<AFFIRM>FORMULAIO..24 30-Sep-81 15:28:12

```
(FILECREATED "26-Sep-81 16:33:47" <AFFIRM>FORMULAIO..24 64388

     changes to:  listRewriteRules listTypeParts CannedMessage listTypeAutos

     previous date: "31-Mar-81 20:42:28" <AFFIRM>FORMULAIO..23)


(PRETTYCOMPRINT FORMULAIOCOMS)

(RPAQQ FORMULAIOCOMS [(FNS * FORMULAIOFNS)
        (VARS (EqConsFlg 2)
              (InterfaceProcessing NIL)
              (InternalOperatorNames <<'= EQIO> <'+ ADDIO> <'- DIFFIO> <'< LTIO> <'> GTIO> <'*
                                     MULTIO> <'/ DIVIO> <'& ANDIO> <'~ NOTIO> <'! NOTIO> <'%| ORIO>
                                     <'~= NEIO> <'<= LEIO> <'>= GEIO> <'notEqual NEIO>
                                     <'lessThanEqual LEIO> <'greaterThanEqual GEIO>>)
              (PatternDebug NIL)
              (PropStorage (create PropStorage))
              (RewriteRuleOp "==")
              StatusCategories)
        (RECORDS TerminalEnvironment AFFIRMEnvironment)
        (ADDVARS (GLOBALVARS RewriteRuleOp))
        (* error messages!)
        (FNS CannedMessage)
        (BLOCKS (EqConsBLOCK EqCons EqCons1 (ENTRIES EqCons)
                             (GLOBALVARS EqConsList temp)
                             (NOLINKFNS . T])

(RPAQQ FORMULAIOFNS (AddNeeds CompactPropStorage DefnRangeConvert EqCons EqCons1 FindIHs
                              FindSubexpressions FitsOnLine GetExpression GetProp# Loaded?
                              MapToInternal PrettyPrint PrettyPrintCommand PrettyPrintHistory
                              PrintArgs PrintCurrentProposition PrintList ReadExpression
                              ReadExpressionList TranslateTo# UseNumbers decipherRuleMatchInput
                              getDeclarations getFileNameFromDirectory getInfixOps getInterfaces
                              getOneRuleMatch getRewriteRules getRuleMatches getViewableElements
                              initNeeds listDeclarations listInfixOps listInterfaces listNeeds
                              listOneRule listRewriteRules listRuleMatches listTypeAutos
                              listTypeParts loadNeededFiles loadNeededTypes makeExpFromPattern
                              makePatterns needReport needs parenthesize parse printHelper))

(DEFINEQ
```

1

```
(AddNeeds
  [LAMBDA (object elements typeName implicitNeeds)          (* D.Thompson "22-Feb-80 13:47")

          (* * This routine adds elements to the needs data structure.)


    (PROG (oldValue newValue (typeNeeds (<NIL>)))
          (elements←(MKLIST elements))
          (if implicitNeeds
              then elements←(LDIFFERENCE elements <typeName ! PredefinedNames:Types>))
          (if elements
              then typeNeeds←(if typeName:Needs
                                 else typeNeeds)
                   oldValue←(FASSOC object typeNeeds)
                   (if oldValue
                       then oldValue←oldValue::1)
                   newValue←(REMOVEDUPLICATES (if object='Types
                                                  then (for type in < ! oldValue ! elements>
                                                          collect (AFFIRMSpellingCorrect type
                                                                                         KnownNames:Types
                                                                                                 type T T))
                                                else < ! oldValue ! elements>))
                   (PUTASSOC object newValue typeNeeds)
                   typeName:Needs←(if typeNeeds:1=NIL
                                      then typeNeeds::1
                                    else typeNeeds])
```

2

```
(CompactPropStorage
  [LAMBDA (expr)                                          (* R.Bates "30-Oct-79 16:48")
    (if (NUMBERP EqConsFlg)
        then (if (ILEQ EqConsFlg 0)
                 then NIL
               elseif EqConsFlg=1
                 then (EqCons expr)
```

```
        else

    (* It is very important that EqCons be called with expr last and not first since the conses in the first part are
    already in PropStorage.)

                (EqCons <(for i from 1 to EqConsFlg-1 as prop on PropStorage:ExprTo#
                           join <prop:1:1>)
                        expr>))
    else (if EqConsFlg
             then (EqCons expr])
```

3

```
(DefnRangeConvert
  [LAMBDA (pairs)                                                    (* R.Erickson "29-Jan-81 16:18")

    (* * Converts a list of DefnRangePairs (used in swap and invoke) into something printable)

    (* Input pairs have form (fn range), where range is NIL or list of (elem NIL) or (lwb upb); fn and range cannot both
    be NIL.)

    (* * Output is of form "f|1:3,5|,g,|-256|")

   (Separate (for spec in pairs bind ranges
                collect (if spec:2
                            then                                     (* computes form of %|stuff%|)
                                 ranges←(Separate (for rpair in spec:2
                                                     collect (if (NLISTP rpair)
                                                                 then (CannedMessage 'format T
<'DefnRangeConvert >))
                                                                 (if rpair:2
                                                                     then <rpair:1 Colon rpair:2>
                                                                   else rpair:1))
                                                  T))
                                 (< !(if spec:1
                                         then <spec:1>)
                                    !(if spec:2
                                         then <'%| ! ranges '%| >)
                                  >))
             T])
```

4

```
(EqCons
  [LAMBDA (Expr)                                                     (* R.Bates "26-Oct-79 12:51")
    (PROG (EqConsList temp)
          (EqCons1 Expr])
```

5

```
(EqCons1
  [LAMBDA (Expr)                                                     (* R.Bates "17-Jul-79 16:51")
    (for part on Expr do (if (LISTP part:1)
                             then (if temp←(MEMBER part:1 EqConsList)
                                      then (part:1←temp:1)
                                    else EqConsList←(ATTACH part:1 EqConsList)
                                         (EqCons1 part:1)))
                         (if (LISTP part::1)
                             then (if temp←(MEMBER part::1 EqConsList)
                                      then (part::1←temp:1)
                                    else EqConsList←(ATTACH part::1 EqConsList])
```

6

```
(FindIHs
  [LAMBDA (expression)                                               (* D.Thompson "24-Jun-80 17:54")
    (if (LISTP expression)
        then (if (type? Qexpression expression)
                 then (FindIHs expression:expr)
               elseif expression:Operator=IH2OP
```

```
        then expression
      else (for arg in expression:Arguments join (FindIHs arg])
```

7

```
(FindSubexpressions
  [LAMBDA (operator expression)                    (* D.Thompson "23-Jul-80 12:15")

        (* * This routine collects all the subexpressions of its argument expression that have operator as the main
        operator.)


    (if (LISTP expression)
        then (if (type? Qexpression expression)
                 then (FindSubexpressions operator expression:expr)
             elseif expression:Operator=operator
               then <expression>
             else (for arg in expression:Arguments join (FindSubexpressions operator arg])
```

8

```
(FitsOnLine
  [LAMBDA (expression lineLength)                  (* D.Thompson "25-Jun-80 10:34")

        (* * This routine determines if the expression provided as the first parameter will fit on a line whose length is
        given by the second parameter.)


        (* * Currently this routine estimates roughly, rather than actually computing. It should always estimate so that
        something that won't fit is never said to fit, but something that does fit may be rejected.)


    (if (5*(ElementsIn expression)) lt (4*(lineLength/MagicCount))
        then T
      else NIL])
```

9

```
(GetExpression
  [LAMBDA (p#)                                     (* R.Erickson "23-Jul-79 13:03")
                                                   (* dummy)
    (if (FASSOC p# PropStorage:#ToExpr)::1
      else (AffirmError <"unknown prop#" p#> 'internal])
```

10

```
(GetProp#
  [LAMBDA (expr)                                   (* R.Erickson " 7-Aug-79 17:03")
                                                   (* given expr or #)

        (* * yields a prop#, >= 0)


    (if (FIXP expr)
        then expr
      else (PROG (n)
                 (n←(TranslateTo# expr))
                 (if (MINUSP n)
                     then n←(-n)
                          (MakeNode n))
                 (RETURN n])
```

11

```
(Loaded?
  [LAMBDA (name object)                            (* R.Erickson "19-Mar-80 18:53")

        (* * This routine determines whether a particular object has been already loaded or not. If not, it returns NIL;
        if so, it returns the object's name.)


    (SELECTQ object
             [Files (PROG (unpak filedates)
                          (unpak←(UNPACKFILENAME name))
                          (filedates←(GETPROP (U-CASE (LISTGET unpak 'NAME))
```

```
                                        'FILEDATES))
                        (RETURN (AND (if (LISTGET unpak 'VERSION)
                                         then                    (* If version is supplied, then we want an EXACT name
                                                                    match with the currently loaded file.
                                                                    Maybe this is too stringent!)
                                             name=filedates:1::1
                                       else                      (* just anything with this root name)
                                            filedates)
                                     name]
                    (Types (AFFIRMSpellingCorrect name KnownNames:Types NIL T T))
                    (Unexpected 'AFFIRMObject])
```

12

(MapToInternal

```
  [LAMBDA (symbol unaryOp)                                  (* D.Thompson " 4-Oct-79 12:20")
    (PROG (newName extension)
          (extension←(Extension symbol))
          (if newName←(CADR (FASSOC (Shorten symbol)
                                    InternalOperatorNames))
              then (if newName=DIFFIO and unaryOp
                       then newName←NEGIO)
                   (RETURN (if extension
                               then (ExtendName newName extension)
                             else newName])
```

13

(PrettyPrint

```
  [LAMBDA (x sameLine usePrettyNorm)                        (* D.Thompson " 9-Mar-80 18:29")
    x←(if usePrettyNorm
          then (getPrettyNorm)
        else (RemoveIfs x))
    (if sameLine
        then (INFIX\PRINT3 x)
      else (INFIX\PRINT x])
```

14

(PrettyPrintCommand

```
  [LAMBDA (List File ReadTable)                             (* D.Thompson " 3-Jan-80 16:21")
    (PROG (break)
          (if (LISTP List)
              then break←(GETBRK ReadTable)
                   (printout File List:1 ,)
                   (for (atom first second) on List::1 do (printout File atom:1)
                                                          (if ~(second←((NCHARS atom:2)=1
                                                                        and (CHCON1 atom:2)
                                                                            MEMB break)
                                                                   or first)
                                                              then (printout File ,))
                                                          (first←second)
                      first (first←((NCHARS atom:1)=1 and (CHCON1 atom:1)
                                                          MEMB break)))
                   (if (LISTP List)
                       then (printout File T))
            elseif List
              then (printout File List T])
```

15

(PrettyPrintHistory

```
  [LAMBDA NIL                                               (* D.Thompson "18-Mar-81 11:54")
    (for event in LISPXHISTORY:1::1 as n from LISPXHISTORY:2-1 by -1 bind eventNumber
       do (eventNumber←(if n gt 0
                           then n
                         else n+LISPXHISTORY:4))
          (if event:2=Prompt or event:2=BatchErrorPrompt
              then                                          (* AFFIRM command)
                   (printout NIL .TAB0 0 .I4 eventNumber , event:2 , # (PrettyPrintCommand
                             (if (LISTP event:1:-1)
                                 then event:1:-1
                               else event:1)
                             NIL PascalReadTable)
                           T)
            else                                            (* lisp command)
                 (printout NIL .TAB0 0 .I4 eventNumber , Prompt , "e" , .PPVTL event:1 T])
```

16

```
(PrintArgs
  [LAMBDA (argList separator terminator firstPosition)       (* D.Thompson " 6-Nov-80 20:19")
    (PROG (lineLength)
          (lineLength←(LINELENGTH))
          (if separator=T
              then separator←", ")
          (if (FIXP firstPosition)
            else firstPosition←0)
          (for arg on (MKLIST argList) by arg::1 finally (if terminator
                                                             then (printout NIL terminator))
             do (if (FitsOnLine arg:1 lineLength-(POSITION))
                  else (printout NIL .TAB firstPosition))
                (if (LISTP arg:1)
                    then (printout NIL .PPVTL arg:1)
                  else (printout NIL arg:1))
                (if (LISTP arg::1) and separator
                    then (printout NIL separator])
```

17

```
(PrintCurrentProposition
  [LAMBDA (printFlag)                                        (* D.Thompson "28-Oct-80 12:48")
    (if CurrentPropn~=TRUE or printFlag=T
        then (PrettyPrint CurrentPropn NIL T)
             (if (AnyInnerIfs CurrentPropn:expr NIL)
                 then (printout NIL .TAB0 0 "(The 'cases' command is applicable)" T])
```

18

```
(PrintList
  [LAMBDA (lst)
    (for x on lst do (INFIX\PRINT3 x:1)
                     (if x::1
                         then (PRINTLINES ","])
```

19

```
(ReadExpression
  [LAMBDA (fileorAtomList terminator dontAbort)              (* D.Thompson " 3-Nov-79 13:05")

          (* * This routine parses and then type-checks one expression.)


    (PROG (ex)
          (if (ex←(parse 'expression fileorAtomList terminator)) and ex←(pexec ex)
              then (RETURN ex)
            elseif dontAbort
              then (RETURN NIL)
            else (AffirmError])
```

20

```
(ReadExpressionList
  [LAMBDA (fileOrAtomList terminator)                        (* D.Thompson " 2-Nov-79 13:37")

          (* * This routine parses a non-empty sequence of expressions. It performs an error exit if a syntax or interface
          error occurs.)


    (PROG (exlist)
          (if (exlist←(parse 'expressionSeq fileOrAtomList terminator))
              then [RETURN (for ex in exlist:expression collect (if (typeify ex)
                                                                    else (AffirmError]
            else (AffirmError])
```

21

```
(TranslateTo#
  [LAMBDA (expr)
    (CLISP: UNDOABLE)                                        (* R.Bates "24-Oct-79 18:53")

          (* dummy; will be used for swapping. Return -n if new. We keep #ToExpr as (... (# . expr) ...) and ExprTo# as
```

(... (expr . #) ...))

```
(if (SASSOC expr PropStorage:ExprTo#)::1
  else (PROG (n)
             (CompactPropStorage expr)
             (n←(add PropStorage:Top# 1))
             (push PropStorage:ExprTo# (<expr ! n>))
             (push PropStorage:#ToExpr (<n ! expr>))
             (RETURN (-n])
```

22

(UseNumbers

```
  [LAMBDA (lst)                                    (* R.Erickson " 7-Aug-79 17:06")
    (for el in (MKLIST lst) collect (GetProp# el])
```

23

(decipherRuleMatchInput

```
  [LAMBDA (ratoms)                                 (* D.Thompson " 1-Feb-80 08:18")
```

(* This routine takes a list of atoms (presumably just read from input) and returns the three pieces parts, typeSet, and lhs, where lhs is a parsed expression sequence, typeSet is a possibly empty list of corrected type names, and parts is a possibly empty list of corrected type part names.)*

```
    (PROG (correctedName interfaces lhs lhsName P partName parts temp typeName typeSet)
          (if PatternDebug
              then (printout NIL .TAB0 0 "debug:" 10 "whole input list =" , .PPVTL ratoms T))
          (if lhsName←(for ele in ratoms thereis (LITATOM ele) and (L-CASE ele)='lhs)
              then (if PatternDebug
                       then (printout NIL .TAB0 0 "debug:" 10 "lhsName =" , lhsName T))
                   P←(ARRAY 3)
                   (ELT P 1)←(lhsName MEMB ratoms)
                   (if PatternDebug
                       then (printout NIL .TAB0 0 "debug:" 10 "P1 =" , .PPVTL (ELT P 1)
                                  T))
                   (if typeName←(for ele in ratoms thereis (LITATOM ele) and (L-CASE ele)
                                                                          MEMB '(type types))
                       then ((ELT P 2)←(typeName MEMB ratoms))
                     else ((ELT P 2)←NIL))
                   (if PatternDebug
                       then (printout NIL .TAB0 0 "debug:" 10 "P2 =" , .PPVTL (ELT P 2)
                                  T))
                   (if partName←(for ele in ratoms thereis (LITATOM ele) and (L-CASE ele)
                                                                          MEMB '(part parts))
                       then ((ELT P 3)←(partName MEMB ratoms))
                     else ((ELT P 3)←NIL))
                   (if PatternDebug
                       then (printout NIL .TAB0 0 "debug:" 10 "P3 =" , .PPVTL (ELT P 3)
                                  T))
                   [if (ELT P 2) or (ELT P 3)
                       then (if PatternDebug
                                then (printout NIL .TAB0 0 "debug:" 10 "sorting..." T))
                            (for i from 1 to 2
                               do (for j from i+1 to 3
                                     do (if PatternDebug
                                            then (printout NIL .TAB0 0 "debug:" 10 "comparing" , i ,
                                                       "and"
                                                       , j , "..."))
                                        (if (TAILP (ELT P i)
                                                   (ELT P j))
                                              or (ELT P i)=NIL and (ELT P j)
                                                                   ~=NIL
                                            then (if PatternDebug
                                                     then (printout NIL .TAB0 0 "debug:" 10
                                                                "swapping."
                                                                T))
                                                 temp←(ELT P i)
                                                 (ELT P i)←(ELT P j)
                                                 (ELT P j)←temp
                                          else (if PatternDebug
                                                   then (printout NIL .TAB0 0 "debug:" 10 "order OK."
                                                              T]
                   (if PatternDebug
                       then (printout NIL .TAB0 0 "debug:" 10 "after sort:" T 10 "P1 =" , .PPVTL
                                  (ELT P 1)
                                  T 10 "P2 =" , .PPVTL (ELT P 2)
```

```
                                T 6 "and P3 =" , .PPVTL (ELT P 3)
                                T))
                        [for i from 1 to 2 do ((ELT P i)←(LDIFF (ELT P i)
                                                                (ELT P i+1]
                        (if PatternDebug
                            then (printout NIL .TAB0 0 "debug:" 10 "after splitting:" T 10 "P1 =" , .PPVTL
                                        (ELT P 1)
                                        T 10 "P2 =" , .PPVTL (ELT P 2)
                                        T 6 "and P3 =" , .PPVTL (ELT P 3)
                                        T))
                        [for i from 1 to 3 do (if (ELT P i)
                                                  then (SELECTQ (L-CASE (ELT P i):1)
                                                                (lhs lhs← (ELT P i)
                                                                     ::1)
                                                                ((part parts)
                                                                  parts←
                                                                  (ELT P i)
                                                                  ::1)
                                                                ((type types)
                                                                  typeSet←
                                                                  (ELT P i)
                                                                  ::1)
                                                                (SHOULDNT]
                        (if PatternDebug
                            then (printout NIL .TAB0 0 "debug:" 10 "after breakout:" T 10 "lhs =" , .PPVTL
                                        lhs T 10 "parts =" , .PPVTL parts T 6 "and typeSet =" , .PPVTL
                                        typeSet T))
                        (if lhs←(parse 'expressionSeq lhs CommandTerminator)
                            then lhs←(makePatterns (MKLIST lhs:expression))
                          else (AffirmError))
                        typeSet←(for type in typeSet eachtime correctedName←(AFFIRMSpellingCorrect type
                                                                                    KnownNames:Types NIL
                                                                                                T NIL)
                                    collect correctedName when correctedName)
                        parts←(for part in parts eachtime correctedName←(AFFIRMSpellingCorrect part
                                                                              KnownNames:TypeParts NIL
                                                                                           T NIL)
                                  collect correctedName when correctedName)
                        (if 'interface MEMB parts
                            then interfaces←T
                                 parts←(REMOVE 'interface parts))
                        (if PatternDebug
                            then (printout NIL .TAB0 0 "debug:" 10 "returning:" T 10 "typeSet =" , .PPVTL
                                        typeSet T 10 "parts =" , .PPVTL parts T 10 "lhs =" , .PPVTL lhs
                                        T 6 "and interfaces =" , (if interfaces
                                                                     then "true"
                                                                   else "false")
                                        Period T))
                        (RETURN <typeSet parts lhs interfaces>)
                    else (AffirmError "There's no lhs specification."])
```

24

(getDeclarations

```
[LAMBDA (typeName longNames)                    (* D.Thompson "17-Dec-79 11:05")
                                                (* given a list of local decls of the form
                                                (x\Interface EWT x\Stack Integer), gather together those
                                                with the same type.)

  (PROG ((groups (<NIL>)))
        (for vt in typeName:LocalDeclarations bind t eachtime t←vt::1:Type
           do (PUTASSOC t < !(FASSOC t groups)::1 (if longNames
                                                      then vt::1:Expression
                                                    else (Shorten vt:1))
                        >
                      groups))
        (RETURN groups::1])
```

25

(getFileNameFromDirectory

```
[LAMBDA (name directory object fileCategory IO)          (* D.Thompson " 5-Sep-80 20:14")

        (* * This routine obtains file names from directories according to the object type of the file contents.)


  name←(U-CASE name)
  directory←(U-CASE directory)
  (SELECTQ object
           (Types (SELECTQ fileCategory
```

```
                    (Compiled (getFileName <'DIRECTORY directory 'NAME name 'EXTENSION
                                            (PACK* AFFIRMVERSION 'COM) >
                                           NIL NIL IO))
                    (OldCompiled (getFileName <'DIRECTORY directory 'NAME name 'EXTENSION
                                               'COM >
                                              NIL NIL IO))
                    (Saved (getFileName <'DIRECTORY directory 'NAME name 'EXTENSION
                                         (PACK* AFFIRMVERSION 'LISP) >
                                        NIL NIL IO))
                    (OldSaved (getFileName <'DIRECTORY directory 'NAME name 'EXTENSION NIL>
                                           NIL NIL IO))
                    (Source (getFileName <'DIRECTORY directory 'NAME name 'EXTENSION 'AXIOMS
                                          >
                                         NIL NIL IO))
                    (DatedSource (getFileName <'DIRECTORY directory 'NAME name 'EXTENSION
                                               (PACK* AFFIRMVERSION 'AXIOMS) >
                                              NIL NIL IO))
                    (Unexpected 'TypeFileType)))
          (Files (getFileName <'DIRECTORY directory 'NAME name 'EXTENSION NIL> NIL NIL IO))
          (Unexpected 'AFFIRMObject])
```

26

(getInfixOps

```
  [LAMBDA (typeName)                                    (* D.Thompson "18-Dec-79 16:55")

        (* * This routine returns the infix operator names of the type provided as the parameter. The names are short.)

    (for i in (MKLIST typeName:Infix) collect (Shorten i])
```

27

(getInterfaces

```
  [LAMBDA (typeName)                                    (* D.Thompson "25-Nov-79 00:22")

        (* * This routine returns the interface declarations for the type provided as the first parameter.)

    (PROG (rules typesNotVars)
          (typesNotVars←((UserProfile 'TypesInInterfaces)='Types))
          [if [rules←(for f in (Functions typeName T) join (Rules f '(interface]
              then (for r in rules do (r:LHS←(REMOVE NIL
                                                     <r:LHS:1
                                                       !(for p in r:LHS::1
                                                           collect (if typesNotVars
                                                                       then p:RHS
                                                                      else p:LHS))
                                                     >))
                                             (r:RHS←(Shorten r:RHS:Type]
          (RETURN rules])
```

28

(getOneRuleMatch

```
  [LAMBDA (typeSet parts pattern)                       (* D.Thompson "24-Nov-79 23:59")

        (* * This routine obtains the rules in a selected set of types that -
        (1) are contained in one of the requested parts, and -
        (2) match the pattern.)

    (for type in typeSet collect <type (for rule in (for function in (Functions type)
                                                          join (Rules function parts))
                                            collect rule when (if pattern:1
                                                                  then pattern:1=(Shorten
                                                                                   rule:LHS:Operator)
                                                                 else T)
                                                              and (if pattern:2
                                                                      then pattern:2 MEMB
                                                                           (ShortenAllAtoms
                                                                             (Operators rule:LHS))
                                                                     else T))
                                    >])
```

29

```
(getRewriteRules
  [LAMBDA (typeName parts)                          (* D.Thompson "25-Nov-79 20:11")

        (* * This routine returns the rewrite rules (axioms, lemmas, definitions, and schemas) for a type.)


    (PROG (rules)
          (parts←(MKLIST parts))
          [if (rules←(for f in (Functions typeName) join (Rules f parts)))
              then (for r in rules do (r:LHS←(REMOVE NIL r:LHS]
          (RETURN rules])
```

30

```
(getRuleMatches
  [LAMBDA (typeSet parts pattern)                   (* D.Thompson "25-Nov-79 00:00")

        (* * This routine oversees the generation of the rules matching user-specified patterns.)


    (SELECTQ pattern:1
            (AND (for match1 in (getRuleMatches typeSet parts pattern:2) as match2
                    in (getRuleMatches typeSet parts pattern:3) collect
                                                          <match1:1 (INTERSECTION match1:2
                                                                                  match2:2)
                                                             >))
            (DIFFERENCE (for match1 in (getRuleMatches typeSet parts pattern:2) as match2
                           in (getRuleMatches typeSet parts pattern:3)
                           collect <match1:1 (LDIFFERENCE match1:2 match2:2)
                                   >))
            (OR (for match1 in (getRuleMatches typeSet parts pattern:2) as match2
                   in (getRuleMatches typeSet parts pattern:3)
                   collect
                    <match1:1 (for pairedRule
                                 in (SORT (for rule in < ! match1:2 ! match2:2>
                                             collect <rule:LHS:Operator rule>)
                                          T)
                                 collect pairedRule:2)
                      >))
            (PROGN (getOneRuleMatch typeSet parts pattern])
```

31

```
(getViewableElements
  [LAMBDA (object)                                  (* D.Thompson "21-Oct-80 18:07")

        (* * This routine returns the names of one of the fields of the known names structure, removing those that we don't
        want the user to see. -
        NIL means there no names (or no showable ones, at least); -
        T means that this particular category is not namable; and -
        non-NIL list means these are it.)


    (SELECTQ object
            (AFFIRMObjects (SORT (LDIFFERENCE (for e in KnownNames:AFFIRMObjects
                                                 when (NLISTP e) collect e)
                                              HiddenNames:AFFIRMObjects)
                                 'UPPER))
            (Arcs (SORT KnownNames:Arcs 'UPPER))
            (Axioms T)
            (Commands (SORT (LDIFFERENCE (for c in KnownNames:Commands when (NLISTP c)
                                            collect c)
                                         HiddenNames:Commands)
                            'UPPER))
            (Definitions T)
            (Directories (for d in KnownNames:Directories collect (EVAL d)))
            (DiscardOptions (SORT (LDIFFERENCE KnownNames:DiscardOptions HiddenNames:DiscardOptions)
                                  'UPPER))
            (Disconnected T)
            (Files (SORT (LDIFFERENCE KnownNames:Files HiddenNames:Files)
                         'UPPER))
            (FileTypes (LDIFFERENCE KnownNames:FileTypes HiddenNames:FileTypes))
            (Groups (SORT KnownNames:Groups 'UPPER))
            (HelpTopics KnownNames:HelpTopics)
            (History T)
            (Interfaces T)
            (Lemmas T)
            (Lhs T)
            (Nodes (SORT (Names)
```

```
                    'UPPER))
          (PrintObjects (LDIFFERENCE (for p in KnownNames:PrintObjects when (NLISTP p)
                                           collect p)
                                     HiddenNames:PrintObjects))
          (ProfileEntries (SORT (LDIFFERENCE KnownNames:ProfileEntries HiddenNames:ProfileEntries)
                                'UPPER))
          (Schemas T)
          (Types (SORT (LDIFFERENCE KnownNames:Types HiddenNames:Types)
                       'UPPER))
          (TypeParts (LDIFFERENCE (for p in KnownNames:TypeParts when (NLISTP p) collect p)
                                  HiddenNames:TypeParts))
          (Variables T)
          (Unexpected T)
          (Unexpected 'AFFIRMObject])
```

32

```
(initNeeds
  [LAMBDA (typeName)                                    (* D.Thompson "18-Dec-79 17:24")
    (for need in typeName:Needs do (needs need:1 need::1 typeName])
```

33

```
(listDeclarations
  [LAMBDA (typeName)                                    (* D.Thompson " 6-Nov-80 17:25")

        (* given a list of local decls of the form (x\Interface EWT x\Stack Integer), gather together those with the same
        type, and print them as x,y,z:Integer)


    (PROG (lineLength)
          (lineLength←(LINELENGTH))
          (for g in (getDeclarations typeName) first (printout NIL .TAB0 0 T)
             do (printout NIL # (printName 'declare 'SystemPrintedCommand)
                          , # (PrintArgs (SORT g::1 'UPPER)
                                         T NIL Tab)
                          #
                          (if (MINUSP lineLength-(POSITION)
                                      -(NCHARS g:1)
                                      - 3)
                              then (printout NIL .TAB Tab))
                          Colon , # (INFIX\PRINT3 g:1)
                          CommandTerminator T])
```

34

```
(listInfixOps
  [LAMBDA (typeName interfaces)                         (* D.Thompson " 7-Nov-80 09:57")

        (* * This routine lists infix operators of the type provided as its first parameter, restricting the listing to just
        those infix operators present in the list provided as its second parameter.)


    (PROG (seivedInfixOps)
          (if seivedInfixOps←(for op in (INTERSECTION (getInfixOps typeName)
                                                      (for i in interfaces collect (Shorten i:1)))
                                collect (Skim (Convert op)))
              then (printout NIL .TAB0 0 T # (printName 'infix 'SystemPrintedCommand)
                             , # (PrintArgs (SORT seivedInfixOps 'UPPER)
                                            T CommandTerminator Tab)
                             T])
```

35

```
(listInterfaces
  [LAMBDA (typeName grouped)                            (* D.Thompson " 6-Nov-80 20:25")

        (* * This routine lists the interface declarations for the type provided as the first parameter.)


    (PROG ((groups (<NIL>))
           lineLength rules)
          (lineLength←(LINELENGTH))
          (if rules←(getInterfaces typeName)
              then (if grouped
                       then (for r in rules do (PUTASSOC r:RHS < !(FASSOC r:RHS groups)::1 r:LHS>
                                                       groups))
```

```
                    (for g in groups::1 do (printout NIL .TAB0 0 T #
                                                (printName (Plural 'interface g::1)
                                                           'SystemPrintedCommand))
                                           (if (EQLENGTH g::1 1)
                                               then (printout NIL ,)
                                             else (printout NIL .TAB Tab))
                                           [for i on g::1 by i::1
                                              do (printout NIL # (INFIX\PRINT3 i:1))
                                                 (if i::1
                                                     then (printout NIL Comma ,)
                                                          (if (FitsOnLine i:2 lineLength-(
                                                                                POSITION))
                                                              else (printout NIL .TAB Tab]
                                           (if (MINUSP lineLength-(POSITION)
                                                       -(NCHARS g:1)
                                                       - 3)
                                               then (printout NIL .TAB Tab))
                                           (printout NIL Colon , g:1 CommandTerminator T)
                                           (listInfixOps typeName g::1))
                  else (printout NIL .TAB0 0 T)
                       (for r in rules do (printout NIL # (printName 'interface
                                                                     'SystemPrintedCommand)
                                                          , # (INFIX\PRINT3 r:LHS)
                                                          Colon , r:RHS CommandTerminator T))
                       (listInfixOps typeName (for r in rules collect r:LHS])
```

36

(listNeeds

```
  [LAMBDA (typeName grouped)                                   (* D.Thompson " 6-Nov-80 20:23")

        (* * This routine produces the listing of the entries in the "Needs" lists. This list currently contains files and
        types.)


    (PROG (elements object)
          (for need in typeName:Needs do (object←need:1)
                                         (elements←need::1)
                                         (if grouped
                                             then (printout NIL .TAB0 0 T #
                                                            (printName <'needs (Plural (L-CASE object)
                                                                                       elements)
                                                                       >
                                                                       'SystemPrintedCommand)
                                                            , # (PrintArgs elements T
                                                                           CommandTerminator Tab)
                                                            T)
                                           else (for e in elements first (printout NIL .TAB0 0 T)
                                                   do (printout NIL .TAB0 0 #
                                                                (printName
                                                                  <'needs (L-CASE (Plural object 1))
                                                                    >
                                                                  'SystemPrintedCommand)
                                                                , e CommandTerminator T])
```

37

(listOneRule

```
  [LAMBDA (lhs separator rhs terminator)                       (* D.Thompson " 6-Nov-80 11:57")
    (PROG (lhsPosition parens?)
          (lhsPosition←(POSITION))
          (lhs←(RemoveIfs lhs))
          (rhs←(RemoveIfs rhs))
          (parens?←(parenthesize rhs))
          (INFIX\PRINT3 lhs)
          (if (FitsOnLine rhs (LINELENGTH)
                          -(POSITION))
              then (printout NIL ,)
            else (printout NIL .TAB (lhsPosition+3)))
          (printout NIL (if separator
                            else NullString)

                        (if parens?
                            then LeftParenthesis
                          else NullString)
                        #
                        (INFIX\PRINT3 rhs)
                        (if parens?
                            then RightParenthesis
```

```
                    else NullString)
                  (if terminator
                    else NullString)
                  T])
```

38

```
(listRewriteRules
  [LAMBDA (typeName part grouped)                    (* R.Erickson "31-Aug-81 13:10")

          (* * This routine lists the rewrite rules (axioms, lemmas, definitions, and schemas) for a type.)


          (* * Damned if I understand this! -RWE)


    (PROG (commandName mainOp previousMainOp rules)
          (if rules←(getRewriteRules typeName part)
              then                                   (* only act if some are found)
                                                     (* some parts have idiosyncratic command names;
                                                     translate)
               commandName←(SELECTC part
                                    (RuleTypes:Define (QUOTE define))
                                    (RuleTypes:Lemma (QUOTE rulelemma))
                                    part)
               (SELECTC
                 part
                 [(LIST RuleTypes:Axiom RuleTypes:Lemma RuleTypes:Automatic)
                   (COND
                     [grouped
                       (for r on rules by (CDR r) first (SETQ previousMainOp NIL)
                          do [SETQ mainOp (fetch Operator of (fetch LHS of (CAR r]
                             (COND
                               [(NEQ previousMainOp mainOp)
                                 (SETQ previousMainOp mainOp)
                                 (COND
                                   ((AND (CDR r)
                                         (EQ (fetch Operator of (fetch LHS of (CADR r)))
                                             mainOp))
                                     (printout NIL .TAB0 0 T # (printName (Plural commandName 2)
                                                                          (QUOTE SystemPrintedCommand)
                                                                          )
                                                             T .TAB Tab # (listOneRule (fetch LHS of (CAR r))
                                                                                       RewriteRuleOp
                                                                                       (fetch RHS of (CAR r))
                                                                                       Comma)))
                                   (T (printout NIL .TAB0 0 T # (printName (Plural commandName 1)
                                                                           (QUOTE
                                                                             SystemPrintedCommand))
                                                              , # (listOneRule (fetch LHS of (CAR r))
                                                                               RewriteRuleOp
                                                                               (fetch RHS of (CAR r))
                                                                               CommandTerminator]
                                  (T (printout NIL .TAB Tab #
                                               (listOneRule (fetch LHS of (CAR r))
                                                            RewriteRuleOp
                                                            (fetch RHS of (CAR r))
                                                            (COND
                                                              ((AND (CDR r)
                                                                    (EQ (fetch Operator
                                                                               of (fetch LHS
                                                                                     of (CADR r)))
                                                                        mainOp))
                                                                Comma)
                                                              (T CommandTerminator]
                             (T (for r in rules first (SETQ previousMainOp NIL)
                                   do (SETQ mainOp (fetch Operator of (fetch LHS of r)))
                                      (COND
                                        ((NEQ previousMainOp mainOp)
                                          (printout NIL .TAB0 0 T)))
                                      (SETQ previousMainOp mainOp)
                                      (printout NIL .TAB0 0 (printName commandName
                                                                       (QUOTE SystemPrintedCommand))
                                                          , # (listOneRule (fetch LHS of r)
                                                                           RewriteRuleOp
                                                                           (fetch RHS of r)
                                                                           CommandTerminator]
                      [(LIST RuleTypes:Define RuleTypes:Schema)
                        (COND
```

```
                [grouped (printout NIL .TAB0 0 T # (printName commandName
                                                          (QUOTE SystemPrintedCommand)))
                         (COND
                           ((EQLENGTH rules 1)
                             (printout NIL . # (listOneRule (fetch LHS of (CAR rules))
                                                            RewriteRuleOp
                                                            (fetch RHS of (CAR rules))
                                                            CommandTerminator)))
                           (T (printout NIL T)
                              (for r on rules by (CDR r) first [SETQ previousMainOp
                                                                  (fetch Operator
                                                                     of (fetch LHS
                                                                           of (CAR rules]
                                 do [SETQ mainOp (fetch Operator of (fetch LHS of (CAR r]
                                    (COND
                                      ((NEQ previousMainOp mainOp)
                                        (SETQ previousMainOp mainOp)
                                        (printout NIL .TAB0 0 T)))
                                    (printout NIL .TAB Tab # (listOneRule (fetch LHS
                                                                             of (CAR r))
                                                                          RewriteRuleOp
                                                                          (fetch RHS
                                                                             of (CAR r))
                                                                          (COND
                                                                            ((CDR r)
                                                                              Comma)
                                                                            (T CommandTerminator]
                (T (for r in rules first (SETQ previousMainOp NIL)
                      do (SETQ mainOp (fetch Operator of (fetch LHS of r)))
                         (COND
                           ((NEQ previousMainOp mainOp)
                             (printout NIL .TAB0 0 T)))
                         (SETQ previousMainOp mainOp)
                         (printout NIL .TAB0 0 # (printName commandName
                                                            (QUOTE SystemPrintedCommand))
                                   . # (listOneRule (fetch LHS of r)
                                                    RewriteRuleOp
                                                    (fetch RHS of r)
                                                    CommandTerminator]
                   (Unexpected (QUOTE TypePart])
```

39

(listRuleMatches

```
  [LAMBDA (pattern matches printHeader interfaces)          (* D.Thompson "19-Jun-80 16:08")

        (* * This routine oversees the listing of the rules matching user-specified patterns.)


    (PROG (foundOne)
          (if printHeader
              then (printout NIL .TAB0 0 T # (INFIX\PRINT3 (makeExpFromPattern pattern))
                           Colon T))

          (* * want to list appropriate interfaces here)


          [for match in matches do (if match:2~=NIL
                                       then foundOne←T
                                            (printout NIL .TAB0 0 T "Type" . #
                                                      (PrettyPrint match:1 T)
                                                      Colon)
                                            (for rule in match:2
                                               do (printout NIL .TAB0 0 T .TAB Tab #
                                                            (listOneRule rule:LHS RewriteRuleOp
                                                                         rule:RHS]
          (if foundOne
            else (printout NIL .TAB0 0 "(No rules matched.)" T])
```

40

(listTypeAutos

```
  [LAMBDA (part type)                                       (* R.Erickson "22-Sep-81 22:09")
                                                            (* part is one of (distinct nochange) args have been
                                                            validated.)

        (* * Called internally to print this part of a type spec.)
```

```
    (printout NIL .TAB0 0 T)
    (SELECTQ part
            [distinct (PROG (prop pres post)

(* We either print one command per group of fns or, if all constructors are targets, give no args but comment it
with those actually used. pres is a list of cmd args, post the comment to follow)

                            (prop←type:Distinct)
                            (if prop:1=T
                                then pres← <NIL> post←prop:2
                              else pres←prop)
                            (for args in pres do          (* copied from listDeclarations)
                                                  (printName 'distinct 'SystemPrintedCommand)
                                                  (if args
                                                      then (SPACES 1)
                                                           (PrintArgs (SORT (Shorten args)
                                                                            'UPPER)
                                                                      T NIL Tab))
                                                  (if post
                                                      then (printout NIL , LeftCurlyBracket #
                                                                   (PrintArgs (SORT (Shorten post)
                                                                                    'UPPER)
                                                                              T NIL Tab)
                                                                   RightCurlyBracket))
                                                  (printout NIL CommandTerminator T]
        [nochange

(* * Print commands, "nochange <selectors>", possibly with "by <type>" if not the same type)

                    (PROG (prop pSels pType)
                          (prop←type:NoChange)
                          (for entry in prop do (pSels←entry:3)
                                                (* selectors)
                                                (pType←entry:1)
                                                (* type)
                                                (* :2 = T if all selectors)
                                                (printName 'nochange 'SystemPrintedCommand)
                                                (SPACES 1)
                                                (if entry:2
                                                    then (printout NIL LeftCurlyBracket))
                                                (PrintArgs (SORT (Shorten pSels)
                                                                 'UPPER)
                                                           T NIL Tab)
                                                (if entry:2
                                                    then (printout NIL RightCurlyBracket))
                                                (if pType~=type
                                                    then (SPACES 1)
                                                         (printName 'by 'SystemPrintedCommand)
                                                         (printout NIL , pType))
                                                (printout NIL CommandTerminator T]
        (SHOULDNT])
```

41

```
(listTypeParts
  [LAMBDA (type parts)                                (* R.Erickson " 3-Sep-81 18:46")

(* * This routine oversees the listing of a sequence of parts of a type.)

    (PROG (typeName)
          (if typeName←(CheckForType type NIL T)
              then parts←(if parts
                             then                     (* user specified: spelling-correct)
                                  (for p in parts bind correctPartName
                                     eachtime (correctPartName←(AFFIRMSpellingCorrect p
                                                                    KnownNames:TypeParts NIL
                                                                          NIL NIL))
                                                      (* asks user if necessary)

                                     when correctPartName collect correctPartName)
                           else                       (* default: all but auto)
                                (REMOVE RuleTypes:Automatic (getViewableElements 'TypeParts)))
                  (for p in parts do                  (* SELECTC is like SELECTQ, but evaluates keys
                                                         (at compile time))
                                     (SELECTC p
                                              ((QUOTE needs)
                                                (listNeeds typeName (UserProfile (QUOTE
```

```
                                                    NeedsGrouping)
                                                  T)))
                ((QUOTE declare)
                  (listDeclarations typeName))
                ((QUOTE interface)
                  (listInterfaces typeName
                                  (UserProfile (QUOTE
                                                 InterfaceGrouping)
                                               T)))
                ((QUOTE (distinct nochange))
                  (listTypeAutos p typeName))
                ((LIST RuleTypes:Axiom RuleTypes:Lemma
                       RuleTypes:Schema)
                  (listRewriteRules typeName p
                                    (UserProfile (PACK* p
                                                        (QUOTE
                                                          Grouping))
                                                 T)))
                (RuleTypes:Define (listRewriteRules
                                    typeName p
                                    (UserProfile (QUOTE DefineGrouping)
                                                 T)))
                (RuleTypes:Automatic (listRewriteRules typeName p T))
                NIL])
```

42

(loadNeededFiles

```
[LAMBDA (fileNames)                                    (* D.Thompson "5-Sep-80 20.15")

(* * This routine loads any files contained in the list of file names provided as the first parameter that are not
already loaded.)


  (PROG (file)
        (fileNames←(for f in fileNames collect (makeFileName f)))
        (for f in fileNames do (if (Loaded? f 'Files)
                                   then (needReport f 'Files 'AlreadyLoaded)
                                 elseif file←(getFileName f)
                                   then (AffirmLoad file)
                                 else (needReport f 'Files 'NotFound)))
        (RETURN (for f in fileNames collect (UNPACKFILENAME f])
```

43

(loadNeededTypes

```
[LAMBDA (typeNames)                                    (* D.Thompson "22-Feb-80 15:37")

(* * This routine loads (or ensures the previous existence of) a list of types provided as the parameter.)


  (PROG (askUser directory file files pendingType typeName)
        (typeNames←(for type in typeNames eachtime typeName←(if (LISTP type)
                                                                then (PACK type)
                                                              else type)
                        collect (if (CheckForType typeName NIL 'ok)
                                  else typeName)))
        [if askUser←(UserProfile 'LoadNeededTypes) MEMB OffValues
            then (needReport typeNames 'Types 'NotLoaded)
          else askUser←(askUser='Ask)
               (for type in typeNames
                  do (if typeName←(Loaded? type 'Types)
                         then (needReport typeName 'Types 'AlreadyLoaded)
                       else files←(for d in KnownNames:Directories eachtime directory←(EVAL d)
                                     join (for fileType in KnownNames:FileTypes
                                             eachtime file←(getFileNameFromDirectory type directory
                                                                                     'Types
                                                                                     fileType)
                                             when file collect <(GETFILEINFO file 'IWRITEDATE)
                                                               (GETFILEINFO file 'WRITEDATE)
                                                               file fileType>))
                                  (if files←(REVERSE (SORT files T))
                                      then file←(if askUser
                                                    then (askAboutFiles files type 'Types)
                                                  else files:1)
                                           (if file
                                               then (SELECTQ file:4
                                                             ((Compiled Saved OldCompiled OldSaved)
                                                               (AffirmLoad file:3))
```

```
                                        ((Source DatedSource)
                                          pendingType←CurrentType
                                          (ReadCommandFile file:3)
                                          (if (Loaded? type 'Types)
                                            else (needReport type 'Types 'NotLoaded))
                                           (* ensu. the just-defined type is closed off properly)
                                          (while CurrentType~=pendingType
                                                  and (PopTypeStack) do T)
                                          (if CurrentType~=pendingType
                                              then (Edit pendingType)))
                                        (Unexpected 'TypeFileType T))
                                  else (needReport type 'Types 'NotLoaded))
                            else (needReport type 'Types 'NotFound)
                                 (MinimalTypeSpec type]
        (RETURN typeNames])
```

44

```
(makeExpFromPattern
  [LAMBDA (pattern)                                    (* D.Thompson "24-Nov-79 23:52")

        (* * This routine reconstructs a printable expression from a search pattern.)


    (SELECTQ pattern:1
            ((AND DIFFERENCE OR)
              <pattern:1
              (makeExpFromPattern pattern:2)
              (makeExpFromPattern pattern:3)
              >)
            (NIL <pattern:2>)
            (PROGN (if pattern:2=NIL
                       then pattern:1
                   else pattern])
```

45

```
(makePatterns
  [LAMBDA (expressionSequence)                         (* D.Thompson "24-Nov-79 23:42")

        (* * This routine constructs a sequence of search patterns from the sequence of expression provided as the
        parameter.)


    (for expression in expressionSequence collect (if (LISTP expression)
                                                      then
                                                           (* the pattern will have at least an internal op)
                                                          (SELECTQ expression:1
                                                                  ((AND DIFFERENCE OR)
                                                           (* A pattern operation)
                                                                    <expression:1
                                                                    (makePatterns expression:2)
                                                                    (makePatterns expression:3)
                                                                    >)
                                                                  (PROGN
                                                           (* normal simple pattern)
                                                                         (if (EQLENGTH expression 1)
                                                                             then
                                                           (* only internal op)
                                                                                 <NIL
                                                                                 expression:1>
                                                                           else
                                                           (* both main op and internal op.
                                                           Note that any other components of the expression are
                                                           implicitly eaten.)
                                                                                <expression:1
                                                                                  expression:2>)))
                                                      else     (* only main op)
                                                           <expression NIL>])
```

46

```
(needReport
  [LAMBDA (elements objectCategory disposition)        (* D.Thompson " 8-Jan-80 10:14")
    (if disposition~='AlreadyLoaded
        then (printout NIL .TAB0 0 "needs:" 10 disposition Colon , .PPVTL (MKLIST elements)
                   T])
```

47

```
(needs
  [LAMBDA (object elements typeName)                    (* D.Thompson " 3-Jan-80 12:02")

          (* * This routine ensures that the needed elements in the list of elements provided as the second parameter, each an
          AFFIRM object of the kind indicated by the first parameter, are loaded before further processing of the current type
          specification. Loops are avoided. The current objects this routine handles are files and types.)


    (if object←(AFFIRMSpellingCorrect object KnownNames:AFFIRMObjects)
        then elements←(MKLIST elements)
             elements←(SELECTQ object
                               (Types (loadNeededTypes elements))
                               (Files (loadNeededFiles elements))
                               (PROGN (printout NIL .TAB0 0 "Sorry, the" , object ,
                                              "part of the %"needs%" command is not yet functional."
                                                   T "(If you really need this, gripe about it!)" T)
                                      NIL))
             (AddNeeds object elements typeName)
      else (AffirmError])
```

48

```
(parenthesize
  [LAMBDA (exp)                                         (* R.Bates "12-Nov-79 16:28")
    (PROG (mainOp)
          (if (NLISTP exp)
              then (RETURN NIL))
          (mainOp←(Shorten exp:1))
          (RETURN (if mainOp MEMB <ANDIO EQIO IMPIO ORIO>
                      then T
                    elseif mainOp=QOP and exp:given=NIL and exp:find=NIL
                      then (parenthesize exp:expr)
                    else NIL])
```

49

```
(parse
  [LAMBDA (rootNonterminal fileOrAtomList terminator)   (* D.Thompson "26-Feb-80 11:31")

          (* * This routine acts as AFFIRM's main interface to the parser. It is called with 3 parameters, the nonterminal
          we're looking to match (most commonly expression or expression Seq), the file or atom list to read from / eat, and
          the termination atom (commonly the value of the global variable CommandTerminator).)


    PARSEDTOKENS← <NIL> (if (reduceExpression (PARSER rootNonterminal <fileOrAtomList terminator> T))
                            else (if fileOrAtomList=T
                                     then (CLEARBUF T))
                                 (ParsingError NIL)
                                 NIL])
```

50

```
(printHelper
  [LAMBDA (type target option)                          (* D.Thompson "14-Oct-80 18:01")

          (* * This routine is called by print. The real work is done here.)


    (SELECTQ type
             [assumptions                                (* list all assumptions and where used)
                          (PROG (pairs)
                                (pairs←(for t in Theorems when t:status='assumed
                                          collect <(TheoremId t)
                                                  (UsedBy t)
                                                  >))
                                (for pair in (SORT pairs 'UPPER)
                                   do (if pair:2
                                          then (AFFIRMMAPRINT (CONCAT pair:1 " is used by")
                                                              (for t in pair:2
                                                                 collect (TheoremId t))
                                                              T T)
                                        else (printout NIL .TAB0 0 T pair:1 , "is unused."))
                                      (printout NIL .TAB0 0 "assume" , pair:1 Comma , #
                                                (PrettyPrint (NodeToExpr pair:1)
                                                             T)
```

```
                                        CommandTerminator T]
(BadEquations (if BadEquations
                  then (for badRule in BadEquations as i from 1
                          do (printout NIL .TAB0 0 T .I3 i 10 #
                                       (listOneRule badRule:LHS RewriteRuleOp
                                                    badRule:RHS)))
                  else (printout NIL .TAB0 0
                                 "There are no rules in the BadEquations list."
                                 T)))
[(both named proof)
  (for nodes on (printWhat? target T) do (SELECTQ type
                                                (proof (PrintProof nodes:1 option))
                                                (both (PrintBoth nodes:1 option))
                                                (named (PrintProof nodes:1 'named))
                                                (SHOULDNT))
                                         (if nodes::1
                                             then (printout NIL .SKIP 4]
[IH (PROG (IHs)
          (if IHs←(FindSubexpressions IH2OP (if (type? Qexpression CurrentPropn)
                                                then CurrentPropn:expr
                                                else CurrentPropn))
              then [for IH in (INTERSECTION IHs IHs)
                      do (printout NIL .TAB0 0 T .TAB Tab #
                                   (listOneRule IH RewriteRuleOp
                                                (ExpandVisibleDefs <IH2OP> IH]
              else (printout NIL .TAB0 0 "IH isn't locally defined." T]
(implist (printout NIL .TAB0 0 # (MAPRINT IMPLIST)
                   T))
[known (PROG (list object)
             (for x in target
                do (if object←(AFFIRMSpellingCorrect x KnownNames:AFFIRMObjects)
                       then (if list←(getViewableElements object)
                                then
                                 (if list=T
                                     then (printout NIL .TAB0 0 "There are no namable" ,
                                                    (AppendChar (L-CASE object)
                                                                's)
                                                    ExclamationPoint T)
                                   else (AFFIRMMAPRINT
                                          (Plural (CONCAT "The known "
                                                          (RemoveChar (L-CASE object)
                                                                      's)
                                                          " names are")
                                                  list)
                                          list T T))
                              else (printout NIL .TAB0 0 "There aren't currently any" ,
                                             (RemoveChar (L-CASE object)
                                                         's)
                                             , "names." T))
                     else (printout NIL .TAB0 0 DoubleQuote (L-CASE x T)
                                    DoubleQuote , "isn't an AFFIRM object." T]
(names (printHelper 'known '(Nodes)))
[next (PROG (nx)
            (nx←(Next))
            (SELECTQ nx:1
                     (STAY (printout NIL .TAB0 0 "You're at the only leaf."))
                     (LEAF (printout NIL .TAB0 0 "The next leaf is")
                           (printPropn nx:2:1))
                     (DOWN (AFFIRMMAPRINT (Plural "The next lemmas are" nx::1)
                                          (for n in nx::1 collect n:-1)
                                          T T))
                     (UP (AFFIRMMAPRINT (Plural "The unproven ancestors are" nx::1)
                                        nx::1 T T))
                     (NIL (printout NIL .TAB0 0 "The proof of this part is done." T)
                          (printHelper 'status '(unproven)))
                     (SHOULDNT]
(original (PrettyPrint (GetExpression CurrentNode:prop#)))
[parts (PROG (interfaces lhs parts pieces printPattern typeSet)
             (pieces←(decipherRuleMatchInput target))
             (interfaces←pieces:4)
             (lhs←pieces:3)
             (parts←(if pieces:2
                        else <'axiom >))
             (typeSet←(if pieces:1
                          else KnownNames:Types))
             (printPattern←(EQLENGTH lhs 1))    (* we don't handle interfaces yet.)
             (for pattern in lhs do (listRuleMatches pattern
                                                     (getRuleMatches typeSet parts
                                                                     pattern)
                                                     printPattern interfaces]
(prop (for it in (printWhat? target '*) do (printPropn it)))
```

```
        (result (PrettyPrint CurrentPropn NIL T))
        [status (PROG (categories)
                      (if target=NIL or (EQLENGTH target 1) and target:1 MEMB '(theorems
                                                                                unproved
                                                                                unproven)
                          then                          (* user asked for general info: group according to
                                                        status.)
                               categories←(for i in (if target and target:1 MEMB '(unproven
                                                                                   unproved)
                                                         then (LDIFFERENCE StatusCategories '(
                                                                                assumed proved))
                                                         else StatusCategories)
                                                   collect <i>)
                               (for node in (printWhat? target 'theorems)
                                  bind (thm xname status)
                                  do (if thm←(NodeToThm node)
                                         then status←(Status thm)
                                              xname←(NodeId node)
                                              (PUTASSOC status <xname
                                                                !(FASSOC status categories)::1>
                                                        categories)))
                               (for c in categories bind status
                                  do (status←(if c:1='lemmas
                                                 then "awaiting lemma proof"
                                                 else c:1))
                                     (if c::1
                                         then (AFFIRMMAPRINT (CONCAT "The " status
                                                                     (Plural " theorems are"
                                                                             c::1))
                                                             (SORT c::1 'UPPER)
                                                             T T)
                                       else (printout NIL .TAB0 0 "No theorems are" , status
                                                      Period T)))
                          else                          (* user asked for specific theorem names: simply list
                                                        each.)
                               (for node in (printWhat? target 'theorems) bind thm
                                  do (printout NIL .TAB0 0 (NodeId node)
                                               , "is" , (if thm←(NodeToThm node)
                                                            then (PrettyStatus thm)
                                                            else "not a theorem")
                                               Period T]
        (type target← (CheckForType target)
              (printout NIL .TAB0 0 T # (printName 'type 'SystemPrintedCommand)
                        , # (printName target 'TypeName)
                        CommandTerminator T # (listTypeParts target option)
                        T # (printName 'end 'SystemPrintedCommand)
                        , LeftCurlyBracket # (printName target 'TypeName)
                        RightCurlyBracket , CommandTerminator T))
        (uses (if Annotating
                  then (Dprintout NIL .TAB0 0 "@Begin[Format]" T "@TabClear" T "@Tabs(25, 45)" T
                              "Theorem@\Uses:@\Used by:")
                       (Dprintout T .TAB0 0 "Theorem" .TAB0 25 "Uses:" .TAB0 45 "Used by:")
                  else (printout NIL .TAB0 0 "Theorem" .TAB0 25 "Uses:" .TAB0 45 "Used by:"))
              [for node in (printWhat? target 'theorems) bind (thm uses usedBy)
                 do (thm←(NodeToThm node))
                    (if thm
                        then                            (* else ignore it)
                             uses←(if (for u in (Facts thm) collect (TheoremId u))
                                      else <Hyphen>)
                             usedBy←(if (for b in (UsedBy thm) collect (TheoremId b))
                                        else <Hyphen>)
                             uses←(FillOut uses usedBy NullString)
                             usedBy←(FillOut usedBy uses NullString)
                             (if Annotating
                                 then (Dprintout NIL .TAB0 0 (TheoremId thm)
                                                 Comma , (PrettyStatus thm)
                                                 "@\" uses:1 "@\" usedBy:1)
                                      (Dprintout T .TAB0 0 (TheoremId thm)
                                                 Comma , (PrettyStatus thm)
                                                 .TAB0 25 uses:1 .TAB0 45 usedBy:1)
                                      (for u in uses::1 as b in usedBy::1
                                         do (Dprintout NIL .TAB0 0 "@\" u "@\" b)
                                            (Dprintout T .TAB0 25 u .TAB0 45 b))
                               else (printout NIL .TAB0 0 (TheoremId thm)
                                              Comma , (PrettyStatus thm)
                                              .TAB0 25 uses:1 .TAB0 45 usedBy:1)
                                    (for u in uses::1 as b in usedBy::1
                                       do (printout NIL .TAB0 25 u .TAB0 45 b T]
              (if Annotating
                  then (Dprintout NIL .TAB0 0 "@End[Format]" T)))
        [variables (PROG (temp)
```

```
                        (if (type? Qexpression CurrentPropn)
                            then temp←(COPY CurrentPropn)
                                 temp:expr←Ellipses
                                 (PrettyPrint temp)
                          else (printout NIL .TAB0 0 "There aren't any variables."]
            (SHOULDNT])
)

(RPAQ EqConsFlg 2)

(RPAQ InterfaceProcessing NIL)

(RPAQ InternalOperatorNames <<'= EQIO> <'+ ADDIO> <'- DIFFIO> <'< LTIO> <'> GTIO> <'* MULTIO> <'/
      DIVIO> <'& ANDIO> <'~ NOTIO> <'! NOTIO> <'%| ORIO> <'~= NEIO> <'<= LEIO> <'>= GEIO> <'notEqual
      NEIO> <'lessThanEqual LEIO> <'greaterThanEqual GEIO>>)

(RPAQ PatternDebug NIL)

(RPAQ PropStorage (create PropStorage))

(RPAQ RewriteRuleOp "==")

(RPAQQ StatusCategories (untried tried assumed lemmas proved))
[DECLARE: EVAL@COMPILE

(RECORD TerminalEnvironment (LineBuffering EchoMode EchoControls DeleteControls DeleteEchoMode))

(RECORD AFFIRMEnvironment (BreakAccess LineLength GCMessage GCPages HistoryLength))
]

(ADDTOVAR GLOBALVARS RewriteRuleOp)
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

```
(* error messages!) ]

(DEFINEQ

                                                                    51

(CannedMessage
  [LAMBDA (atomicKey generateError auxList)          (* R.Erickson "24-Sep-81 16:58")

        (* * Utility which is used to print an error or informative message. This might take different forms depending on
        whether the user has seen it before in this session, since we use the global CannedMessagesSeen.)


        (* * atomicKey designates the message -
        generateError asks to end with ERROR! -
        auxList is extra information to print, depends on the message.)


    (PROG (first)
          (if atomicKey MEMB CannedMessagesSeen
            else first←T
                 (push CannedMessagesSeen atomicKey))

          (* * SELECTQ keys are in alphabetic order.)


          (SELECTQ atomicKey
                   (discard                          (* user discards part of a specification;
                                                     war of possible inconsistencies.
                                                     auxList:1 is one of (variable))
                            (printout NIL T (if first
                                                then

"If the variable(s) being discarded are referenced  in any interfaces,
rules, or theorems, then an inconsistency may be introduced.  This will
not result in invalid proofs, but may be misleading and make entry
of certain expressions impossible."
                                                else
                                              "Remember to check for any other uses of the variable(s).")
                                            T))
                   (format                           (* someone called an internal fn with bad argument
                                                     format)
                           (printout NIL T (if first
                                               then
                                            "Please report (to Affirm@isi) that an illegal format was passed to:"
                                               else "Illegal internal format:")
                                   , auxList:1 T))
                   (incompatible                     (* auxList:1 is a command which has been called in
                                                     several incompatible ways. auxList:2 is context.)
                                 (printout NIL T "The" , auxList:1 ,
                                     "command has been called in ways which are incompatible for"
                                          , auxList:2 T))
                   (illegal                          (* Catchall: the user entered a name or something in
                                                     illegal format. auxList:1 is the category. and
                                                     auxList::1 additional info about what was wrong.)
                            (printout NIL T "Illegal format for" , auxList:1 Colon)
                            (for x in auxList::1 do (printout NIL , x))
                            (TERPRI))
                   (invokeNoFn                       (* invoke needs to have the fn specified)
                               (printout NIL T (if first
                                                   then
                                "The invoke command must be given an operator for each argument, in particular:"
                                                   else "Invoke needs a function for:")
                                           ,)
                               (PrettyPrint auxList:1 T))
                   (key                              (* somebody passed an unknown key in an internal call)
                        (printout NIL T (if first
                                            then
                                         "Please report (to Affirm@isi) that an unknown key was used internally."
                                            else "unknown internal key:")
                                  , auxList T))
                   (varandFun                        (* attempt to use same name as both variable and
                                                     function)
                              (printout NIL T
                                        "Cannot be both a variable and interface in a single type:"
                                        ,
                                        (Shorten auxList:1)))
                   [varConflict (PROG ((kind (auxList:1))
                                       (conflicts (auxList:2))
```

```
                                (type (auxList:3)))

        (* * attempt to declare variable which conflicts with existing; we refuse. auxList:1 indicates variety, is one of: -
        prime :user declaring x' to conflict with x -
        root :user gave conflicting decls for x -
        vc :VCGen shows conflicts in program -
        internal :shouldn't happen)

                                                        (* auxList:2 has entries <badVar existingType>;
                                                        entries for prime have third element with name of root
                                                        var. auxList:3 is attempted type.)
                                        (printout NIL T (SELECTQ kind
                                                            (prime
                        "Cannot declare a primed variable to conflict with the unprimed root var.")
                                                            (root "Declaration conflict.")
                                                            (vc (if first
                                                                    then

"Typechecking of program gives variable type conflict.  Probable cause:
conflicting use in program or existing var in current type."
                                                                        else
                                                             "Program variable type conflict."))
                                                            (internal (AffirmError
                                                                        <
"Illegal automatic declaration" auxList:2> 'internal))
                                                            (CannedMessage 'key
                                                                        <'varConflict type>))))
                                        (if kind='vc or commandFile~=Terminal
                                            then                (* user doesn't know type)
                                                 (printout NIL , "Cannot declare of type" , type))
                                        (for c in conflicts
                                           do (printout NIL T (Shorten c:1)
                                                              ,)
                                              (SELECTQ kind
                                                       (prime (printout NIL "conflicts with" ,
                                                                        (Shorten c:3)
                                                                        , "which is of type"))
                                                       ((vc root)
                                                         (printout NIL "is already of type"))
                                                       (SHOULDNT))
                                              (printout NIL , c:2]
                                (CannedMessage 'key T <atomicKey 'CannedMessage >))
                        (if generateError
                            then (ERROR!])
)
[DECLARE: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY
(BLOCK: EqConsBLOCK EqCons EqCons1 (ENTRIES EqCons)
        (GLOBALVARS EqConsList temp)
        (NOLINKFNS . T))
]
(DECLARE: DONTCOPY
  (FILEMAP (NIL (1863 57977 (AddNeeds 1875 . 2910) (CompactPropStorage 2914 . 3586) (DefnRangeConvert
3590 . 4649) (EqCons 4653 . 4819) (EqCons1 4823 . 5354) (FindIHs 5358 . 5754) (FindSubexpressions 5758
 . 6359) (FitsOnLine 6363 . 6999) (GetExpression 7003 . 7304) (GetProp# 7308 . 7754) (Loaded? 7758 .
8731) (MapToInternal 8735 . 9216) (PrettyPrint 9220 . 9501) (PrettyPrintCommand 9505 . 10273) (
PrettyPrintHistory 10277 . 11128) (PrintArgs 11132 . 11910) (PrintCurrentProposition 11914 . 12271) (
PrintList 12275 . 12412) (ReadExpression 12416 . 12845) (ReadExpressionList 12849 . 13388) (
TranslateTo# 13392 . 13980) (UseNumbers 13984 . 14165) (decipherRuleMatchInput 14169 . 19119) (
getDeclarations 19123 . 19782) (getFileNameFromDirectory 19786 . 21087) (getInfixOps 21091 . 21410) (
getInterfaces 21414 . 22169) (getOneRuleMatch 22173 . 22969) (getRewriteRules 22973 . 23458) (
getRuleMatches 23462 . 24646) (getViewableElements 24650 . 26752) (initNeeds 26756 . 26953) (
listDeclarations 26957 . 27762) (listInfixOps 27766 . 28527) (listInterfaces 28531 . 30232) (listNeeds
 30236 . 31208) (listOneRule 31212 . 32050) (listRewriteRules 32054 . 36482) (listRuleMatches 36486 .
37425) (listTypeAutos 37429 . 39862) (listTypeParts 39866 . 41836) (loadNeededFiles 41840 . 42567) (
loadNeededTypes 42571 . 44913) (makeExpFromPattern 44917 . 45421) (makePatterns 45425 . 46893) (
needReport 46897 . 47159) (needs 47163 . 48170) (parenthesize 48174 . 48632) (parse 48636 . 49321) (
printHelper 49325 . 57974)) (58826 64209 (CannedMessage 58838 . 64206)))))
STOP
```