINEVAL-CHAR (QUOTE %.))
                          2)
                         ((EQ LAST-PRINEVAL-CHAR (QUOTE :))
                          2)
                         (T 1)))
         (COND
            [(MEMB X (QUOTE (// /# %. ! ? , :)))
             (COND
                ((EQ X (QUOTE //))
                 (ITERPRI *FILE*))
                ((EQ X (QUOTE /#))
                 (ISPACES (IDIFFERENCE *INDENT* (IPOSITION *FILE* NIL NIL))
                        *FILE*)
                 (ISPACES (IDIFFERENCE PARAGRAPH-INDENT 2)
                        *FILE*)
                 (SETQ LAST-PRINEVAL-CHAR (QUOTE %.)))
                [(OR (EQ X (QUOTE ,))
                     (EQ X (QUOTE :)))
                 (COND
                    ([AND [NOT (MEMB LAST-PRINEVAL-CHAR (QUOTE (%. , :]
                          (NOT (IEQP 0 (IPOSITION *FILE* NIL NIL]
                     (ISPACES (IDIFFERENCE *INDENT* (IPOSITION *FILE* NIL NIL))
                            *FILE*)
                     (IPRINC X *FILE*)
                     (SETQ LAST-PRINEVAL-CHAR X]
                ((OR (EQ X (QUOTE %.))
                     (EQ X (QUOTE !))
                     (EQ X (QUOTE ?)))
                 (ISPACES (IDIFFERENCE *INDENT* (IPOSITION *FILE* NIL NIL))
                        *FILE*)
                 (IPRINC X *FILE*)
                 (SETQ LAST-PRINEVAL-CHAR (QUOTE %.)))
                (T (ERROR1 (PQUOTE (PROGN THE CODE FOR PRIN5* IS INCONSISTENT : THE MEMB SAYS ONE THING
                                          AND THE COND SAYS ANOTHER %.))
                           (BINDINGS)
                           (QUOTE HARD]
            ((EQ X NIL)
             NIL)
            (T (ISPACES (IDIFFERENCE *INDENT* (IPOSITION *FILE* NIL NIL))
                      *FILE*)
               (COND
                  ((IGREATERP (IPLUS (IPOSITION *FILE* NIL NIL)
                                     SPACES
                                     (NCHARS X)
                                     1)
                              (LINEL *FILE*))
                   (ITERPRI *FILE*)
                   (ISPACES *INDENT* *FILE*))
                  (T (ISPACES SPACES *FILE*)))
               [COND
                  ((NUMBERP X)
                   (IPRINC X *FILE*))
                  (T (COND
                        ((EQ LAST-PRINEVAL-CHAR (QUOTE %.))
                         (IPRINC (CHARACTER (U-CASECODE (NTHCHARCODE X 1)))
```

```
                              *FILE*)
                        (for I from 2 to (NCHARS X) do (IPRINC (NTHCHAR X I)
                                                               *FILE*)))
                     (T (IPRINC X *FILE*]
               (SETQ LAST-PRINEVAL-CHAR NIL)))
        NIL])
```

(PRINEVAL

```
  [LAMBDA (FORM *ALIST* *INDENT* *FILE*)
    (PRINEVAL1 FORM])
```

(PRINEVAL1

```
  [LAMBDA (SUBFORM)                                          (* kbr: "19-Oct-85 18:31")
    (COND
       [(NLISTP SUBFORM)
        (PRIN5* (COND
                   ((FIXP SUBFORM)
                    (SPELL-NUMBER SUBFORM))
                   (T SUBFORM]
       (T (SELECTQ (CAR SUBFORM)
                (@ (PRINEVAL1 (PEVAL (CADR SUBFORM))))
                (? (for SUBFORM1 in (BM-NTH [ADD1 (RANDOM-NUMBER (LENGTH (CDR SUBFORM]
                                                 SUBFORM)
                      do (PRINEVAL1 SUBFORM1)))
                (COND (for SUBFORM1 in (CDR SUBFORM) thereis (COND
                                                               ((PEVAL (CAR SUBFORM1))
                                                                (for SUBFORM2 in (CDR SUBFORM1)
                                                                   do (PRINEVAL1 SUBFORM2))
                                                                T))))
                (PLURAL? [COND
                            ((PLURALP (PEVAL (CADR SUBFORM)))
                             (PRINEVAL1 (CADDR SUBFORM)))
                            (T (PRINEVAL1 (CADDDR SUBFORM]))
                (PROGN (for SUBFORM1 in (CDR SUBFORM) do (PRINEVAL1 SUBFORM1)))
                (PEVAL SUBFORM])
```

(PRINT-DEFN-MSG

```
  [LAMBDA (NAME ARGS)                                        (* kbr: "19-Oct-85 16:31")
    (PROG (TEMPS MEASURE RELATION LEMMAS FLG CONCL TIME N DEFNS)
          (SETQ LAST-PRIN5-WORD (QUOTE %.))
          (SETQ TIME (TIME-IN-60THS))
          (COND
             (IN-BOOT-STRAP-FLG (SETQ IOTHMTIME (DIFFERENCE (TIME-IN-60THS)
                                                            TIME))
                    (RETURN NIL)))
          (SETQ TEMPS (GETPROP NAME (QUOTE JUSTIFICATIONS)))
          [COND
             ((NOT (TOTAL-FUNCTIONP NAME))
              (ERROR1 (PQUOTE (PROGN THE ADMISSIBILITY OF (!PPR NAME NIL)
                                     HAS NOT BEEN ESTABLISHED %. WE WILL ASSUME THAT THERE EXISTS A FUNCTION
                                     SATISFYING THIS DEFINITION %. AN INDUCTION PRINCIPLE FOR THIS FUNCTION
                                     HAS ALSO BEEN ASSUMED , CORRESPONDING TO THE OBVIOUS SUBGOAL INDUCTION
                                     FOR THE FUNCTION %. THESE ASSUMPTIONS MAY RENDER THE THEORY INCONSISTENT
                                     %. // //))
                      (BINDINGS (QUOTE NAME)
                             NAME)
                      (QUOTE WARNING)))
             (T (SETQ N (SUB1 (LENGTH TEMPS)))
                (PRINEVAL (PQUOTE (PROGN /#))
                       (BINDINGS)
                       0 PROVE-FILE)
                (for TEMP in TEMPS as I from 1
                   do (SETQ MEASURE (fetch (JUSTIFICATION MEASURE-TERM) of TEMP))
                      (SETQ RELATION (fetch (JUSTIFICATION RELATION) of TEMP))
                      (SETQ FLG NIL)
                      (SETQ LEMMAS NIL)
                      (SETQ DEFNS NIL)
                      [for X in (fetch (JUSTIFICATION LEMMAS) of TEMP) do (COND
                                                                            ((EQ X (QUOTE ZERO))
                                                                             (SETQ FLG T))
                                                                            ((GETPROP X (QUOTE
                                                                                          TYPE-PRESCRIPTION-LST
                                                                                           ))
                                                                             (SETQ DEFNS (CONS X DEFNS)))
                                                                            (T (SETQ LEMMAS (CONS X LEMMAS]
                      (PRINEVAL [PQUOTE (PROGN (COND
                                                  (FINALLY? (COND
                                                              ((EQUAL N 2)
                                                               IN ADDITION)
                                                              (T FINALLY))
                                                            ,))
                                               (@ JUSTIFICATION-SENTENCE)
                                               EACH RECURSIVE CALL %.
                                               (COND
```

```
                                                ((EQUAL I 1)
                                                 HENCE , (!PPR NAME NIL)
                                                 IS ACCEPTED UNDER THE (? (PRINCIPLE OF DEFINITION)
                                                                          (DEFINITIONAL PRINCIPLE))
                                                 %.
                                                 (COND
                                                    ((EQUAL N 1)
                                                     THE DEFINITION OF (!PPR NAME NIL)
                                                     CAN BE JUSTIFIED IN ANOTHER WAY %.)
                                                    (OTHERS THERE ARE (@ N)
                                                           OTHER
                                                           (? (EXPLANATIONS OF)
                                                              (MEASURES AND WELL-FOUNDED FUNCTIONS
                                                                           EXPLAINING))
                                                           THE RECURSION ABOVE %.]
                                (BINDINGS (QUOTE N)
                                       N
                                       (QUOTE NAME)
                                       NAME
                                       (QUOTE I)
                                       I
                                       (QUOTE JUSTIFICATION-SENTENCE)
                                       (JUSTIFICATION-SENTENCE)
                                       (QUOTE RELATION)
                                       RELATION
                                       (QUOTE MEASURE)
                                       MEASURE
                                       (QUOTE DEFNS)
                                       DEFNS
                                       (QUOTE LEMMAS)
                                       LEMMAS
                                       (QUOTE FLG)
                                       FLG
                                       (QUOTE NUMBER)
                                       (LENGTH (fetch (JUSTIFICATION LEMMAS) of TEMP))
                                       (QUOTE FINALLY?)
                                       (AND (NOT (EQUAL I 1))
                                            (NOT (EQUAL N 1))
                                            (EQUAL I (ADD1 N)))
                                       (QUOTE OTHERS)
                                       (GREATERP N 1))
                                0 PROVE-FILE]
         (COND
            ([NOT (IEQP TYPE-SET-UNKNOWN (CAR (TYPE-PRESCRIPTION NAME]
             [SETQ TEMP-TEMP (CONS (DUMB-CONVERT-TYPE-SET-TO-TYPE-RESTRICTION-TERM (CAR (TYPE-PRESCRIPTION
                                                                                         NAME))
                                                                                    (CONS NAME ARGS))
                                   (for FLG in (CDR (TYPE-PRESCRIPTION NAME)) as I from 0 when FLG
                                      collect (LIST (QUOTE EQUAL)
                                                    (CONS NAME ARGS)
                                                    (BM-NTH I ARGS]
             [SETQ CONCL (COND
                            ((NULL (CDR TEMP-TEMP))
                             (CAR TEMP-TEMP))
                            (T (CONS (QUOTE OR)
                                     TEMP-TEMP]
             (PRINEVAL (PQUOTE (PROGN (? (NOTE THAT)
                                         (OBSERVE THAT)
                                         (FROM THE DEFINITION WE CAN CONCLUDE THAT))
                                      (!PPR CONCL NIL)
                                      IS A THEOREM %.))
                       (BINDINGS (QUOTE CONCL)
                                 CONCL)
                       0 PROVE-FILE)))
         (SETQ IOTHMTIME (DIFFERENCE (TIME-IN-60THS)
                                     TIME))
         (RETURN NIL])
```

(TH-IFY

```
  [LAMBDA (N)                                                    (* kbr: "26-Oct-85 14:00")
    (SELECTQ N
         (1 (QUOTE FIRST))
         (2 (QUOTE SECOND))
         (3 (QUOTE THIRD))
         (4 (QUOTE FOURTH))
         (5 (QUOTE FIFTH))
         (6 (QUOTE SIXTH))
         (7 (QUOTE SEVENTH))
         (8 (QUOTE EIGHTH))
         (9 (QUOTE NINTH))
         (10 (QUOTE TENTH))
         (11 (QUOTE 11TH))
         (12 (QUOTE 12TH))
         (13 (QUOTE 13TH))
         (COND
            [(FIXP N)
```

```
            (PACK (NCONC (UNPACK N)
                         (SELECTQ (REMAINDER N 10)
                              (1 (QUOTE ST))
                              (2 (QUOTE ND))
                              (3 (QUOTE RD))
                              (QUOTE TH]
          (T N])
```

(UN-NOTICE-CLAUSE

```
  [LAMBDA (CL)                                                    (* kbr: "26-Oct-85 13:52")
    (SETQ TEMP-TEMP (ASSOC CL CLAUSE-ALIST))
    [COND
       ((NULL TEMP-TEMP)
        (ERROR1 (PQUOTE (PROGN UN-NOTICE-CLAUSE WAS CALLED ON A CLAUSE NOT IN CLAUSE-ALIST !))
               NIL
               (QUOTE HARD]
    (SETQ CLAUSE-ALIST (DREMOVE TEMP-TEMP CLAUSE-ALIST))
    TEMP-TEMP])

)

(RPAQQ PPRCOMS ((* BM-PPR *)
                (FNS PPRIND PPRPACK PPR1 PPR2 PPR22 TERPRISPACES)))


          (* * BM-PPR *)

(DEFINEQ

(PPRIND
  [LAMBDA (FMLA LEFTMARGIN RPARCNT PPR-MACRO-LST PPRFILE)         (* kbr: "20-Oct-85 16:00")
    (PROG (MARG2 PPR-MACRO-MEMO STARTLIST)
          (SETQ MARG2 (LINEL PPRFILE))
          (COND
             ((NLISTP FMLA)
              (IPRIN1 FMLA PPRFILE)
              (RETURN NIL)))
          (SETQ POS (COND
                       ((SETQ TEMP-TEMP (ASSOC PPRFILE IPOSITION-ALIST))
                        (CDR TEMP-TEMP))
                       (T 0)))
          (SETQ SPACELEFT (IDIFFERENCE MARG2 LEFTMARGIN))
          (PPR1 FMLA (ADD1 RPARCNT))
          (SETQ NEXTNODE (CDAR STARTLIST))
          (SETQ NEXTIND (CAAR STARTLIST))
          (SETQ PPR-MACRO-MEMO (DREVERSE PPR-MACRO-MEMO))
          (SETQ NEXT-MEMO-KEY (CAR (CAR PPR-MACRO-MEMO)))
          (SETQ NEXT-MEMO-VAL (CDR (CAR PPR-MACRO-MEMO)))
          (PPR2 FMLA LEFTMARGIN RPARCNT)
          (IPOSITION PPRFILE POS NIL)
          (RETURN NIL])

(PPRPACK
  [LAMBDA NIL                                                     (* kbr: "19-Oct-85 16:31")
    (CONS (COND
             ((ILESSP MINREM DLHDFMLA)
              (SETQ REMAINDER 0)
              (MINUS (ADD1 MINREM)))
             (T (SETQ REMAINDER (IDIFFERENCE MINREM DLHDFMLA))
                (ADD1 DLHDFMLA)))
          FMLA])

(PPR1
  [LAMBDA (FMLA RPARCNT)                                          (* kbr: "22-Oct-85 16:08")
    (LET (DLHDFMLA RUNFLAT MINREM L RUNSTART RUNEND (PPR-MACRO-LST PPR-MACRO-LST))
         (PROG NIL
               (COND
                  ((NOT (LISTP FMLA))
                   (SETQ NCHARS (IPLUS RPARCNT (NCHARS FMLA)))
                   (SETQ REMAINDER (IDIFFERENCE SPACELEFT NCHARS))
                   (RETURN NIL)))
               (COND
                  ((NLISTP (CAR FMLA))
                   [COND
                      ((SETQ TEMP1 (ASSOC (CAR FMLA)
                                          PPR-MACRO-LST))
                       (SETQ TEMP1 (APPLY* (CDR TEMP1)
                                           FMLA))
                       (SETQ PPR-MACRO-MEMO (CONS (CONS FMLA TEMP1)
                                                  PPR-MACRO-MEMO))
                       (COND
                          ((NLISTP TEMP1)
                           (SETQ NCHARS (IPLUS RPARCNT (NCHARS TEMP1)))
                           (SETQ REMAINDER (IDIFFERENCE SPACELEFT NCHARS))
```

```
                        (RETURN NIL))
                       (T (SETQ FMLA TEMP1]
              (COND
                 ((AND (EQ (QUOTE QUOTE)
                           (CAR FMLA))
                       (NOT (NLISTP (CDR FMLA)))
                       (NULL (CDDR FMLA)))
                  (PPR1 (CADR FMLA)
                        RPARCNT)
                  (AND NCHARS (SETQ NCHARS (ADD1 NCHARS)))
                  (SETQ REMAINDER (SUB1 REMAINDER))
                  (RETURN NIL)))
              [SETQ DLHDFMLA (ADD1 (NCHARS (CAR FMLA]
              (SETQ L FMLA))
             (T (SETQ DLHDFMLA 0)
                (SETQ L (RPLACD NILCONS FMLA))
                (GO OVER)))
          (COND
             ((NULL (CDR FMLA))
              (SETQ NCHARS (IPLUS RPARCNT DLHDFMLA))
              (SETQ REMAINDER (IDIFFERENCE SPACELEFT NCHARS))
              (RETURN NIL)))
      OVER
          (SETQ RUNFLAT DLHDFMLA)
          (SETQ MINREM 1000)
          (SETQ SPACELEFT (SUB1 SPACELEFT))
      LOOPFLAT
          (SETQ L (CDR L))
          (COND
             ((NULL L)
              (SETQ SPACELEFT (ADD1 SPACELEFT))
              (COND
                 ((AND (NOT (IGREATERP RUNFLAT SPACELEFT))
                       (NOT (IGREATERP RUNFLAT FORCEIN)))
                  (SETQ NCHARS RUNFLAT)
                  (SETQ REMAINDER (IDIFFERENCE SPACELEFT RUNFLAT)))
                 (T (SETQ STARTLIST (CONS (PPRPACK)
                                          NIL))
                    (SETQ ENDLIST STARTLIST)
                    (SETQ NCHARS NIL)))
              (RETURN NIL)))
          (COND
             ((NLISTP L)
              (RPLACA (CDR DOTCONS)
                      L)
              (SETQ L DOTCONS)))
          [COND
             ((NLISTP (CAR L))
              (SETQ TEMP1 (NCHARS (CAR L)))
              (SETQ RUNFLAT (IPLUS TEMP1 (ADD1 RUNFLAT)))
              (SETQ TEMP1 (IDIFFERENCE SPACELEFT TEMP1))
              [COND
                 ((NULL (CDR L))
                  (SETQ RUNFLAT (IPLUS RPARCNT RUNFLAT))
                  (SETQ TEMP1 (IDIFFERENCE TEMP1 RPARCNT]
              (COND
                 ((ILESSP TEMP1 MINREM)
                  (SETQ MINREM TEMP1)))
              (GO LOOPFLAT))
             (T (PPR1 (CAR L)
                      (COND
                         ((NULL (CDR L))
                          (ADD1 RPARCNT))
                         (T 1)))
                (COND
                   ((ILESSP REMAINDER MINREM)
                    (SETQ MINREM REMAINDER)))
                (COND
                   (NCHARS (SETQ RUNFLAT (IPLUS NCHARS (ADD1 RUNFLAT)))
                           (GO LOOPFLAT]
          (SETQ RUNSTART STARTLIST)
          (SETQ RUNEND ENDLIST)
      LOOPIND
          (SETQ L (CDR L))
          (COND
             ((NULL L)
              (SETQ STARTLIST (CONS (PPRPACK)
                                    RUNSTART))
              (SETQ ENDLIST RUNEND)
              (SETQ NCHARS NIL)
              (SETQ SPACELEFT (ADD1 SPACELEFT))
              (RETURN NIL)))
          (COND
             ((NLISTP L)
              (RPLACA (CDR DOTCONS)
                      L)
              (SETQ L DOTCONS)))
```

```
               (COND
                  ((NLISTP (CAR L))
                   [SETQ TEMP1 (IDIFFERENCE SPACELEFT (NCHARS (CAR L]
                   [COND
                      ((NULL (CDR L))
                       (SETQ TEMP1 (IDIFFERENCE TEMP1 RPARCNT]
                   (COND
                      ((ILESSP TEMP1 MINREM)
                       (SETQ MINREM TEMP1)))
                   (GO LOOPIND)))
               (PPR1 (CAR L)
                     (COND
                        ((NULL (CDR L))
                         (ADD1 RPARCNT))
                        (T 1)))
               (COND
                  ((ILESSP REMAINDER MINREM)
                   (SETQ MINREM REMAINDER)))
               (COND
                  (NCHARS)
                  (T (RPLACD RUNEND STARTLIST)
                     (SETQ RUNEND ENDLIST)))
               (GO LOOPIND])
```

(PPR2

```
  [LAMBDA (FMLA MARG1 RPARCNT)                                (* kbr: "20-Oct-85 16:02")
    (PROG (NONLFLAG TEMP)
          (COND
             ((NLISTP FMLA)
              (PRIND FMLA PPRFILE)
              (RETURN NIL)))
          [COND
             ((EQ FMLA NEXT-MEMO-KEY)
              (SETQ FMLA NEXT-MEMO-VAL)
              (SETQ PPR-MACRO-MEMO (CDR PPR-MACRO-MEMO))
              (SETQ NEXT-MEMO-KEY (CAR (CAR PPR-MACRO-MEMO)))
              (SETQ NEXT-MEMO-VAL (CDR (CAR PPR-MACRO-MEMO)))
              (COND
                 ((NLISTP FMLA)
                  (PRIND FMLA PPRFILE)
                  (RETURN NIL]
          (COND
             ((AND (EQ (CAR FMLA)
                       (QUOTE QUOTE))
                   (NOT (NLISTP (CDR FMLA)))
                   (NULL (CDDR FMLA)))
              (PRIN1 "'" PPRFILE)
              (PPR2 (CADR FMLA)
                    (ADD1 MARG1)
                    RPARCNT)
              (RETURN NIL)))
          (COND
             [(EQ FMLA NEXTNODE)
              (SETQ MARG1 (IPLUS MARG1 (ABS NEXTIND)))
              (SETQ NONLFLAG (IGREATERP NEXTIND 0))
              (SETQ STARTLIST (CDR STARTLIST))
              (COND
                 ((NULL STARTLIST))
                 (T (SETQ NEXTNODE (CDR (CAR STARTLIST)))
                    (SETQ NEXTIND (CAR (CAR STARTLIST]
             (T (PPR22 FMLA)
                (RETURN NIL)))
          (PRIN1 "(" PPRFILE)
          [COND
             ((NLISTP (CAR FMLA))
              (PRIND (CAR FMLA)
                     PPRFILE)
              (COND
                 ((NULL (CDR FMLA))
                  (PRIN1 ")" PPRFILE)
                  (RETURN NIL)))
              (COND
                 ((AND (LISTP (CDR FMLA))
                       [OR (NLISTP (SETQ TEMP (CADR FMLA)))
                           (AND (NOT (EQ (CADR FMLA)
                                         NEXTNODE))
                                (PROGN (COND
                                          ((EQ FMLA NEXT-MEMO-KEY)
                                           (SETQ TEMP NEXT-MEMO-VAL)))
                                       (OR (NLISTP TEMP)
                                           (AND (EQ (CAR TEMP)
                                                    (QUOTE QUOTE))
                                                (NOT (NLISTP (CDR TEMP)))
                                                (NLISTP (CADR TEMP))
                                                (NULL (CDDR TEMP]
                       (ILESSP (IPLUS POS (NCHARS TEMP)
```

```
                                        RPARCNT)
                                MARG2))
                    (PRIN1 " " PPRFILE)
                    (PPR2 (CADR FMLA)
                          MARG1 RPARCNT)
                    (SETQ FMLA (CDR FMLA))
                    (GO LOOP1))
                   (NONLFLAG (PRIN1 " " PPRFILE))
                   (T (TERPRISPACES MARG1 PPRFILE)))
              (SETQ FMLA (CDR FMLA]
      LOOP
           (COND
              ((NLISTP FMLA)
               (PRIN1 "." PPRFILE)
               (PRIN1 " " PPRFILE)
               (PRIND FMLA PPRFILE)
               (PRIN1 ")" PPRFILE)
               (RETURN NIL)))
           (PPR2 (CAR FMLA)
                 MARG1
                 (COND
                    ((NULL (CDR FMLA))
                     (ADD1 RPARCNT))
                    (T 1)))
      LOOP1
           (COND
              ((NULL (CDR FMLA))
               (PRIN1 ")" PPRFILE)
               (RETURN NIL)))
           (COND
              ((AND (NLISTP (CAR FMLA))
                    (LISTP (CDR FMLA))
                    [OR (NLISTP (SETQ TEMP (CADR FMLA)))
                        (AND (NOT (EQ TEMP NEXTNODE))
                             (PROGN (COND
                                        ((EQ FMLA NEXT-MEMO-KEY)
                                         (SETQ TEMP NEXT-MEMO-VAL)))
                                    (OR (NLISTP TEMP)
                                        (AND (EQ (CAR TEMP)
                                                 (QUOTE QUOTE))
                                             (NOT (NLISTP (CDR TEMP)))
                                             (NLISTP (CADR TEMP))
                                             (NULL (CDDR TEMP]
                    (ILESSP (IPLUS POS (NCHARS TEMP)
                                   RPARCNT)
                            MARG2))
               (PRIN1 " " PPRFILE)
               (PPR2 (CADR FMLA)
                     MARG2 RPARCNT)
               (SETQ FMLA (CDR FMLA))
               (GO LOOP1)))
           (TERPRISPACES MARG1 PPRFILE)
           (SETQ FMLA (CDR FMLA))
           (GO LOOP])
```

(PPR22

```
  [LAMBDA (X)                                          (* kbr: "19-Oct-85 16:31")
    (COND
       ((NLISTP X)
        (PRIND X PPRFILE))
       (T (PRIN1 "(" PPRFILE)
          (PROG NIL
            LOOP
                (COND
                   ((NLISTP X)
                    (COND
                       ((NULL X)
                        (PRIN1 ")" PPRFILE))
                       (T (PRIN1 "." PPRFILE)
                          (PRIN1 " " PPRFILE)
                          (PRIND X PPRFILE)
                          (PRIN1 ")" PPRFILE)))
                    (RETURN NIL))
                   (T (PPR2 (CAR X)
                            MARG2 0)
                      (SETQ X (CDR X))
                      (COND
                         ((NULL X))
                         (T (PRIN1 " " PPRFILE)))
                      (GO LOOP])
```

(TERPRISPACES

```
  [LAMBDA (N FILE)                                     (* kbr: "22-Oct-85 15:53")
    (TERPRI FILE)
    (for I from 1 to N do (PRIN1 " " FILE))
```

```
    (SETQ POS N])

)

(FILESLOAD COMPILEBANG)

(DECLARE: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS

(ADDTOVAR NLAMA TOGGLE REFLECT PROVE-LEMMA ENABLE DISABLE DEFN DCL ADD-SHELL ADD-AXIOM)

(ADDTOVAR NLAML )

(ADDTOVAR LAMA )
)

(PUTPROPS BOYERMOORE COPYRIGHT ("Xerox Corporation" 1985 1986))
```

FUNCTION INDEX

{MEDLEY}<lispusers>BOYERMOORE.;1

MACRO INDEX

CONSTANT INDEX

RECORD INDEX

VARIABLE INDEX