<AFFIRM>LOGIC..31 30-Sep-81 15:53:18

```
(FILECREATED "28-Sep-81 19:11:45" <AFFIRM>LOGIC..31 73239

     changes to:  UnspecifiedLHSides

     previous date: "26-Sep-81 16:41:36" <AFFIRM>LOGIC..30)


(PRETTYCOMPRINT LOGICCOMS)

(RPAQQ LOGICCOMS [(RECORDS LGraphEntry)
                  (FNS * EqualityFunctions)
                  (PROP (EqualOp EQOP)
                        Equal)
                  (P (replace EQOP of EQOP with T)
                     (replace EqualOp of EQOP with EQOP))
                  (FNS * LOGICFNS)
                  (* ↑)
                  (FNS * UnificationFunctions)
                  (* ↑)
                  (FNS * ChainingFunctions)
                  (* ↑)
                  (FNS * InstantiationFunctions)
                  (* ↑)
                  (FNS * RulesFunctions)
                  (* ↑)
                  (FNS * RenamingFunctions)
                  (* ↑)
                  (FNS * SkolemizationFunctions)
                  (FNS * NoChangeFunctions)
                  (VARS (PRIMEDELIM (QUOTE #))
                        (PRIMELIMIT 2)
                        (NEWSYMBOLCHAR (QUOTE $))
                        (ShortenLessPrimesList (LIST 92 39 (CHCON1 PRIMEDELIM)))
                        (LeftUnification NIL))
                  [VARS (Accept (QUOTE (Accept " another equation from the terminal instead")))
                        (AddRuleOptionList NIL)
                        (Instead (QUOTE (Instead " accept another equation from the terminal")))
                        (KeepAsIs (QUOTE (Keep " it as it is (this may result in a stack overflow!)"))
                                  )
                        (Reverse (QUOTE (Reverse " it")))
                        [ReverseAndYes (QUOTE ((Reverse " it")
                                               (Yes ", reverse it" RETURN 'Reverse]
                        (Suppress (QUOTE (Suppress " it (put it on the list %"BadEquations%")")))
                        (TreatAsEquation (QUOTE (Treat " it as Equation == TRUE")))
                        (Yes (QUOTE (Yes NIL]
                  (DECLARE: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS
                            (ADDVARS (NLAMA)
                                     (NLAML Qexpression)
                                     (LAMA])
[DECLARE: EVAL@COMPILE

(RECORD LGraphEntry (LExpr LStatus))
]

(RPAQQ EqualityFunctions (EqualityFromIf EqvWithQex GetEqualOp ReverseEquality affirmed?))
(DEFINEQ
```

1

```
(EqualityFromIf
  [LAMBDA (x)                                          (* D.Taylor "13-Sep-79 16:44")
    (if x:Operator=IFOP
        then [if x:ThenPart=TRUE and x:ElsePart=FALSE
                 then <EQVOP x:Test TRUE>
             elseif x:ThenPart=FALSE and x:ElsePart=TRUE
               then <EQVOP x:Test FALSE>
             else (PROG (thenEq elseEq)
                        (if x:ThenPart=TRUE
                            then elseEq←(EqualityFromIf x:ElsePart)
                                 (RETURN <elseEq:Operator elseEq:LHS
                                              <IFOP x:Test elseEq:LHS elseEq:RHS>>)
                          elseif x:ElsePart=TRUE
                            then thenEq←(EqualityFromIf x:ThenPart)
                                 (RETURN <thenEq:Operator thenEq:LHS
                                              <IFOP x:Test thenEq:RHS thenEq:LHS>>)
                          elseif x:ThenPart=FALSE
                            then elseEq←(EqualityFromIf x:ElsePart)
                                 (RETURN <ANDOP elseEq <NOTOP x:Test>>)
                          elseif x:ElsePart=FALSE
                            then thenEq←(EqualityFromIf x:ThenPart)
```

```
                    (RETURN <ANDOP thenEq x:Test>)
               else thenEq←(EqualityFromIf x:ThenPart)
                    elseEq←(EqualityFromIf x:ElsePart)
                    (if (EQUAL thenEq:LHS elseEq:LHS)
                        then (RETURN <thenEq:Operator thenEq:LHS
                                                <IFOP x:Test thenEq:RHS elseEq:RHS>>)
                      else (PRINTLINES T "**** can't make equality out of " T)
                           (PrettyPrint x)
                           (ERROR!]
    else x])
```

2

```
(EqvWithQex
  [LAMBDA (x)                                          (* D.Musser "18-Jul-79 16:11")
    <EQVOP x:LHS (EVAL x:RHS)
    >])
```

3

```
(GetEqualOp
  [LAMBDA (type)                                       (* D.Musser "24-Jun-79 12:31")
    (if type:EqualOp
      else (type:EqualOp←(ExtendName EQOP type])
```

4

```
(ReverseEquality
  [LAMBDA (eqexp)                                      (* D.Musser "25-Jun-79 18:43")
    <eqexp:Operator eqexp:RHS eqexp:LHS>])
```

5

```
(affirmed?
  [LAMBDA NIL                                          (* D.Thompson " 1-Nov-79 17:44")

        (* * This routine determines whether or not the rewrite rule database is convergent. -
        An AFFIRM command function.)


    (PROG (ContradictionFound (rulelimit NIL)
                              (rulecount 0))
          (Completer NIL 'lemma])
)

(PUTPROPS Equal EqualOp Equal)

(PUTPROPS Equal EQOP T)
(replace EQOP of EQOP with T)
(replace EqualOp of EQOP with EQOP)

(RPAQQ LOGICFNS (AddHypothesis ApplyAllEqHyps ApplyEqHyp CheckIntroducedExpr CompleteSubs ConstOrder
                               EqvFromIf GetConjunct ListOfConjuncts NonvarSubexpressions Operators
                               PutForm SecondaryOperators operators1))
(DEFINEQ
```

6

```
(AddHypothesis
  [LAMBDA (hypo propn)                                 (* R.Erickson "15-Sep-80 17:12")

        (* * Given an hypothesis and a quantified expression, add the hypothesis)


    (CheckForQexpression propn T)
    (create Qexpression
            expr ←(<IFOP hypo propn:expr TRUE>) using propn])
```

7

```
(ApplyAllEqHyps
  [LAMBDA (x)                                          (* D.Musser "10-Aug-79 14:19")

        (* x is an expression in implication form (result of Removeifs). Substitutions are performed based on equality
        hypotheses, for each implication that is contained in x. Choice of which substitutions to perform is based on
```

contents of non-local variable ReplaceList. If ReplaceList is NIL, all equality hypotheses are used to make RHS for LHS substitutions. If ReplaceList is non NIL, and a LHS occurs in ReplaceList then RHS for LHS substitution is made; otherwise if RHS occurs in ReplaceList then LHS for RHS substitution is made; otherwise, no substitution is made.)

```
(if x:Operator=ANDOP
    then <ANDOP (ApplyAllEqHyps x.Arg1)
           (ApplyAllEqHyps x:Arg2) >
  elseif x:Operator=IMPOP
    then x← <IMPOP (ApplyAllEqHyps x:Arg1)
              (ApplyAllEqHyps x:Arg2) >
         (for i from 1 bind h eachtime h←(GetConjunct x:Arg1 i) when (IsEq h:Operator)
            do (if ReplaceList=NIL or (MEMBER h:LHS ReplaceList) or (MEMBER h:RHS ReplaceList)
                   then x←(ApplyEqHyp h x))
            repeatuntil h=NIL)
         x
  elseif x:Operator=IFOP
    then <IFOP x:Test (if (IsEq x:Test:Operator)
                          then (ApplyEqHyp x:Test (ApplyAllEqHyps x:ThenPart))
                        else (ApplyAllEqHyps x:ThenPart))
           (ApplyAllEqHyps x:ElsePart) >
  else x])
```

8

(ApplyEqHyp

```
[LAMBDA (h x)                                        (* D.Musser "11-Jun-79 22:25")
```

(h is an equation, x is a predicate expression. Performs a substitution based on h and ReplaceList (non-local variable), as described in the comment in ApplyAllEqHyps.)*

```
(if x=h
    then x
  elseif x:Operator=ANDOP or x:Operator=IMPOP
    then <x:Operator (ApplyEqHyp h x:Arg1)
           (ApplyEqHyp h x:Arg2) >
  elseif ReplaceList=NIL or (MEMBER h:LHS ReplaceList)
    then (UCI\SUBST h:RHS h:LHS x)
  elseif (MEMBER h:RHS ReplaceList)
    then (UCI\SUBST h:LHS h:RHS x)
  else x])
```

9

(CheckIntroducedExpr

```
[LAMBDA (expr propn restriction except)              (* R.Erickson " 3-Feb-81 17:14")
```

(* The user has supplied an expression which is to be added to the Qexpression propn. We wish to check on the free variables which occur in this expression; if they pass the test, we return them, otherwise cause an error. Variables in except are always allowed.)*

```
(PROG (frees given find bads)
      (frees←(Frees expr))
      (given←(for v in propn:given collect v:Operator))
      (find←(for v in propn:find collect v:Operator))    (* this :Operator business is strange, but widely done.)
      (SELECTQ restriction
               (given (if bads←(LDIFFERENCE frees < ! given ! except>)
                          then (AffirmError <"The free variables" (Shorten bads)

                                            "are not universally-quantified in the proposition."
                                            >)))
               (NIL (if bads←(LDIFFERENCE frees < ! given ! find ! except>)
                        then (AffirmError <"The free variables" (Shorten bads)
                                          "are not bound in the proposition." >)))
               (SHOULDNT))
      (RETURN frees])
```

10

(CompleteSubs

```
[LAMBDA (sublist ex)                                 (* R.Bates "14-Dec-79 14:43")
  (if (LITATOM ex)
      then (PROG (val)
                 (val←(FASSOC ex sublist))
                 (RETURN (if val
                             then (CompleteSubs sublist val::1)
```

```
                        else ex)))
    elseif (LISTP ex)
      then <ex:Operator !(for z in ex:Arguments collect (CompleteSubs sublist z)) >
    else ex])
```

11

```
(ConstOrder
  [LAMBDA (x y)                                    (* edited: " 6-Apr-79 09:26")
    (if (LISTP x)
        then (if (LISTP y)
                 then (EXTENT x) lt (EXTENT y)
               else NIL)
    elseif (LISTP y)
      then T
    elseif (EQUAL x y)
      then NIL
    else (ALPHORDER x y])
```

12

```
(EqvFromIf
  [LAMBDA (x)                                      (* R.Bates "18-Jun-79 17:26")

        (* x is assumed to be of the form (IfThenElse p q r). Returns the equation (p eqv (if (q eqv not r) then q else p))
        (where RHS is simplified: note that if (q eqv not r) simplifies to TRUE then result is (p eqv q); otherwise result
        is a conditional equation))

    <EQVOP x:Test (EVAL <IFOP <EQVOP x:ThenPart <IFOP x:ElsePart FALSE TRUE>> x:ThenPart x:Test>)
    >])
```

13

```
(GetConjunct
  [LAMBDA (x i)                                    (* R.Bates "23-Jan-80 13:40")

        (* x is a predicate expression, I is a positive integer. Result is the i-th conjunct in x, where the conjuncts are
        numbered from left to right in print order. If there are fewer than i conjuncts, result is NIL.)

    (CAR (FNTH (ListOfConjuncts x)
               i])
```

14

```
(ListOfConjuncts
  [LAMBDA (ex)
    (if (LISTP ex) and ex:Operator=ANDOP
        then <!!(ListOfConjuncts ex:Arg1) !(ListOfConjuncts ex:Arg2) >
      else <ex>])
```

15

```
(NonvarSubexpressions
  [LAMBDA (x)                                      (* Computes list of all nonvariable subexpressions of x
                                                   (including x itself))
    (if (LISTP x)
        then <x !(for y in x:Arguments join (NonvarSubexpressions y))
             >])
```

16

```
(Operators
  [LAMBDA (ex)                                     (* R.Erickson "13-Dec-79 18:53")

        (* * Returns a list of all symbols which appear in operators positions in ex.)

    (REMOVEDUPLICATES (operators1 ex])
```

17

```
(PutForm
```

```
  [LAMBDA (alist)

          (* alist is a nonempty list ((a1 . b1) ... (an . bn)). Result is conjunction of equations a1 = b1 and ...
          and an = bn (in internal form).)


    (PROG (result)
          (alist←(REVERSE alist))
          (result← <EQOP alist:1:1 alist:1::1>)
          (for pair in alist::1 do result← <ANDOP <EQOP pair:1 pair::1> result>)
          (RETURN result])
```

18

```
(SecondaryOperators
  [LAMBDA (lhs)                                          (* R.Erickson "12-Dec-79 13:23")

          (* * all operators which occur in this LHS and are not primary)


    (REMOVE lhs:Operator (Operators lhs])
```

19

```
(operators1
  [LAMBDA (ex)                                           (* R.Erickson "13-Dec-79 18:53")
    (if (LISTP ex)
        then <ex:Operator !(for a in ex:Arguments join (operators1 a))
             >])
)
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

```
(* ↑)  ]

(RPAQQ UnificationFunctions (LeftUnifiableWithSomeSubexpression LeftUnifier OccursUsingBindings
                                                                Unifier Unify1
                                                                UnifyWithSubExpressions))
(DEFINEQ

                                                                                                20

(LeftUnifiableWithSomeSubexpression
  [LAMBDA (x y)
    (OR (LeftUnifier x y)
        ~='NotLeftUnifiable
        (if (LISTP y)
            then (for a in y:Arguments unless (ATOM a) thereis (LeftUnifiableWithSomeSubexpression
                                                                x a])

                                                                                                21

(LeftUnifier
  [LAMBDA (x y)
    (RESETVARS (MatchBindings (LeftUnification T)
                              u)
               (u←(Unifier x y))
               (RETURN (if u~='NotUnifiable and (Matcher (SUBLIS u x)
                                                         (SUBLIS u y))
                           then <MatchBindings u>
                           else 'NotLeftUnifiable])

                                                                                                22

(OccursUsingBindings
  [LAMBDA (x y)                                                    (* R.Bates "14-Dec-79 14:43")
    (if (EQUAL x y)
        then T
      elseif (LITATOM y) and ~(y : IsConstant)
        then (if (FASSOC y UnifierBindings)
                 then (OccursUsingBindings x (FASSOC y UnifierBindings)::1))
      elseif (LISTP y)
        then (for z in y:Arguments thereis (OccursUsingBindings x z])

                                                                                                23

(Unifier
  [LAMBDA (x y)
    (RESETVARS (UnifierBindings)
               (RETURN (if (Unify1 x y)
                           then (for b in UnifierBindings collect <b:1
                                                                   !(CompleteSubs UnifierBindings
                                                                                  b::1)
                                                                   >)
                           else 'NotUnifiable])

                                                                                                24

(Unify1
  [LAMBDA (x y)                                                    (* R.Bates "14-Dec-79 14:43")
    (if (LITATOM x) and ~(x : IsConstant)
        then (if (FASSOC x UnifierBindings)
                 then (Unify1 (FASSOC x UnifierBindings)::1 y)
               elseif (EQUAL x y)
                 then T
               elseif (OccursUsingBindings x y)
                 then LeftUnification
               else UnifierBindings← <<x ! y> ! UnifierBindings>)
      elseif (LITATOM y) and ~(y : IsConstant)
        then (if (FASSOC y UnifierBindings)
                 then (Unify1 x (FASSOC y UnifierBindings)::1)
               elseif (OccursUsingBindings y x)
                 then NIL
               else UnifierBindings← <<y ! x> ! UnifierBindings>)
      elseif (NLISTP x)
        then (EQUAL x y)
      elseif x:Operator=y:Operator
        then (for z in x:Arguments as w in y:Arguments always (Unify1 z w])
```

25

```
(UnifyWithSubExpressions
  [LAMBDA (x y)
    (PROG (z)
          (z←(Unifier x y))
          (RETURN (if z='NotUnifiable
                      then (for a in y:Arguments unless (ATOM a) join (UnifyWithSubExpressions x a))
                    else <z>])
)
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

```
(* T)  ]

(RPAQQ ChainingFunctions (Chainings Chainings1 ChooseChainingsAndNarrowings TryChainingsAndNarrowings)
)
(DEFINEQ
```

26

```
(Chainings
 [LAMBDA (x)                                                    (* D.Musser "25-Apr-79 10:16")
   (REMOVEDUPLICATES (Chainings1 x])
```

27

```
(Chainings1
  [LAMBDA (x)                                                   (* D.Musser "25-Apr-79 10:16")
    (if (NLISTP x)
        then NIL
      elseif x:Operator=IFOP
        then < !!(UnifyWithSubExpressions x:Test x:ElsePart) !!(UnifyWithSubExpressions x:Test
                                                                                    x:ThenPart)
             !!(Chainings1 x:ThenPart) !(Chainings1 x:ElsePart) >
      else NIL])
```

28

```
(ChooseChainingsAndNarrowings
  [LAMBDA (x level choice substitutions)                        (* R.Bates " 4-Sep-80 15:19")
    (PROG (clist c chainingNo subs psubs)
          (clist←(Chainings x))
          (if clist
              then chainingNo←choice:1
                   (if chainingNo gt (FLENGTH clist)
                       then (printout NIL .TAB0 0 "Only " (FLENGTH clist)
                                      " choice(s) at level "
                                      (level+1)
                                      T)
                            (AffirmError))
                   c←(FNTH clist chainingNo):1
                   (printout NIL T .TAB0 (level*5)
                             chainingNo "/" (FLENGTH clist)
                             ":  " # (PrettyPrint (PutForm c)
                                                  T))
                   x←(EVAL (SUBLIS c x))
                   (RETURN (if choice::1=NIL
                               then subs←(for c in < ! substitutions c> join c)
                                    psubs←(PutForm subs)
                                    (let psubs T 'choose)
                               else (ChooseChainingsAndNarrowings x level+1 choice::1
                                                                  < ! substitutions c>)))
            else (AffirmError (CONCAT "Extra arguments: " choice)
                              NIL T])
```

29

```
(TryChainingsAndNarrowings
  [LAMBDA (x level substitutions)                               (* R.Bates " 4-Sep-80 15:28")
    (PROG (clist result psubs temp)
          (clist←(Chainings x))
          (if clist
              then (RETURN (for c in clist as chainingNo from 1
                              do (if c=NIL
                                     then (AffirmError
                    "Search came up with the null substitution;
 Please send transcript to PV group"
                                                   'mild)
                                  else (printout NIL T .TAB0 (level*5)
                                                 chainingNo "/" (FLENGTH clist)
                                                 ":  " # (PrettyPrint (PutForm c)
                                                                      T))
                                       result←(EVAL (SUBLIS c x))
                                       (if result=TRUE
                                           then (printout NIL .TAB0 0
                                                          "Proved by chaining and narrowing"
                                                          T "using the substitution " T)
                                                psubs←(PutForm (for c in < ! substitutions c>
```

```
                                        join c))
                                (PrettyPrint psubs)
                                (RETURN (let psubs T 'search))
                            else (if temp←(TryChainingsAndNarrowings result level+1
                                                                     <
                                                                       ! substitutions
                                                                       c>)
                                     then (RETURN temp])
)
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

```
(* ↑)  ]

(RPAQQ InstantiationFunctions (CircularSubs CircularSubs1 Instance Instance? InstanceQ
                                            InvalidDependencies SkolemizeProperly))
(DEFINEQ

                                                                                    30

(CircularSubs
  [LAMBDA (sublist)                                          (* edited: "12-SEP-78 16:36")
    (PROG (deps (xpanded NIL)
                (circularity NIL))

          (* given a substitution list, return a list of vars which participate in the first detected circularity.
          deps: first level variable dependencies. xpanded: elements of deps taken to transitive closure)


          (deps←(for s in sublist collect <s:1 !(Frees s::1)
                                           >))
          (for d in deps until circularity do (CircularSubs1 d:1 NIL))
          (RETURN circularity])

                                                                                    31

(CircularSubs1
  [LAMBDA (gofrom wherebeen)                                 (* R.Bates "23-Jan-80 12:58")
    (PROG (res)                                              (* return the full dependencies
                                                             (including reflexive) for variable "gofrom". If we run
                                                             into wherebeen elements, halt and note circularity)
          (RETURN (if res←(FASSOC gofrom xpanded)
                      then (for been in wherebeen when been MEMB res do circularity←
                                                                          <been ! circularity>)
                           < ! res>
                      elseif res←(FASSOC gofrom deps)
                        then (if gofrom MEMB wherebeen
                                 then circularity← < !! circularity
                                                       !(for x in wherebeen repeatuntil x=gofrom
                                                          collect x)
                                                    >)
                             res← <res:1 !(for x in res::1 until circularity
                                                join (CircularSubs1 x <gofrom ! wherebeen>))
                                  >
                             xpanded← <res ! xpanded> res
                      else                                   (* assert gofrom ~memb wherebeen)
                           <gofrom>])

                                                                                    32

(Instance
  [LAMBDA (subs qex)                                         (* R.Erickson "20-Feb-80 14:07")

          (* * subs is a Sublist, from atomic variable to value. (Values might be atomic.) We perform the instantiation,
          fiddling with Skolem fns. Correctness uncertain. Returns just the :expr part of qex.)


          (* * Our format, as I understand it, is the following: Given and Find lists contain Skolem functions, showing the
          variable dependencies. In the expression, Given variables will be Skolem functions, any time after Instance has been
          called. (For this reason, some TheoremProver commands call Instance with subs = NIL. Surely it would be possible to
          establish this invariant from the start.) Find variables start out atomic, and will stay that way until preturbed
          (instantiated OR rendered Given by lemma application and Instance); they never get changed back to atoms from Skolem
          fns, apparently. RWE.)


    (PROG (qsubs ex1 rules)
          (qsubs← < ! subs !(for v in qex:given collect <v:Operator ! v>)
                  >)                                         (* a sublist which, besides instantiating atoms, also
                                                             expands any atomic givens out to the full thing.)
          (ex1←(CompleteSubs qsubs qex:expr))

          (* * step 2: Instantiate any Skolem finds. This is done as rewrite rules, so we handle dependency lists properly.
          (For example, find "i(j,k)" with (i k) in subs causes "i(xx 1)" to be 1) Note that, because of the behavior of
          Qreduction, the arity of finds in ex1 may be less that that in the find list. In such cases, any references in the
          new value to missing args will become free. (For example, if the find list contains "i(j,k)" and subs includes
          (i k), then an occurance of "i(xx)" will translate to k.) Such free references will be untouched by CompleteSubs.)
```

```
        (rules←(for v in qex:find when (FASSOC v:Operator subs) and (OccursAsOperatorIn v:Operator
                                                                    qex:expr)
                bind value collect                          (* a rule assigning the new value)
                                   (value←(CompleteSubs (for s in qsubs
                                                           unless s:1 MEMB v:Arguments
                                                           collect
                                                         (* expand fully any nonargument variables, and/or
                                                         instantiate)
                                                                 s)
                                                         (SUBLIS subs v:Operator)))
                                                         (* RuleList has entries with CAR the equation)
                                   (<<EQOP v value>>))))
        [if rules
            then ex1←(RESETVARS ((UsingRuleList T)
                                 (RuleList rules))
                                (RETURN (EvalRules ex1]
        (RETURN ex1])
```

33

```
(Instance?
  [LAMBDA (subs qex)                                    (* R.Erickson "16-Sep-80 17:52")
    NIL])
```

34

```
(InstanceQ
  [LAMBDA (qex)                                         (* R.Erickson "16-Sep-80 17:53")
    NIL])
```

35

```
(InvalidDependencies
  [LAMBDA (varlist qexpr)                               (* edited: " 6-Apr-79 08:50")

        (* returns a list (bag) containing given or find vars from qexpr which, directly or indirectly, have Skolem
        dependencies upon vars in varlist, and which, therefore, varlist items may not depend on. Not reflexive :
        x is permitted to depend upon itself varlist + result are names -
        no Skolem args.)


    (PROG (qs deps)                                     (* list of Skolem fns w/ args :)
          (qs← < ! qexpr:given ! qexpr:find>)           (* list of lists (Skfn ... all it depends on))
          (deps←(for v in qs collect (CompleteDeps qs v)))
          (if (NLISTP varlist)
              then varlist← <varlist>)
          (RETURN (for v in varlist join (for d in deps when v MEMB d::1 collect d:1])
```

36

```
(SkolemizeProperly
  [LAMBDA (expr context)                                (* R.Erickson "17-Sep-80 12:37")

        (* * given an expression, ensure that its variables have been properly Skolemized. Otherwise, problems may show up
        in normalization or commands like complete. If expr is a non-Qexpression, one may supply context, a Qex which tells
        the sense of vars.)


    (if (type? Qexpression expr)
        then (create Qexpression
                     expr ←(Instance NIL expr) using expr)
      elseif context
        then (CheckForQexpression context T)
             (Instance NIL (create Qexpression
                                   expr ← expr using context))
      else expr])
)
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

```
(* ↑)  ]

(RPAQQ RulesFunctions (AcceptNewRules AddAutoRules AddRule Completer CriticalPairs ReadRules
                                ReportContradictionAndAbort Rule addRuleCannot addRuleDirection
                                rulequan))
(DEFINEQ
```

37

(AcceptNewRules

```
  [LAMBDA (kind)                                          (* D.Thompson "29-Oct-80 17:30")
```

(* This routine accepts rules from the terminal, after the user has answered "ACCEPT" to a system question during Knuth-Bendix convergence processing.)*

```
    (PROG (done rule rules)
          (if [UNDONLSETQ (until done do (printout NIL .TAB0 0 kind ,)
                                         (if rules←(ReadRules Terminal CommandTerminator kind)
                                             then done←T
                                                  (for r in rules
                                                     do (if rule←(pexec r T)
                                                            and (NoFreeVarsInRule rule kind)
                                                          then (AddRule rule kind T)
                                                        else done←NIL)))
                                         (if done
                                           else (printout NIL .TAB0 0 "Try again:"]
              else (printout NIL .TAB0 0 "(returning to previous rule processing...)" T])
```

38

(AddAutoRules

```
  [LAMBDA (rules source auxInfo)                          (* R.Erickson "26-Aug-81 14:05")
```

(rules is a list of equalities; source an atomic key, one of (reflexive distinct nochange); auxInfo the target names, may be long)*

(* Automatic rules have already been computed; this is a central place for their entry. We may print a message.)*

```
    (if rules
        then                                              (* all but reflexive announce themselves)
             (SELECTQ source
                      ((distinct nochange)
                        (printout NIL .TAB0 0 "Generating" , (LENGTH rules)
                                  , "'" source "' rules for the functions:" ,)
                        (AFFIRMMAPRINT NIL (Shorten auxInfo)
                                       T T))
                      NIL)
             (PROG ((ContradictionFound)
                    (rulelimit)
                    (rulecount 0))                        (* rebind these vars because of the archaic limit mechanism.)
                   (for rule in rules do (AddRule rule RuleTypes:Automatic T))
                   (if Completion
                       then (Completer NIL RuleTypes:Automatic])
```

39

(AddRule

```
  [LAMBDA (equation kind ruleReadIn sentenceStarted)
    (CLISP:(RuleList UNDOABLE)
      (Unchecked UNDOABLE))                               (* D.Thompson "13-Nov-80 13:25")
                                                          (* CLISP: delcarations are ignored at present! RE 7/80)
```

(* Makes the equation into a rewrite rule, if possible, and adds it to global list Unchecked. Checks rule for variable sets, self-looping. If UsingRuleList, rules are added to RuleList; otherwise, they are edited into Lisp.)*

```
    (PROG (reducedEquation directive au keyList optionList question)
          (if ProvingMode
              then (listOneRule equation:LHS RewriteRuleOp equation:RHS))
          (if ProvingMode and (PROGN question←"Should this be set up as a goal to be proved? "
                                     (AFFIRMUSER NIL 'N question))='Y
```

```
    then (MakeTheorem (ExprToNode equation))
else                                                    (* try to make it a rewrite rule.)
    (if rulelimit and (IGEQ rulecount rulelimit)
        then                                            (* rechecking -- was interrupted last time.)
             (AffirmError "RuleCount > RuleLimit!"))
    (if ruleReadIn
        then

(* * only need to simplify rule if read in, since otherwise these are already simplified rules from Completer
interactions.)


             reducedEquation←(EvaluateRule equation)
             (if (AnywhereBound reducedEquation:LHS:Arguments reducedEquation:RHS)
                 then (printout NIL (if sentenceStarted
                                        then "...Whoops!"
                                      else NullString)
                                .TAB0 0
                      "The arguments to a rule may not be bound on the right-hand-side!"
                                #
                                (listOneRule reducedEquation:LHS RewriteRuleOp
                                             reducedEquation:RHS))
                     (AffirmError))
             (if ~(EQUAL reducedEquation equation)
                 then (if reducedEquation=TRUE
                          then (if ruleReadIn
                                   then (printout NIL "Rule simplifies to TRUE."))
                               (RETURN)
                        elseif reducedEquation=FALSE
                          then (ReportContradictionAndAbort)
                        else equation←(if reducedEquation:Operator=IFOP
                                          then (EqualityFromIf reducedEquation)
                                       elseif reducedEquation:Operator=QOP
                                          then (EqvWithQex equation)
                                       else reducedEquation)))
  elseif equation:Operator=IFOP
    then equation←(EqualityFromIf equation))
directive←(addRuleDirection equation ruleReadIn sentenceStarted)
                                                        (* direction unseen)
(if directive:1='Reverse
    then equation← <equation:Operator equation:RHS equation:LHS>)
(if directive:2
    then                                                (* user has not seen)
         (printout NIL (if sentenceStarted
                           then "a new"
                         else "New")
                   , "rule" Colon .TAB0 5 # (listOneRule equation:LHS RewriteRuleOp
                                                         equation:RHS))
         (if CautiousCompletion
             then question←"Ok? " keyList← <Yes Suppress TreatAsEquation> au←(
                    AFFIRMUSER NIL NIL question keyList NIL NIL AddRuleOptionList)
                  (if au~='Yes
                      then                              (* Suppress or Treat)
                           directive:1←au)))
(if (ATOM equation:LHS)
    then (ReportContradictionAndAbort))
(SELECTQ directive:1
         (Suppress (/SET 'BadEquations <equation ! BadEquations>))
         (Instead                                       (* save the old one to check later)
                  (AcceptNewRules kind)
                  (/SET 'Unchecked (MERGE <<(if (LISTP Unchecked)
                                                and (FIXP Unchecked:-1:1)
                                              then Unchecked:-1:1+1
                                            else (COUNT equation:LHS))
                                          ! equation>>
                                        Unchecked T)))
         (PROGN                                         (* a real rule)
                (if directive:1='Treat
                    then equation← <EQVOP equation TRUE>)
                                                        (* CAUTION: the primary operator of the original rule
                                                        will now be on the LHS, so its type matters.)
                (if rulelimit
                    then rulecount←rulecount+1)
                (if UsingRuleList
                    then (/SET 'RuleList <<equation !(Operators equation:LHS)
                                                        > ! RuleList>)
                  else                                  (* only compiled rules have LeftHandSides recorded)
                       (CompileRuleIntoLisp equation kind))
                (/SET 'Unchecked (MERGE <<(if equation:RHS=FALSE
                                              then 1
                                            else (COUNT equation:LHS))
```

```
                                        ! equation>>
                                      Unchecked T))
                                         (* Unchecked is in ascending order by LHS size)
                              ])
```

40

```
(Completer
  [LAMBDA (equations kind)                                    (* R.Erickson "26-Aug-81 13:24")
    (PROG (erset rule1 newrule uncheckedwithrule1 removedKind)
          (erset←(ERSETQ
              (PROGN (for equation in equations do (AddRule equation kind T))
                                                              (* changes Unchecked list)
                     [for number from 1 while Unchecked
                        do (if UserInterrupt
                               then UserInterrupt←NIL
                                    (printout NIL .TAB0 0
                                     "Interrupted. Type command(s), then type  ok;  to continue."
                                        T)
                                    (if (RESETVARS ((Completion NIL))
                                                   (RETURN (Dtvs Terminal)))='aborted
                                        then (AffirmError)))
                            (rule1←Unchecked:1::1)
                            (/SET 'Unchecked Unchecked::1)
                                                              (* undoable (pop Unchecked))
                            (printout NIL .TAB0 0 number Slash (number+(FLENGTH Unchecked)))
                            (newrule←(SimplifyRule rule1))
                            (if newrule=FALSE
                                then (ReportContradictionAndAbort)
                              elseif newrule=TRUE
                                then (RemoveRule rule1)
                              elseif ~(EQUAL newrule:LHS rule1:LHS)
                                       and ~(EQUAL (EqualityFromIf newrule):LHS rule1:LHS)
                                       and ~(EQUAL newrule:LHS rule1:LHS)
                                then removedKind←(RemoveRule rule1)
                                     (AddRule newrule removedKind)
                                                              (* changes Unchecked)
                              else uncheckedwithrule1←(RulesThatMightInteract rule1
                                                                              (for r in Unchecked
                                                                                 collect r::1))
                                   (for rule2 in uncheckedwithrule1 unless rule1=NIL
                                      do (printout NIL Period)
                                         (newrule←(SimplifyRule rule2))
                                         (if newrule:Operator=QOP
                                             then newrule←(EqvWithQex rule2))
                                         (if newrule=FALSE
                                             then (ReportContradictionAndAbort)
                                           elseif newrule=TRUE
                                             then (RemoveRule rule2)
                                                  (if (EQUAL rule1 rule2)
                                                      then rule1←NIL)
                                           elseif ~(EQUAL newrule:LHS rule2:LHS)
                                                    and ~(EQUAL (EqualityFromIf newrule):LHS rule2:LHS)
                                             then removedKind←(RemoveRule rule2)
                                                  (if (EQUAL rule1 rule2)
                                                      then rule1←NIL)
                                                  (AddRule newrule removedKind)
                                                              (* changes Unchecked)
                                           else (for cp in (CriticalPairs rule1
                                                                          (RenameVariables
                                                                            rule2
                                                                            (Frees rule1)))
                                                   eachtime (printout NIL ExclamationPoint)
                                                            (newrule←(if RuleList
                                                                         then (EvalRules cp)
                                                                       else (EVAL cp)))
                                                   when newrule~=TRUE
                                                   do (if newrule=FALSE
                                                          then (ReportContradictionAndAbort)
                                                       else (printout NIL .TAB0 0 "From" , #
                                                                  (listOneRule rule1:LHS
                                                                               RewriteRuleOp
                                                                               rule1:RHS)
                                                                  "and" ,, #
                                                                  (listOneRule rule2:LHS
                                                                               RewriteRuleOp
                                                                               rule2:RHS)
                                                                  "we obtain" ,)
                                                            (AddRule (DeclareVariables newrule)
                                                                     RuleTypes:Lemma NIL T)
```

```
                                                    (* changes Unchecked)
                                                    ]
                (printout NIL ,, "Affirmed." T)
                T)))
        (if erset
            then                                    (* no limit)
                 rulecount←NIL
          else (AffirmError (if ~ContradictionFound
                                then "Interrupted by Control-E.")
                            NIL T])
```

41

```
(CriticalPairs
  [LAMBDA (r1 r2 onlyOneWay)                        (* D.Musser "28-Jul-79 20:33")

        (* r1 and r2 are rewrite rules with nonoverlapping variable sets; result is list of critical pair equations
        (alpha = beta) corresponding to superpositions of left hand sides of r1 and r2)


    (PROG (nvs cps)
          (nvs←(NonvarSubexpressions r2:LHS))
          (if onlyOneWay
              then nvs←nvs::1)                      (* avoid duplicate consideration of entire r1:LHS with
                                                    entire r2:LHS)
          (cps←(for ex in nvs bind s alpha beta eachtime s←(Unifier r1:LHS ex)
                  when (if s='NotUnifiable
                           then NIL
                         else alpha←(SUBLIS s (SubstWhenEq r1:RHS ex r2:LHS))
                              beta←(SUBLIS s r2:RHS) ~(EQUAL alpha beta))
                  collect <r2:Operator alpha beta>))
          (if ~onlyOneWay
              then cps← < !! cps !(CriticalPairs r2 r1 T)
                        >)
          (RETURN cps])
```

42

```
(ReadRules
  [LAMBDA (fileOrAtomList terminator kind)          (* D.Thompson "19-Jun-80 17:29")
    (PROG (ruleSeq)
          (RETURN (if (ruleSeq←(parse 'ruleSeq fileOrAtomList terminator))
                      then (for rulerec in ruleSeq:rule bind expr
                                collect                 (* compute the proper expression form)
                                       (expr←rulerec:expression)
                                       (if rulerec:expression#
                                           then expr← <EQOP expr rulerec:expression#>
                                         elseif expr:Operator=EQOP
                                         elseif kind MEMB '(axiom lemma) and (Translate expr)
                                                                          :Type='Boolean
                                           then expr← <IFOP expr TRUE FALSE>
                                         else (printout NIL .TAB0 0 "The formula" , #
                                                        (PrettyPrint expr T)
                                                        T "isn't an equation." T)
                                              (AffirmError))
                                       expr])
```

43

```
(ReportContradictionAndAbort
  [LAMBDA NIL                                       (* R.Bates " 2-JAN-79 09:10")
    ContradictionFound←T
    rulecount←NIL
    (PRINTLINES T "Contradiction found." T)
    (ERROR!])
```

44

```
(Rule
  [LAMBDA (lhs% )                                   (* R.Erickson "29-Jan-80 15:58")

        (* * Given the left-hand side of a rule, produces the text of the rule with the proper EqualOp.
        This will work for defns, etc; the global variable KindOfRule gets set.)


    <[if lhs% :Operator:EqualOp
       elseif (Extension lhs% :Operator T)='Interface
```

```
        then                                        (* must compute relevant Equal operator.
                                                    Needn't be undoable.)
             lhs% :Operator:EqualOp←EQOP
      else                                          (* We need to be able to compute the EqualOp here for
                                                    compatibility with old files which didn't happen to
                                                    compute it in time.)
                                                    (* compute in the foreign type.
                                                    Hope we don't get "please declare" errors.)
          lhs% :Operator:EqualOp←(GetEqualOp (TypeOfExpression (Shorten lhs% )
                                                          (Extension lhs% :Operator]
    lhs%
    (RESETVARS ((ListingFlag T)
                (REPORTFLAG NIL)
                (UseRules T))
               (RETURN (APPLY lhs% :Operator lhs% :Arguments)))
    >])
```

45

(addRuleCannot

```
  [LAMBDA (rule auxmessage sentenceStarted)          (* D.Thompson "19-Jun-80 16:58")

        (* * Cannot make it a rule, what now? Returns one of Suppress, Treat, Keep, Instead.)


    (PROG (question keyList)
          (printout NIL (if sentenceStarted
                            then "an e"
                           else "E")
                  "quation which cannot be made into a rewrite rule")
          (for m in (MKLIST auxmessage) finally (printout NIL Colon) do (printout NIL , m))
          (listOneRule rule:LHS RewriteRuleOp rule:RHS)
          (question←"What now, boss?")
          (keyList← <Suppress TreatAsEquation Instead KeepAsIs>)
          (RETURN (AFFIRMUSER NIL NIL question keyList NIL NIL AddRuleOptionList])
```

46

(addRuleDirection

```
  [LAMBDA (rule ruleReadIn sentenceStarted)          (* D.Thompson "29-Oct-80 15:28")

        (* * Computes the proper treatment for the new rule. May be a priori, or may ask user; asks approval if not read in
        or if a priori. Return value:1 is one of (NIL (as is) ,Reverse, Suppress, Treat, Instead (another)) Return value:2 = T
        if user has Not seen/approved this rule.)


    (PROG (leftvars rightvars bothvars diffvars lu ru apriori ask keyList question result1)
          (leftvars←(SORT (FreeVars rule:LHS)))
          (rightvars←(SORT (FreeVars rule:RHS)))
          (lu←(LeftUnifiableWithSomeSubexpression rule:LHS rule:RHS))
                                                     (* will be non-NIL if left hand side is left-unifiable
                                                     with some subexpression of right hand side -
                                                     thus L->R is forbidden)
          (ru←(LeftUnifiableWithSomeSubexpression rule:RHS rule:LHS))
                                                     (* will be non-NIL if right hand side is left-unifiable
                                                     with some subexpression of left hand side -
                                                     forbids R->L)
          [if (EQUAL leftvars rightvars)
              then

        (* * variables the same- would one direction loop?)


                   (if lu and ~ru
                       then apriori←'Reverse
                     elseif ru and ~lu
                       then                          (* ok)
                     elseif lu and ru
                       then ask←(addRuleCannot rule "because it would loop" sentenceStarted)
                                                     (* Returns one of Suppress Treat Keep Instead)
                     else                            (* no unifications; can't force direction)
                          (if ruleReadIn
                              then                   (* accept quietly)
                            else                     (* generated rules must be confirmed for direction)
                                 (printout NIL (if sentenceStarted
                                                   then "a p"
                                                  else "P")
                                         "ossible new rule:" T .SP 5 #
                                         (listOneRule rule:LHS RewriteRuleOp rule:RHS))
```

```
                    question←"Ok? " keyList← <Yes Reverse Suppress TreatAsEquation
                                                Instead>
                    ask←(AFFIRMUSER NIL NIL question keyList NIL NIL AddRuleOptionList)
                                                (* one of Reverse Suppress Yes Treat Instead)
                ))
  else

(* * leftvars ~EQUAL rightvars. This will force our choice of direction. We must check for unifications, unless
ultimate LHS is a variable.)

        bothvars←(INTERSECTION leftvars rightvars)
        diffvars←(Shorten (LDIFFERENCE (UNION leftvars rightvars)
                                       bothvars))
                                                (* for user message only)
        (if (EQUAL bothvars leftvars)
            then                                (* leftvars is proper subset of rightvars , so R->L)
                 (if ru and (LISTP rule:RHS)
                     then ask←(addRuleCannot rule <"because vars" ! diffvars
                                                  "force its reversal, but it would loop"
                                                    >
                                                 sentenceStarted)
                     else apriori←'Reverse)
          elseif (EQUAL bothvars rightvars)
            then                                (* rightvars is proper subset of leftvars ,leave as
                                                L->R)
                 (if lu and (LISTP rule:LHS)
                     then ask←(addRuleCannot rule <"because it would loop, and vars"
                                                   ! diffvars "prohibit reversal" >
                                                 sentenceStarted))
          else (if (ATOM rule:LHS) and (ATOM rule:RHS)
                   then (ReportContradictionAndAbort)
                   else ask←(addRuleCannot rule <"because of variables" ! diffvars>
                                                sentenceStarted]

(* * Assert: apriori is NIL or Reverse. ask is Suppress Treat Reverse Instead Yes or NIL)

        (if apriori and ruleReadIn
            then (printout NIL (if sentenceStarted
                                   then "...Whoops!"
                                   else "")
                               .TAB0 0 "The equation" , # (listOneRule rule:LHS RewriteRuleOp rule:RHS)
                               "can't be made into a rewrite rule without reversing it." T)
                 question←"Ok? " keyList← < ! ReverseAndYes Instead Suppress TreatAsEquation
                                             KeepAsIs>
                 ask←(AFFIRMUSER NIL NIL question keyList NIL NIL AddRuleOptionList)
                                                (* one of Reverse Instead Suppress Treat Keep)
            )
        (result1←(SELECTQ ask
                          ((Keep Yes)
                            NIL)
                          ((Instead Reverse Suppress Treat)
                            ask)
                          (NIL apriori)
                          (AffirmError "Illegal case from addRuleDirection" 'internal)))
        (RETURN <result1 ! ~ask>])
```

47

```
(rulequan
  [LAMBDA (rules context)                       (* R.Erickson "24-Oct-79 13:49")

        (* any NEW variables to arise in rules should be disjoint from each other and from existing vars in context.
        Note:we assume rules will not have variables arise which equal existing ones (or that this is harmless) !)

    (PROG ((f (for v in context:find collect (if (LISTP v)
                                                 then v:1
                                                 else v)))
           sub s)
          (RETURN (for r in rules collect (sub←NIL)
                                          (for v in (FreeVars r) when v ~MEMB f
                                             do (s←(SetupName v (MakeExtension v CurrentType)))
                                                (if s
                                                    then     (* changed name)
                                                         sub← <<v ! s> ! sub> newrulevars←
                                                         <<s> ! newrulevars>
                                                    else newrulevars← <<v> ! newrulevars>))
                                          (SUBLIS sub r])
```

used freely: newrulevars

```
)
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

```
(* ↑)  ]

(RPAQQ RenamingFunctions (AnywhereBound ChooseASymbol DeclareVariables FreeVars FreeVars1 Frees
                                        Frees1 NPrimes NewSSymbol NewSymbolFrom NotThisSymbol
                                        Numerify Primes PrimesToMatch RenameBoundVariables
                                        RenameVariables SetupName TreatFreeAsGiven Variables
                                        Variables1 finds))

(DEFINEQ
```

48

```
(AnywhereBound
  [LAMBDA (vars exp)                                    (* R.Erickson " 9-JAN-79 16:44")
    (AND (LISTP exp)
         (if exp:Operator=QOP
             then (for var in vars thereis (FoundIn var exp:given) or (FoundIn var exp:find))
                  or (AnywhereBound vars exp:expr)
           else (for x in exp:Arguments thereis (AnywhereBound vars x])
```

49

```
(ChooseASymbol
  [LAMBDA (x ext type)                                  (* R.Erickson "23-Feb-81 18:22")

          (* given x'\anything, seek a derived symbol y' suchthat y' in CurrentContext translates via root y to a symbol with
          extension "ext", and type "type". We may need to declare, vie AddDeclarations and /SET y. Return y'\ext)


    (PROG (ypar ye ype)
          (if ~x
              then (CannedMessage 'internalArg T <NIL 'ChooseASymbol >))
          (ypar←(ShortenLessPrimes x))                  (* short root. Seek Ss as needed)
          (while (NotThisSymbol ypar ext type) do ypar←(NewSSymbol ypar))
          (ype←(ExtendName ypar ext))
          (if ~(FASSOC ypar CurrentContext)
              then (AddDeclarations <ypar !(create ExpressionWithType
                                                   Expression ← ype
                                                   Type ← type)
                                    >)
                   (/SET ype ype))
          (ye←(PrimesToMatch ype x))
          (/SET ye ye)
          (RETURN ye])
```

50

```
(DeclareVariables
  [LAMBDA (rule)                                        (* R.Erickson "20-Feb-81 19:07")

          (* The rewriterule may contain variables with any extension, primed and unprimed, and which may not be declared.
          Since the rule is about to be added and thus printed, the user will see these vars  He should be able to type them.
          We declare as necessary (SET already performed.) Return the rule with substitutions performed.)


    (PROG ((vars (Variables rule))
           sub avoid)
          (sub←(for v in vars bind s when s←(SetupName v (MakeExtension v CurrentType)
                                                         vars)
                  collect <v ! s>))
          (RETURN (SUBLIS sub rule])
```

51

```
(FreeVars
  [LAMBDA (x)                                           (* R.Erickson " 2-Nov-79 11:29")

          (* * Returns all unbound variables (as opposed to Skolem constants))


    x←(FreeVars1 x)
    (INTERSECTION x x])
```

52

```
(FreeVars1
```

```
[LAMBDA (x)
  (if (LITATOM x) and ~(x : IsConstant)
      then <x>
    elseif (LISTP x)
      then (if x:Operator=QOP
               then x:free
             elseif x:Operator=ALLOP or x:Operator=SOMEOP
               then (REMOVE x:Arg1 (for y in x:Arguments::1 join (FreeVars y)))
             else (for y in x:Arguments join (FreeVars1 y])
```

53

(Frees

```
[LAMBDA (x)                                          (* edited: "12-SEP-78 20:40")
                                                     (* like FreeVars but returns operators too)
  x←(Frees1 x)
  (INTERSECTION x x])
```

54

(Frees1

```
[LAMBDA (x)                                          (* edited: "13-SEP-78 10:07")
  (if (LITATOM x) and ~(x : IsConstant)
      then <x>
    elseif (LISTP x)
      then (if x:Operator=QOP
               then x:free
             elseif x:Operator=ALLOP or x:Operator=SOMEOP
               then (REMOVE x:Arg1 (for y in x:Arguments::1 join (Frees y)))
             else (PROG (op)
```

(we must distinguish Skolem functions, which we return, from things like IfThenElse. The former have themselves as values, while operators, like sub\Seq, have NOBIND)*

```
                  (op←x:1)
                  (op←(if (EVALV op)=op
                          then <op>
                        elseif (EVALV op)='NOBIND
                          then NIL
                        elseif DTVSCOMPLAIN
                          then (ERROR "Dtvs function atom has bad value:" <op (EVALV op)
                                                                           >)
                        elseif (LISTP op)
                          then (Frees op)
                        else NIL))
                  (RETURN <!! op !(for y in x:Arguments join (Frees y))
                            >])
```

55

(NPrimes

```
[LAMBDA (name)                                       (* R.Erickson "24-Sep-81 16:37")
  (for x on (UNPACK (Shorten name)) until x:1='' or x:1=PRIMEDELIM
     finally (RETURN (if x
                         then (if x:1=''
                                  then (FLENGTH x)
                                else (Numerify x::1 name))
                       else 0])
```

56

(New$Symbol

```
[LAMBDA (old)                                        (* D.Thompson "19-Aug-80 11:15")
                                                     (* given a short name, generate a new one using
                                                     NEWSYMBOLCHAR)
                                                     (* edited: "14-SEP-78 21:08")
  (PROG (unpack result)
        (unpack←(UNPACK old))
        (for ct on unpack bind c eachtime c←ct:1 while c~=NEWSYMBOLCHAR and c~=SingleQuote
           do result← <c ! result> finally result← < !!(DREVERSE result)
                                                     NEWSYMBOLCHAR ! ct>)
        (RETURN (PACK result])
```

57

```
(NewSymbolFrom
  [LAMBDA (x variablesToAvoid)                          (* R.Erickson "24-Feb-81 17:06")

          (* * By adding and removing primes, seek some version of x which is not in variablesToAvoid.)


    (PROG (x1 pre post xi)
          (if ~x
              then (AffirmError "Affirm bug: NIL in NewSymbolFrom" 'internal))
          (pre←(ShortenLessPrimes x))
          (post←(Extension x))
          (for i from 0 do x1←(MakeExtension (PACK <pre !(Primes i)
                                                    >)
                                             post)
             repeatwhile x1 MEMB variablesToAvoid)          (* try to avoid excess primes- start with none)
          (/SET x1 x1)
          (RETURN x1])
```

58

```
(NotThisSymbol
  [LAMBDA (sym ext type)                                (* R.Erickson "25-Feb-81 16:38")
                                                        (* given a short symbol, return T if its interface
                                                        appears in CurrentContext with the wrong extension or
                                                        type)
    (PROG (ewt)
          (ewt←(FASSOC sym CurrentContext):NEW)
          (RETURN (AND ewt (OR ewt:Type~=type (Extension ewt:Expression)
                               ~=ext])
```

59

```
(Numerify
  [LAMBDA (atoms context)                               (* R.Erickson "24-Sep-81 16:36")
                                                        (* list of characters, context variable for error.)

          (* * Given characters which followed PRIMEDELIM in a variable; since the first is not quote, this must be a number,
          as in x#5. Check to make sure the user didn't enter junk, like seq# or inc#a.)


    (if ~atoms
        then (CannedMessage 'illegal T <'variable (Shorten context)
                                        >))
    (PROG (t)
          (t←0)
          (for x in atoms do (if (SMALLP x)
                                 then                   (* add digit on the right)
                                      t←(10*t)+x
                                 else                   (* nonnumeric)
                                      (CannedMessage 'illegal T <'variable (Shorten context)
                                                                 x>)))
          (RETURN t])
```

60

```
(Primes
  [LAMBDA (n)
    (if n leq PRIMELIMIT
        then (to n collect '')
      else <PRIMEDELIM n>])
```

61

```
(PrimesToMatch
  [LAMBDA (root primed)                                 (* R.Erickson "23-Feb-81 18:23")

          (* * root is primeless extended name, which we wish to give as many primes as in the name "primed")


    (PACK <(Shorten root) !(Primes (NPrimes primed))
           TypeSeparator
           ((Extension root) or (CannedMessage 'internalArg T <root 'PrimesToMatch >))
           >])
```

62

```
(RenameBoundVariables
  [LAMBDA (ex variables)                                    (* R.Erickson "10-NOV-78 18:25")
                                                            (* The bound variables of Qexpression ex that also
                                                            appear in list variables are renamed.)
    (PROG (boundVariables s sub bVe avoid)
          (boundVariables← < !(for v in ex:given collect v:1) !(for v in ex:find collect v:1)
                           >)
          (bVe←(for v in boundVariables collect (if (Extension v)
                                                    ~=CurrentType
                                                    then (MakeExtension v CurrentType)
                                                  else v)))
          (avoid←(for v in variables collect (if (Extension v)
                                                 ~=CurrentType
                                                 then (MakeExtension v CurrentType)
                                               else v)))
          (sub←(for v in boundVariables as ve in bVe
                  eachtime                                  (* avoid = variables to avoid, with this extension.
                                                            s = new version of symbol, if different from old.)
                           (s←(SetupName v ve < ! ex:free ! bVe>))
                  when s collect <v ! s>))
          (RETURN (SUBLIS sub ex])
```

63

```
(RenameVariables
  [LAMBDA (ex vars)                                         (* R.Erickson "24-Feb-81 17:11")

          (* The variables in expression ex are renamed, if necessary, to avoid conflict with those in the list vars.
          This is LOCAL IN EFFECT; no declarations are made (since the names may be inappropriate for global use))


    (PROG (exvars avoid)
          (exvars←(Variables ex))
          (avoid← < ! vars ! exvars>)
          (RETURN (SUBLIS (for v in exvars bind nv when v MEMB vars
                             collect (nv←(NewSymbolFrom v avoid))
                                     (avoid← <nv ! avoid>)
                                     (<v ! nv>))
                          ex])
```

64

```
(SetupName
  [LAMBDA (var extended stayfrom)                           (* edited: " 6-Apr-79 08:47")

          (* if necessary (wrong extension or in avoid), changes the name of var, declares it, adds to avoid, and returns new
          name. else NIL. uses + sets free: avoid. var may be a MEMB of stayfrom. Any new symbol will be disjoint from avoid
          and stayfrom)


    (PROG (s (au (< ! avoid ! stayfrom>)))
          (if var~=extended
              then s←(ChooseASymbol var CurrentType (RemoteTypeOf var))
                   (if s MEMB au
                       then s←(NewSymbolFrom s au))
                   avoid← <s ! avoid>
            else (if var MEMB avoid
                     then s←(NewSymbolFrom var au)
                          avoid← <s ! avoid>))
          (RETURN s])
```

65

```
(TreatFreeAsGiven
  [LAMBDA (x)                                               (* edited: "12-SEP-78 16:39")
    (if x:Operator=QOP
        then (if x:free
                 then (create Qexpression
                              given ←(<! x:given !(for v in x:free collect <v>)
                                >)
                              find ←(for v in x:find collect <! v ! x:free>)
                              free ← NIL
                              expr ← x:expr)
               else x)
      else (create Qexpression
                   given ←(for v in (Frees x) collect <v>)
                   find ← NIL
                   free ← NIL
```

```
                expr ← x])
```

66

```
(Variables
  [LAMBDA (ex)                                    (* result is a list of all variables in expression ex)
    (REMOVEDUPLICATES (Variables1 ex])
```

67

```
(Variables1
  [LAMBDA (ex)                                    (* R.Bates "12-FEB-79 15:18")
    (if (LITATOM ex) and ~(ex : IsConstant)
        then <ex>
      elseif (LISTP ex)
        then (if ex:Operator=QOP
                 then <!!(for v in ex:given collect v:1) !!(for v in ex:find collect v:1)
                          ! ex:free>
               else (for a in ex:Arguments join (Variables1 a)))
      else NIL])
```

68

```
(finds
  [LAMBDA (expr qex)                              (* R.Bates "14-Dec-79 14:43")
                                                  (* returns a list of the names of all find vars occuring
                                                  in expr)
    (for v in (FreeVars expr) when (FASSOC v qex:find) collect v])
)
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

```
(* ↑)  ]

(RPAQQ SkolemizationFunctions (AddArg CheckForQexpression CompleteDeps IntroduceQOP MakeQexpressions
                                      Qexpression Qreduction RemoveIfsQ ZapSkolemFunctions))
(DEFINEQ

                                                                                              69

(AddArg
  [LAMBDA (arg expr)
    <expr:Operator arg ! expr:Arguments>])

                                                                                              70

(CheckForQexpression
  [LAMBDA (qex internal)                                        (* R.Erickson "15-Sep-80 17:11")
    (if ~(type? Qexpression qex)
        then (AffirmError <"Not a quantified expression" qex> (if internal
                                                                  then 'internal])

                                                                                              71

(CompleteDeps
  [LAMBDA (sklist ex)                                           (* R.Bates "14-Dec-79 14:43")
                                                                (* given a skolem fn, return list of fn and names of all
                                                                it depends on. sklist is skolem dependencies.)
    (if (LITATOM ex)
        then (PROG (val)
                   (val←(FASSOC ex sklist))
                   (RETURN (if val
                               then (REMOVEDUPLICATES (CompleteDeps sklist val))
                               else <ex>)))
      elseif (LISTP ex)
        then <ex:1 !(for d in ex::1 join (CompleteDeps sklist d)) >
      else <ex>])

                                                                                              72

(IntroduceQOP
  [LAMBDA (x)
    (create Qexpression
            free ←(FreeVars x)
            given ← NIL
            find ← NIL
            expr ← x])

                                                                                              73

(MakeQexpressions
  [LAMBDA (qs env)                                              (* R.Erickson "16-Aug-79 17:00")

          (* * ensure that every element of qs is TRUE or a Qexpression)

    (for q in qs collect (if q=TRUE or (type? Qexpression q)
                             then q
                           else (create Qexpression
                                        expr ← q using env])

                                                                                              74

(Qexpression
  [NLAMBDA (g%  fd%  fr%  ex% )                                 (* R.Bates "28-Jul-80 10:48")

          (* * See Instance for description of format for quantification. We rebind Assumed/Denied since, when we EVAL
          qex:expr, there may be name conflicts which would cause improper simplification.)

    (PROG (fv qex subs Assumed Denied)
          (DECLARE (SPECVARS Assumed Denied))
          [fv←(for v%  in fr% ::1 join (Frees (EVAL v% ]
          (qex←(create Qexpression
                       given ← g%
                       find ← fd%
```

```
                    free ← NIL
                    expr ← ex% ))
        (qex←(RenameBoundVariables qex fv))
        (subs←(for v% in fv when (Extension v% )
                                  ~=CurrentType
                collect <v%  !(ChooseASymbol v%  CurrentType (RemoteTypeOf v% ))
                        >))
        (RETURN (Qreduction qex:given qex:find (SUBLIS subs fv)
                        (SUBLIS subs (EVAL qex:expr])
```

75

```
(Qreduction
  [LAMBDA (givenList findList freeList expression)          (* R.Bates "14-Dec-79 14:43")
    (if (ATOM expression)
        then expression
      elseif expression:Operator=QOP
        then (PROG (g fd fr)
                   (expression←(RenameBoundVariables expression
                                                     < !(for x in givenList collect x:1)
                                                       !(for x in findList collect x:1) ! freeList>))
                   (g←expression:given)
                   (fd←expression:find)
                   (fr←freeList)
                   (for x in expression:free do (if (FASSOC x givenList)
                                                    then fd←(for y in fd collect (AddArg x y))
                                                  elseif (FASSOC x findList)
                                                    then g←(for y in g collect (AddArg x y))
                                                  elseif ~(MEMBER x fr)
                                                    then fr← <x ! fr>))
                   (RETURN (create Qexpression
                                   given ←(< ! givenList ! g>)
                                   find ←(< ! findList ! fd>)
                                   free ← fr
                                   expr ← expression:expr)))
      else (create Qexpression
                   given ← givenList
                   find ← findList
                   free ← freeList
                   expr ← expression])
```

76

```
(RemoveIfsQ
  [LAMBDA (x needNotImpForm)                                (* D.Thompson " 9-Mar-80 18:15")
    (if x:Operator=QOP
        then (if x:expr=TRUE
                 then TRUE
               else (create Qexpression
                            expr ←(RemoveIfs x:expr needNotImpForm) using x))
      else (RemoveIfs x needNotImpForm])
```

77

```
(ZapSkolemFunctions
  [LAMBDA NIL                                               (* R.Bates "29-Jan-80 12:46")

        (* * Whenever the current expression changes, we are called. This is because the arity of skolem fns may vary
        between propositions)


    (if SkolemFunctions
        then (for f in SkolemFunctions do (/PUTD f NIL))
             (/SET 'SkolemFunctions NIL])
)

(RPAQQ NoChangeFunctions (AddLEntry AddSpecifiedLHS InitialLHSGraph MoreGeneralExprs MoreGeneralThan
                                    NilLHS ReplaceVarsWith UnspecifiedLHSides))
(DEFINEQ
```

78

```
(AddLEntry
  [LAMBDA (exp status)                                      (* R.Erickson "21-Sep-81 18:46")
                                                            (* exp an expression with NIL for vars, status an
                                                            :LStatus value. Smashes LGraph freely, see
                                                            UnspecifiedLHSides.)
```

```
(* * Ensure that exp has an entry in LGraph. If status is non-NIL, set :LStatus to it. Ensure that all parents have
:LStatus = 'general. If entry is new, create siblings siblings with :LStatus = NIL. Caller guarantees that exp's args
are made of only constructor fns and NIL. We also know that no parents have :LStatus = 'exact.)

(PROG (old parentTuples sibInsertions)
      (if old←(SASSOC exp LGraph)
          then
               (if status                                    (* ensure proper status)
                   then (old:LStatus←status))
        else

(* * Add self)

            (push LGraph (create LGraphEntry
                                 LExpr ← exp
                                 LStatus ← status)))

(* * Parents, siblings)

      (parentTuples←(MoreGeneralExprs exp))                  (* reentrant triples. See MGE for format.
                                                             We will smash.)

(* * Ensure parents are there, with proper :LStatus.)

      (for tuple in parentTuples do (AddLEntry (COPYALL tuple:1)
                                               'general))
      (if old
        else

(* * Generate and add siblings, which will be different for each parent.)

            (for tuple in parentTuples bind constrs oper do
                                                             (* :3 is the function removed from us to make parent.
                                                             What other constructors might have been used?)
                                                                  (oper←tuple:3:Operator)
                                                                  (constrs←(Constructors
                                                                      (RangeType oper)))
                                                                  ((MEMB oper constrs) or (SHOULDNT))
                                                             (* caller lied)
                                                                  (constrs←(REMOVE oper constrs))
                                                             (* for siblings)
                                                                  (for c in constrs
                                                                     do

(* create a sibling by smashing our common site in parent. Which site is tuple:2:1. Remember to set back when done,
since this is possibly a shared copy.)

                                                                        (tuple:2:1←(NilLHS c))
                                                                        (AddLEntry (COPYALL tuple:1)
                                                                                   NIL)
                                                             (* :1 reflects the change, sibling is unrelated.)
                                                                        )
                                                                  (tuple:2:1←NIL)
                                                             (* restore)])
```

79

```
(AddSpecifiedLHS
  [LAMBDA (lhs)                                              (* R.Erickson "21-Sep-81 19:38")
                                                             (* lhs is an expression. We smash LGraph freely.)

(* * lhs corresponds to the left-hand side of a rule. If it is built on non-constructor functions, we ignore it.
If it is less general than an existing "exact" entry, we also do nothing. If there are already entries less general
than it, we remove them. It is added, with :LStatus "exact", and any needed "general" parents and "NIL" siblings are
inserted. See UnspecifiedLHSides for format of graph.)

    (PROG (nilExp entry)
          (nilExp←(ReplaceVarsWith lhs NIL))

(* * already in LGraph?)
```

```
    (if (NonConstructorsUsed lhs)
        then                                          (* skip)
      elseif entry←(SASSOC nilExp LGraph)
        then                                          (* Check status. If there are existing, less general
                                                      entries, they must be pruned out;
                                                      uncommon.)
             (if entry:LStatus='gereral
                 then                                 (* prune)
                      LGraph←(for lg in LGraph unless (MoreGeneralThan nilExp lg:LExpr)
                                collect lg))
             entry:LStatus←'exact
      else                                            (* not in the graph. We need to add self, siblings, and
                                                      parents. All will have the usual :LStatus.)
           entry←(AddLEntry nilExp 'exact))
    (RETURN entry])
```

80

(InitialLHSGraph

```
  [LAMBDA (fn)                                        (* R.Erickson "21-Sep-81 14:12")
                                                      (* fn extended)

    (* * Return a graph with a single, most general entry. See UnspecifiedLHSides for format.)


    <(create LGraphEntry
             LStatus ← NIL
             LExpr ←(NilLHS fn))
    >])
```

81

(MoreGeneralExprs

```
  [LAMBDA (expr)                                      (* R.Erickson "22-Sep-81 14:54")
                                                      (* an expression with NIL for vars)

    (* * Compute those expressions which are one step more general than expr. Returns a list of triples.
    :1 is unique, and has NIL in place of a single function call in expr:Arguments. :2 points to the tail in :1
    beginning with than NIL. :3 is the funcion and args which were removed; since we omit only one call at a time, its
    arguments will be NIL. -
    Thus, triple:2:1←triple:3 implies triple:1 equal expr. ← Note that we may return nothing, and never change
    expr:Operator.)


    (for topTail on expr:Arguments bind arg prefix suffix genArg ref when arg←topTail:1
       join

    (* * for each nontrivial argument arg, compute generalizations. First, the common stuff.)


      (prefix←(for tail on expr until tail=topTail collect tail:1))
                                                      (* all stuff before arg)
      (suffix←topTail::1)                             (* all stuff after arg.)

    (* * Assert: (EQUAL expr <!prefix arg !suffix>))


    (* * If arg is not a simple fn call, recursion will do the work.)


      (if genArg←(MoreGeneralExprs arg)
          then                                        (* generalized; translate its tuples)
               (for tup in genArg collect << ! prefix tup:1 ! suffix> tup:2 tup:3>)
        else                                          (* do it here, only one generalization: <1 NIL...> to
                                                      NIL)
             ref← <NIL ! suffix>                      (* tail for reentrant pointer)
             <<< ! prefix ! ref> ref arg>>])          (* list of one tuple)
```

82

(MoreGeneralThan

```
  [LAMBDA (exp1 exp2)                                 (* R.Erickson "17-Sep-81 19:28")
                                                      (* both expressions have NIL instead of vars, so we
                                                      don't worry about variable names.)

    (* * Can exp1 be obtained from exp2 by replacing one or more function calls with NIL? Eg. either exp1 is NIL or they
```

are similar function calls, differing in at least one arg by a generalization.)

```
    (OR ~exp1 (AND (LISTP exp1)
                   (LENGTH exp1)=(LENGTH exp2)
                   exp1:Operator=exp2:Operator ~(EQUAL exp1 exp2)
                   (for a1 in exp1 as a2 in exp2 always (EQUAL a1 a2) or (MoreGeneralThan a1 a2])
```

83

(NilLHS

```
  [LAMBDA (fn)                                          (* R.Erickson "22-Sep-81 22:32")
                                                        (* fn extended atom)
```

(* Returns <fn NIL NIL...> to the right number of args. Used in the "nochange" package.)*

```
    (PROG (lefts nils)
```

(* Depends on the fact that there is exactly one LHS for interfaces. The interface has one more arg than the function, as a catch for excess args.)*

```
          (lefts←(LeftHandSides (ExtendName fn IEXT)))
          (if ~lefts
              then (AffirmError <"no interface" fn> 'internal))
          (nils←(to (LENGTH lefts:1:Arguments)+ -1 collect NIL))
          (RETURN <fn ! nils>])
```

84

(ReplaceVarsWith

```
  [LAMBDA (exp new)                                     (* R.Erickson "17-Sep-81 19:54")
```

(* any atoms in non-functional positions in exp are replaced with new, even top-level. Like UCI\SUBST, bad name, except target is all vars.)*

```
    (if (LISTP exp)
        then <exp:Operator !(for arg in exp:Arguments collect (ReplaceVarsWith arg new)) >
      else new])
```

85

(UnspecifiedLHSides

```
  [LAMBDA (fn)                                          (* R.Erickson "28-Sep-81 19:11")
                                                        (* fn an extended atom.)
```

(* Given a function name, returns a list of type-correct left-hand sides. These are the most-general lhsides which may not be unified (in either direction) with any lhsides of user-supplied rules. In this computation we ignore any lhside whose arguments contain con-constructor functions. Maybe we print a message for these.)*

```
    (PROG (specified LGraph unspecs withVars)           (* LGraph is used freely by the graph fns)
          (specified←(LeftHandSides fn))
          (if ~specified
              then                                      (* no rules at all. Caller lied.)
                   (SHOULDNT))
          (LGraph←(InitialLHSGraph fn))
          (for lhs in specified do (AddSpecifiedLHS lhs))
          (unspecs←(for entry in LGraph unless entry:LStatus collect entry:LExpr))
          (withVars←(for exp in unspecs collect (GeneralLHS exp)))
          (RETURN withVars])
)

(RPAQQ PRIMEDELIM #)

(RPAQ PRIMELIMIT 2)

(RPAQQ NEWSYMBOLCHAR $)

(RPAQ ShortenLessPrimesList (LIST 92 39 (CHCON1 PRIMEDELIM)))

(RPAQ LeftUnification NIL)

(RPAQQ Accept (Accept " another equation from the terminal instead"))

(RPAQ AddRuleOptionList NIL)
```

```
(RPAQQ Instead (Instead " accept another equation from the terminal"))

(RPAQQ KeepAsIs (Keep " it as it is (this may result in a stack overflow!)"))

(RPAQQ Reverse (Reverse " it"))

(RPAQQ ReverseAndYes ((Reverse " it")
                      (Yes ", reverse it" RETURN 'Reverse)))

(RPAQQ Suppress (Suppress " it (put it on the list %"BadEquations%")"))

(RPAQQ TreatAsEquation (Treat " it as Equation == TRUE"))

(RPAQQ Yes (Yes NIL))
(DECLARE: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS

(ADDTOVAR NLAMA )

(ADDTOVAR NLAML Qexpression)

(ADDTOVAR LAMA )
)
(DECLARE: DONTCOPY
  (FILEMAP (NIL (1774 4070 (EqualityFromIf 1786 . 3181) (EqvWithQex 3185 . 3336) (GetEqualOp 3340 . 3531
) (ReverseEquality 3535 . 3695) (affirmed? 3699 . 4067)) (4450 11662 (AddHypothesis 4462 . 4801) (
ApplyAllEqHyps 4805 . 6345) (ApplyEqHyp 6349 . 7043) (CheckIntroducedExpr 7047 . 8235) (CompleteSubs
8239 . 8700) (ConstOrder 8704 . 9050) (EqvFromIf 9054 . 9569) (GetConjunct 9573 . 9979) (
ListOfConjuncts 9983 . 10169) (NonvarSubexpressions 10173 . 10478) (Operators 10482 . 10738) (PutForm
10742 . 11163) (SecondaryOperators 11167 . 11420) (operators1 11424 . 11659)) (11879 14399 (
LeftUnifiableWithSomeSubexpression 11891 . 12153) (LeftUnifier 12157 . 12448) (OccursUsingBindings
12452 . 12895) (Unifier 12899 . 13186) (Unify1 13190 . 14122) (UnifyWithSubExpressions 14126 . 14396))
 (14565 17361 (Chainings 14577 . 14730) (Chainings1 14734 . 15142) (ChooseChainingsAndNarrowings 15146
 . 16192) (TryChainingsAndNarrowings 16196 . 17358)) (17559 24052 (CircularSubs 17571 . 18183) (
CircularSubs1 18187 . 19267) (Instance 19271 . 22122) (Instance? 22126 . 22249) (InstanceQ 22253 .
22376) (InvalidDependencies 22380 . 23319) (SkolemizeProperly 23323 . 24049)) (24304 45000 (
AcceptNewRules 24316 . 25204) (AddAutoRules 25208 . 26348) (AddRule 26352 . 31324) (Completer 31328 .
35060) (CriticalPairs 35064 . 36090) (ReadRules 36094 . 36954) (ReportContradictionAndAbort 36958 .
37190) (Rule 37194 . 38423) (addRuleCannot 38427 . 39200) (addRuleDirection 39204 . 44094) (rulequan
44098 . 44997)) (45361 57452 (AnywhereBound 45373 . 45777) (ChooseASymbol 45781 . 46792) (
DeclareVariables 46796 . 47526) (FreeVars 47530 . 47781) (FreeVars1 47785 . 48189) (Frees 48193 .
48454) (Frees1 48458 . 49519) (NPrimes 49523 . 49878) (New$Symbol 49882 . 50563) (NewSymbolFrom 50567
. 51275) (NotThisSymbol 51279 . 51764) (Numerify 51768 . 52666) (Primes 52670 . 52800) (PrimesToMatch
52804 . 53215) (RenameBoundVariables 53219 . 54413) (RenameVariables 54417 . 55089) (SetupName 55093 .
 55875) (TreatFreeAsGiven 55879 . 56455) (Variables 56459 . 56637) (Variables1 56641 . 57105) (finds
57109 . 57449)) (57680 62128 (AddArg 57692 . 57769) (CheckForQexpression 57773 . 58046) (CompleteDeps
58050 . 58696) (IntroduceQOP 58700 . 58842) (MakeQexpressions 58846 . 59226) (Qexpression 59230 .
60198) (Qreduction 60202 . 61365) (RemoveIfsQ 61369 . 61709) (ZapSkolemFunctions 61713 . 62125)) (
62292 72294 (AddLEntry 62304 . 65082) (AddSpecifiedLHS 65086 . 66737) (InitialLHSGraph 66741 . 67143)
(MoreGeneralExprs 67147 . 69127) (MoreGeneralThan 69131 . 69873) (NilLHS 69877 . 70639) (
ReplaceVarsWith 70643 . 71086) (UnspecifiedLHSides 71090 . 72291)))))
STOP
```