<AFFIRM>SPECIFICATION..30 30-Sep-81 15:40:31

```
(FILECREATED "26-Sep-81 19:30:02" <AFFIRM>SPECIFICATION..30 61769

     changes to:  SPECIFICATIONCOMS

     previous date: "26-Sep-81 18:06:11" <AFFIRM>SPECIFICATION..29)


(PRETTYCOMPRINT SPECIFICATIONCOMS)

(RPAQQ SPECIFICATIONCOMS [(FNS * SpecCmds)
                          (* Data types)
                          (FNS * TypeQueries)
                          (FNS * TypeMods)
                          (* Typechecking and interfaces)
                          (FNS * InterfaceQueries)
                          (FNS * InterfaceMods)
                          (FNS * InterfaceInternals)
                          (* other utilities)
                          (FNS * SpecFiles)
                          (FNS * SpecOther)
                          (FNS * SpecLimbo)
                          (VARS SavedFnProps)
                          [VARS (KnownTypes (QUOTE (Basis TypeParameter Boolean BUILTIN Integer
                                                    Induction]
                          (ADDVARS (NOSAVESETVARS KnownTypes))
                          (PROP INFO InitializeLoad)
                          (DECLARE: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS
                                    (ADDVARS (NLAMA)
                                             (NLAML Using Quote InitializeLoad)
                                             (LAMA cases\Schema cases\Interface GenericInterface])

(RPAQQ SpecCmds (Axiom Axioms Define Distinct Edit NoChange Schema adopt declareVar discard
                       discardInterface discardVariable distinct nochange set))
(DEFINEQ

                                                                                          1

(Axiom
  [LAMBDA (rule kind)                                          (* D.Thompson "22-Jun-80 21:40")

          (* * This routine processes the AXIOM and LEMMA commands of AFFIRM. Its paremeters are one reduced parsed equation,
          and one of {axiom lemma}.)


    (PROG (ContradictionFound (rulelimit NIL)
                              (rulecount 0))
          (rule←(pexec rule T))
          (if (NoFreeVarsInRule rule kind)
            else (AffirmError))
          (if [NLSETQ (PROGN (AddRule rule kind T)
                             (if Completion
                                 then (Completer NIL kind]
            else Unchecked←NIL
                 (AffirmError "Restoring the previous set of rules." NIL T])

                                                                                          2

(Axioms
  [LAMBDA (Type)
    (for f in (Functions Type) join (Rules f <'axiom >])

                                                                                          3

(Define
  [LAMBDA (rule)                                               (* D.Thompson "23-Jun-80 10:56")

          (* * This routine processes the DEFINE command of AFFIRM. Its parameter is one parsed equation.)


    (PROG (definition interfaceTranslation unboundVariables)
          (interfaceTranslation←(Translate rule:RHS))
          (if interfaceTranslation:Operator='ExpressionWithType
              then (if ~(InterfaceDefined (MKLIST rule:LHS):Operator)
                       then (printout NIL "Providing an interface for" , (Name rule:LHS)
                                      Ellipses)              (* Name gives me operator)
                            (DeclareFun rule:LHS interfaceTranslation:Type)
                            (printout NIL T))
                   definition←(pexec rule T)
```

```
            (if (NoFreeVarsInRule definition RuleTypes:Define)
              else (AffirmError))
            definition← <definition:Operator definition:LHS (EVAL definition:RHS) >
            (printout NIL .TAB0 0 "define" , # (listOneRule definition:LHS RewriteRuleOp
                                                    definition:RHS CommandTerminator))
            (CompileRuleIntoLisp definition RuleTypes:Define T)
      else (printout NIL .TAB0 0 "Interface error:" , # (INFIX\PRINT3 interfaceTranslation)
                  T)
           (AffirmError])
```

4

```
(Distinct
  [LAMBDA (fns type)                                    (* R.Erickson "17-Sep-81 20:13")
                                                        (* fns are extended. type gives their range type)

    (* * Generate automatic rules stating that the fns return non-equal values. Also state that they are one-to-one.
    This is not conditional on existence of any other, conflicting rules. May be called repeatedly.)

    (PROG (lhsides pairs selfs neqs eqop rules neqrules selfrules)

          (* * compute pairs for comparison)

          (lhsides←(for fn in fns collect (GeneralLHS fn)))
                                                        (* suitable variable names. for current type)
          (pairs←(Combinations <lhsides lhsides>))      (* all pairs of function lh sides)
          (for pair in pairs bind ren do

          (* * separate out the one-to-one pairs from the rest. At the same time, rename vars as needed.)

                                                  (ren← <pair:1 (RenameArgs pair:2 pair:1:Arguments)
                                                        >)
                                                  (if pair:1=pair:2
                                                      then (push selfs ren)
                                                                    (* "<f(x) f(x')>")
                                                    else (push neqs ren)
                                                                    (* "<f(x) g(x' y)>")))

          (* * selfs and neqs have the two kinds of pairs. Compute the proposed rules.)

          (eqop←(GetEqualOp (CheckForType type T)))
          (neqrules←(for pair in neqs collect                (* the first set: "f(...) = g(...) -> FALSE")
                                          (<EQVOP <eqop pair:1 pair:2> FALSE>)))
          (selfrules←(for pair in selfs when pair:1:Arguments
                        collect

          (* the second set: "f(x,y) = f(x',y') -> x = x' and y = y'" Pair:1 = pair:2 except for renaming. We skip any cases where no
          args. NANDOP is nospread AND; depend on rules being simplified before entry.)

                                        (<EQVOP <eqop pair:1 pair:2>
                                                <NANDOP
                                                  !(for v1 in pair:1:Arguments as v2 in pair:2:Arguments
                                                      collect       (* "x = x'" note that v1 had better be declared.)
                                                              (<(GetEqualOp (RemoteTypeOf v1))
                                                                v1 v2>))
                                                  >>))))

          (* * Generate the rules. Put one-to-oneness rules first. Skip any already produced. Reverse, because we pushed,
          gives us the original constructor order for appearance.)

          (rules← < !(REVERSE selfrules) !(REVERSE neqrules)
                  >)
          (rules←(for rule in rules unless (EvaluateRule rule)=TRUE collect rule))
          (AddAutoRules rules 'distinct fns])
```

5

```
(Edit
  [LAMBDA (NewType)                                     (* R.Erickson " 8-Oct-79 15:43")
    (CLISP: UNDOABLE)
    NewType←(CheckForType NewType)
    (PRIN1 "type ")
    (RESETVARS ((DTVSCANON T))
```

```
            (INFIX\PRINT3 NewType))
    (TERPRI)
    (PrintDecls NewType:LocalDeclarations)
    (/SET 'TypeStack <NewType ! TypeStack>)
    (/SET 'CurrentContext NewType:LocalDeclarations)
    (/SET 'CurrentType NewType])
```

6

(NoChange

```
  [LAMBDA (selectors type)                          (* R.Erickson "22-Sep-81 17:41")
                                                    (* both extended. validated)

        (* * Generate the rules. Caller has checked that selectors all have some rule. Inform user of any lhs's we can't
        handle.)


    (PROG (unspecs rules cant)
          (unspecs←(for s in selectors join (UnspecifiedLHSides s)))

          (* * Compute the recursion rules. When not possible, collect NIL and put lhs on cant.)


          (rules←(for lhs in unspecs bind cands rhs
                    collect                         (* which variables are of the recursion type? All vars
                                                       in lhs are unique.)
                            (cands←(for v in (Variables lhs) when (TranslateLitAtom v):Type=type
                                      collect v))
                            (if (LENGTH cands)=1 and ~(MEMB cands:1 lhs:Arguments)
                                then                (* bingo. Want a rule
                                                    "f(x,...cands:1....,y) = f(x.cands:1,y)")
                                     rhs← <lhs:Operator !(for arg in lhs:Arguments
                                                          collect (if (OccursIn cands:1 arg)
                                                                      then cands:1
                                                                    else arg))
                                          >
                                     (create Equation
                                             Operator ← lhs:Operator:EqualOp
                                             LHS ← lhs
                                             RHS ← rhs)
                              else                  (* nope: none (eg. Init) or not unique
                                                    (eg, combo (s1,s2)) or cant simplify arg
                                                    (eg, f (s.off)))
                                   (push cant lhs)
                                   NIL)))
          (if cant
              then                                  (* warn user)
                   (printout NIL T "NoChange can't automatically handle these cases:")
                   (for lhs in cant do (printout NIL .TAB0 Tab)
                                       (PrettyPrint lhs T)))
          (AddAutoRules (REMOVE NIL rules)
                        'nochange selectors])
```

7

(Schema

```
  [LAMBDA (rule)                                    (* D.Thompson "22-Jun-80 21:40")

        (* * This routine processes the SCHEMA command of AFFIRM. Its parameter is one parsed equation.)


    (if ~(InterfaceDefined rule:LHS:Operator)
        then (DeclareFun rule:LHS 'Boolean))
    rule←(pexec rule T)
    (if (NoFreeVarsInRule rule RuleTypes:Schema)
        then (CompileRuleIntoLisp rule RuleTypes:Schema)
      else (AffirmError])
```

8

(adopt

```
  [LAMBDA (type)                                    (* R.Erickson "20-Feb-81 19:02")

        (* * This routine copies the 'root' (i.e., non-primed) declarations of another type over to the one currently being
        edited.
        An AFFIRM command function.)
```

```
    (PROG (decls)
          (if type←(CheckForType type)
              then decls←type:LocalDeclarations
                   (for dcl in decls do (ChooseASymbol dcl:OLD CurrentType dcl:NEW:Type))
                   (printout NIL .TAB0 0 "New environment:")
                   (PrintDecls CurrentContext)
            else (AffirmError])
```

9

```
(declareVar
  [LAMBDA (vars type call)                                  (* R.Erickson "25-Feb-81 19:50")

          (* * Declare "vars" to have type "type" within CurrentType. vars are short. We ignore primed vars, unless the root
          has not been declared or conflicts.)


          (* * call tells us the nature of our caller, and is used in case of error. Possibilities: -
          user :ask if conflict -
          internal :should never conflict -
          vc :tell of error in vc typechecking)


    (PROG (roots primes conflicts locals)                   (* roots will have unprimeds to be declared)
      RESTART
          (type←(CheckForType type NIL 'declare))

          (* * Filter out the primed vars)


          (roots←(for var in vars when (NPrimes var)=0 collect var))
                                                            (* unprimed)
          (primes←(for var in (LDIFFERENCE vars roots) bind root unless root←(ShortenLessPrimes
                                                                                var)
                                                                              MEMB roots
                    collect <var root>))                    (* primes with root not given)

          (* * Check for conflicts between roots and existing declarations. Create conflict list of <var oldType>)


          (conflicts←(for root in roots bind entry when entry←(FASSOC root CurrentContext)
                                                          and entry:NEW:Type~=type
                        collect <root entry:NEW:Type>))

          (* * If no conflicts yet, check for them with orphan primes. Record as <primed oldType root>)


          (if ~conflicts
              then conflicts←(for pair in primes bind entry when entry←(FASSOC pair:2 CurrentContext)
                                                                  and entry:NEW:Type~=type
                                collect <pair:1 entry:NEW:Type pair:2>))

          (* * Will declare roots if only primed given and no conflict)


          (roots←(REMOVEDUPLICATES < ! roots !(for pair in primes collect pair:2)
                                   >))
          (roots←(for root in roots unless (FASSOC root CurrentContext) collect root))

          (* * Warn of conflicts)


          (if conflicts
              then (SELECTQ call
                           (user                            (* if root, notify and ask to redeclare.
                                                            Else just notify.)
                                 (if conflicts:1:3
                                     then                   (* primed vs root)
                                          (CannedMessage 'varConflict T <'prime conflicts type>)
                                   else (CannedMessage 'varConflict NIL <'root conflicts type>)
                                        (PleaseDeclare NIL 'Discard)
                                        (GO RESTART)))
                           ((vc internal)
                             (CannedMessage 'varConflict T <call conflicts type>))
                           (CannedMessage 'key T <'declareVar call>))))

          (* * Check for interface conflicts)
```

```
        (conflicts←(for var in roots when (fetch PrimaryLHSides
                                             of (ExtendName (ExtendName var CurrentType)
                                                            'Interface))
                    collect var))
        (if conflicts
            then (CannedMessage 'varandFun T conflicts))

        (* * Do the declare)


        [locals←(for var in roots bind extended
                   collect (extended←(ExtendName var CurrentType))
                           (/SET extended extended)
                           (create SubPair
                                   OLD ← var
                                   NEW ←(create ExpressionWithType
                                                Expression ← extended
                                                Type ← type]
        (SELECTQ call
                 ((user internal)
                   (PrintDecls locals))
                 NIL)
        (AddDeclarations locals])
```

10

```
(discard
  [LAMBDA (kind targets)                                  (* R.Erickson "24-Feb-81 21:51")
                                                          (* Command function. Expects corrected key, uninterfaced
                                                          args. ? is handled by caller.)
    (SELECTQ kind
             (Disconnected (PROG (orphans named)          (* Delete any nodes without theorem parents.
                                                          Saves space.)
                                 (IgnoreExcess targets)
                                 (orphans←(for node in Nodes unless node:parents
                                                                    or (GetTheoremP node)
                                                                    or (True? node)
                                              collect node:prop#))
                                 (named←(for n in orphans bind name when name←n:name collect name))
                                 (if named
                                     then (AFFIRMMAPRINT "Discarding names" named)
                                          (ClearNames named))
                                 (if orphans
                                     then (printout NIL .TAB0 0 "Reclaiming" , (LENGTH orphans)
                                                    , "nodes."))
                                 (DeleteNodes orphans)))
             (History (IgnoreExcess targets)
                      (forget))
             [Interfaces (PROG (interface)
                               (for target in targets
                                  do (if interface←(MakeExtension target
                                                                  (MakeExtension CurrentType
                                                                                 'Interface))
                                         MEMB
                                         (InterfaceDefined target)
                                         then (discardInterface interface)
                                       else (AffirmError <target "does not have a Interface" > 'mild]
             [Lhs targets← (for target in targets collect (pexec target T))
                  (for target in targets do (if (FGETD target:Operator)
                                                then (RESETVARS ((UseRules T)
                                                                 (ListingFlag ('ZapRule))
                                                                 (REPORTFLAG T)
                                                                 (ReportWord "discarding "))
                                                                (APPLY target:Operator
                                                                       target:Arguments)
                                                                (if ListingFlag
                                                                    then
                                                          (* Report did not find the rule)
                                                                         (AffirmError
                                                                           <
"Rule lhs not found to delete for" target:Operator> 'mild)))
                                              else (printout NIL .TAB0 0 # (PrettyPrint target T)
                                                             "?")
                                                   (AffirmError "isn't an operator" 'mild]
             [Nodes                                       (* user typed "theorem", KnownObjects changed to Nodes)
                    (PROG (thms uses discard closure sorted place)
                          [thms←(REMOVE NIL (for target in targets
                                               collect (GetTheoremP target)
                                                  or (PROGN (printout NIL T "Not a theorem:" ,
                                                                      target)
```

```
                                                                NIL]

(* * We want to conservatively reorder the theorems so, if the user types "discard theorems a,b;"
and b uses a as a lemma, things will still go through by deleting b first.)

                    (uses←(for thm in thms collect <thm:prop# !(Facts thm)
                                                          >))
                    (discard←(REMOVEDUPLICATES (for thm in thms collect thm:prop#)))
                    (closure←(Closure uses))
                    (for new in discard do

(* put new at end of sorted, unless it should precede some entry. place is 1st element of sorted which is in closure
for new; since it is used as a lemma, should be preceded by new.)

                                                (place←(INTERSECTION sorted (ASSOC new closure)::1))
                                                        (* INTERSECTION preserves order of 1st arg.)
                                                (if place
                                                    then place←place:1
                                                            (* 1st critical item)
                                                         sorted← < !(UPTO place sorted)
                                                                   new !(place MEMB sorted) >
                                                  else sorted← < ! sorted new>))
                    (for d in sorted do (UnMakeTheorem (GetNode d]
        (Variables (discardVariable (for target in targets bind var
                                    when var←(if (FASSOC target CurrentContext)
                                                 then target
                                               elseif (IGEQ (NPrimes target)
                                                            1)
                                                 then (AffirmError
                                                        <"Can't discard a primed variable"
                                                          target "Do you mean"
                                                          (ShortenLessPrimes target) >
                                                        'mild)
                                                      NIL
                                               else (AffirmError <target "isn't a variable" >
                                                                 'mild)
                                                    NIL)
                                    collect var)))
        (AffirmError <"Unexpected kind in discard" kind> 'internal])
```

11

```
(discardInterface
  [LAMBDA (f DontPrint)                                    (* R.Bates "4-Sep-80 14:48")
    (PROG (type (shortF (Shorten f)))
          (if DontPrint=NIL
              then (AffirmError <shortF "has been re-declared.  "

"If it has been referenced in any definitions, axioms, lemmas, theorems
or schemas this type may be inconsistent. "
                                >
                               'mild))
          (/REMPROP f 'PrimaryLHSides)
          (/REMPROP type←(MakeExtension f (Shorten (Extension f)))
                    'PrimaryLHSides)
          (/PUTD f NIL)
          (DELFROMFILES f 'FNS)
          (/PUTD type NIL)
          (DELFROMFILES type 'FNS)
          (/PUTPROP shortF 'NoteInterface (REMOVE f (GETPROP shortF 'NoteInterface])
```

12

```
(discardVariable
  [LAMBDA (conflicts)
    (CLISP: UNDOABLE)                                      (* R.Erickson "19-Mar-81 15:40")

          (* * conflicts is a list of short variables to be undeclared from current type.)


    (if conflicts
        then (PROG ((ctname (Name CurrentType))
                    context)
                   (CannedMessage 'discard NIL '(variable))
                                                                (* warn user of possible inconsistencies)
                   (context←(for entry in CurrentContext collect entry
                                    when (if entry:OLD MEMB conflicts
```

```
                    then                    (* nonvariables should be NOBIND)
                         (/SET entry:NEW:Expression 'NOBIND)
                         NIL
                    else                    (* keep)
                       T)))                 (* remove discarded vars)
              (/SET 'CurrentContext ctname:LocalDeclarations←context])
```

13

```
(distinct
  [LAMBDA (ops)
    (CLISP: UNDOABLE)                       (* R.Erickson "23-Sep-81 17:44")
                                            (* optional list of short atoms.
                                            Parser should remove exprs.)

        (* * Command function. Specifies that the stated constructors of the current type (or all if omitted) are unequal,
        one-to-one. Saved and printed with type; rules generated immediately. May be repeated, with or without overlapping.)


    (PROG (fns alist old)

        (* * Determine targets)


        (fns←(if ops
                 then                       (* validate.)
                      (ExtendUsingList ops (Constructors)
                                       <"Not a constructor of type" CurrentType>)
             else

        (* * default. all.)

                      (Constructors)))      (* assert (if ops then ops = (Shorten fns) else fns =
                                            (Constructors)))

        (* * Save on property, determine consistency with preexisting)


        (old←CurrentType:Distinct)
        (if old
            then

        (* * already had something)

                 (if ops
                     then                   (* user specified)
                          (if old:1=T
                              then          (* conflict)
                                   (CannedMessage 'incompatible T <'distinct CurrentType>)
                            elseif (MEMBER fns old)
                              then          (* done before)
                                   (RETURN)
                              else (push CurrentType:Distinct fns))
                   else                     (* all)
                        (if old:1~=T
                            then (CannedMessage 'incompatible T <'distinct CurrentType>)
                           elseif (EQUAL old:2 fns)
                             then           (* did same constructors before)
                                  (RETURN)
                           else old:2←(REMOVEDUPLICATES < ! fns ! old:2>)
                                            (* constructors we missed last time)))
          else

        (* * normal case: first spec for this type)


               (if ops
                   then                     (* only some)
                        CurrentType:Distinct← <fns>
                    else                    (* all. record which we use.)
                         CurrentType:Distinct← <T fns>))

        (* * Produce the rules.)


        (Distinct fns CurrentType])
```

14

```
(nochange
  [LAMBDA (selectors type)
    (CLISP: UNDOABLE)                                    (* R.Erickson "23-Sep-81 17:43")
                                                         (* both are short, optional.)

          (* * Command function. We want each of the selectors to be defined for all args build of data type constructors.
          Where possible, generate automatic axioms for the missing cases, stating that the selctor is unchanged across the
          constructor. -
          Default is all selectors, CurrentType. May be repeated.)


      (PROG (all voids old entry)                        (* validate args.)
            (type←(if type
                      then (CheckForType type)
                    else CurrentType))
            (if selectors
                then selectors←(ExtendUsingList selectors (Selectors)
                                                <"Not a selector of type" CurrentType>)
              else selectors←(Selectors)
                   all←T                                 (* all selectors of type.)
                   )                                     (* now we have extended names)

          (* * Check that the selectors all have some rules; otherwise, NoChange is probably not intended.)


          (voids←(for s in selectors unless (LeftHandSides s) collect s))
          (if voids
              then                                       (* warn user, continue for rest if any.)
                   selectors←(LDIFFERENCE selectors voids)
                   (AffirmError <"NoChange not meaningful for" (Shorten voids) > (if selectors
                                                                                     then
                                                         (* continue)
                                                                                          'mild)))

          (* * Record on property, check consistency with previous. Format of :NoChange is a list of tuples, :1 is recursion
          type, :2 if all selectors, :3 selectors actually used.)


          (if old←CurrentType:NoChange
              then entry←(ASSOC type old))
          (if ~entry
              then                                       (* first time for this recursion type)
                   (push CurrentType:NoChange (<type all selectors>))
            elseif all
              then                                       (* override/overlap earlier)
                   entry:2←T
                   entry:3←(REMOVEDUPLICATES < ! entry:3 ! selectors>)
            elseif entry:2
              then                                       (* Previous was all; can't give specific after general)
                   (CannedMessage 'incompatible T <'nochange CurrentType>)
            else (entry:3←(REMOVEDUPLICATES < ! entry:3 ! selectors>)))
          (NoChange selectors type])
```

15

```
(set
  [LAMBDA (var exp)                                      (* R.Erickson "15-Nov-79 13:29")

          (* * This routine enables the user to set a variable to a value. -
          An AFFIRM command function.)


    exp←(Translate exp)
    exp←exp:Expression
    exp←(EVAL exp)
    (/SET var exp)
    (PrettyPrint <EQOP var exp>])
)
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

(* Data types)]

```
(RPAQQ TypeQueries (Constants Constructors Functions IsType Name NonConstructorsUsed Selectors))
(DEFINEQ
```

16

```
(Constants
  [LAMBDA (Type)                                        (* R.Bates "13-Dec-79 17:09")
    (EVAL (PACK* (Name Type)
                 'Constants])
```

17

```
(Constructors
  [LAMBDA (type)                                        (* R.Erickson "26-Aug-81 14:52")

          (* * Returns a list of the extended names of the functions in type which have range in type and no rules.)

    (PROG (cands constrs)
          (if ~type
              then type←CurrentType)
          (cands←(for ifun in (Functions type T) bind rhs eachtime rhs←(Rules ifun
                                                                            RuleTypes:Interface)
                                                                     :1:RHS
                      when rhs:Type=type collect rhs:Expression:Operator))
                                                        (* functions with range in this type.
                                                        ASSUME: only 1 interface rule in the type)
          (constrs←(for fn in cands unless (Rules fn T) collect fn))
                                                        (* names of those with no axiom/defn/rulelemma...)
          (RETURN constrs])
```

18

```
(Functions
  [LAMBDA (Type Interface?)                             (* D.Thompson "19-Aug-80 11:09")
    (REVERSE (EVALV (if Interface?
                        then (MakeExtension (Name Type)
                                            'InterfaceFNS)
                      else (PACK* (Name Type)
                                  'FNS])
```

19

```
(IsType
  [LAMBDA (x)                                           (* R.Erickson " 3-Aug-81 18:42")

          (* * This routine determines if its parameter is a type name or not.)

    (CAR ((Name x)
          MEMB KnownNames:Types])
```

20

```
(Name
  [LAMBDA (x)                                           (* Edited by R.Bates on 24-JAN-78;
                                                        from version 43)
    (if (NLISTP x)
        then x
      else x:Operator])
```

21

```
(NonConstructorsUsed
  [LAMBDA (lhs)                                         (* R.Erickson "21-Sep-81 15:39")
                                                        (* lhs an expression with extended names.
                                                        Won't work for IFOP, QOP.)

          (* * Does lhs:Arguments contain any functions which are not data type constructors? Return them.)

    (PROG (types constrs bads)                          (* constrs is union of constructor sets for types. those
```

```
                                                        computed so far.)
        [for fn in (REMOVEDUPLICATES (Functions lhs:Arguments)) unless (MEMB fn constrs)
           bind ext do (ext←(Extension fn))
                       (if ext MEMB types
                           then                          (* already have constructors.)
                                (push bads fn)
                          else (push types ext)
                               (pushlist constrs (Constructors ext))
                               ((MEMB fn constrs) or (push bads fn]
        (RETURN (REVERSE bads])
```

22

```
(Selectors
  [LAMBDA (type)                                        (* R.Erickson "26-Feb-81 19:00")

          (* * Returns a list of the extended names of all functions in type with range outside that type)


     (if ~type
         then type←CurrentType)
     (for ifun in (Functions type T) bind rhs eachtime rhs←(Rules ifun RuleTypes:Interface):1:RHS
        when rhs:Type~=type collect rhs:Expression:Operator])
)

(RPAQQ TypeMods (DeclareFun DeclareType GenerateReflexiveRule MakeConstant MinimalTypeSpec))
(DEFINEQ
```

23

```
(DeclareFun
  [LAMBDA (ex type DontPrint)                           (* R.Erickson "25-Aug-81 13:47")

          (* * Creates an interface such that ex returns the given type.)


     (PROG (varex argnames irule translated)            (* (check legality of name and arguments))
           (if (NLISTP ex)
               then (if ~(LITATOM ex) or ex=TRUE or ex=FALSE
                        then (AffirmError <"*** can not declare " ex>))
                    ex← <ex>)
           (if (ASSOC (ShortenLessPrimes ex:Operator)
                      CurrentContext)
               then (CannedMessage 'varandFun T <ex:Operator>)
             elseif (NPrimes ex:Operator)
                    ~=0
               then (AffirmError <"Can't use primes in a function name:" (Shorten ex:Operator)
                                 >))
           (for arg on ex:Arguments thereis (if (MEMBER arg:1 arg::1)
                                                then (AffirmError <"Can't use the same variable twice:"
arg:1>)
                                                     T))

          (* * We want only variables in varex, the template for the interface. We need to compute variable names for the
          nonsimple arguments)


          (translated←(for arg in ex:Arguments collect (Translate arg)))
          (argnames←(for arg in translated collect          (* We want (Translate arg) if a var, the type otherwise)
                                                  (if (LITATOM arg:Expression)
                                                      then arg
                                                    else arg:Type)))
          (argnames←(ArgumentVars argnames ex:Operator))
          (varex← <ex:Operator !(Shorten argnames)
                  >)
          (/replace EqualOp of (ExtendName ex:Operator (Name CurrentType)) with (GetEqualOp type))
                                                          (* compute equality operator for the fn)

          (* * Now, make the interface.)


          (irule←(MakeInterfaceRule varex type (Name CurrentType)
                                    translated DontPrint))
          (CompileRuleIntoLisp irule 'interface T)
          (NoteInterfaces <irule:LHS:Operator>])
```

24

```
(DeclareType
  [LAMBDA (x DontRedeclareComs)
    (CLISP: UNDOABLE)                                    (* R.Erickson "24-Aug-81 16:25")

        (* * This routine initializes the data structure associated with a type. Most of the structure is contained on the
        property list of the type name atom, but some of the structure is stored as the values of other atoms formed by
        packing varioous things onto the ucased type name atom.)


    (PROG (typeName toFun eqop)
          (typeName←(Name x))
          (if typeName:DeclaredType
              then (for n in (Functions x) do (if ~DontRedeclareComs
                                                  then      (* In a compiled file the new compiled definitions are
                                                            already read in, therefore we can't do the /PUTD to
                                                            NIL.)
                                                       (/PUTD n NIL))
                                                  (/SETPROPLIST n NIL))
                       [for n in (Functions x T) do (if ~DontRedeclareComs
                                                        then (/PUTD n NIL))
                                                    (/SETPROPLIST n NIL)
                                                    (toFun←(Shorten n))
                                                    (/PUTPROP toFun 'NoteInterface
                                                              (REMOVE n (GETPROP toFun 'NoteInterface]
            else (if ~DontRedeclareComs
                     then                                (* check for valid name)
                          (ValidateFileName typeName 'Types T)))
          (pushnew KnownNames:Types typeName)
          (if ~DontRedeclareComs
              then (if (NLISTP x)
                       then (MakeConstant x)
                       else (MakeFunction x)))
          (typeName:DeclaredType←x)
          (typeName:LocalDeclarations←NIL)
          (typeName:Infix←NIL)
          (typeName:Needs←NIL)
          (typeName:Distinct←NIL)
          (typeName:NoChange←NIL)
          (eqop←(ExtendName EQOP typeName))
          (if ~DontRedeclareComs
              then (MakeFunction <eqop 'x 'y >))
          (typeName:EqualOp←eqop)
          (eqop:EQOP←T)
          (eqop:EqualOp←(GetEqualOp 'Boolean))
          (if DontRedeclareComs=NIL
              then (SetupFile typeName])
```

25

```
(GenerateReflexiveRule
  [LAMBDA NIL                                            (* R.Erickson " 4-Aug-81 13:22")

        (* * Produce the axiom "x = x -> true" for the current type, declaring the variable too.)


    (PROG (var trVar)
          (var←(UserProfile 'DummyVarName))              (* typically "dummy")
          (declareVar <var> CurrentType 'internal)
          (trVar←(pexec var T))
          (AddAutoRules <<EQVOP <(GetEqualOp CurrentType)
                                  trVar trVar>
                                TRUE>>
                        'reflexive)
          (RETURN CurrentType])
```

26

```
(MakeConstant
  [LAMBDA (x)                                            (* R.Erickson "20-Sep-79 18:01")
    (PROG (Ex)
          (x:IsConstant←T)
          (/SET x x)
          (Ex←(Extension x))
          (if Ex
              then (AddToFile x Ex 'Constants])
```

27

```
(MinimalTypeSpec
  [LAMBDA (typeName dontPrintMessage)                    (* D.Thompson "10-Sep-80 12:52")

          (* * This routine provides a minimal type specification for the type name provided as the parameter.
          the name is returned.)


    (if dontPrintMessage
      else (printout NIL .TAB0 0 "Providing minimal specification for type" , typeName Period T))
    (ProcessCommand 'type <typeName> Terminal 'internal)
    (ProcessCommand 'end NIL Terminal 'internal)
    typeName])
)
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

(* Typechecking and interfaces)]

```
(RPAQQ InterfaceQueries (InterfaceDefined NoFreeVarsInRule PexecLists PrintDecls RangeType
                                          RemoteTypeOf Translate TranslateLitAtom TypeOfExpression
                                          pexec typeify))
(DEFINEQ
```

28

(InterfaceDefined

```
  [LAMBDA (op)                                          (* R.Erickson "21-Jan-80 17:20")

          (* * True if any interface is known for the short name op.)

    (GETPROP op 'NoteInterface])
```

29

(NoFreeVarsInRule

```
  [LAMBDA (rule kind)                                   (* D.Thompson "22-Aug-80 17:05")
    (PROG (unboundVariables)
          (RETURN (if unboundVariables←(LDIFFERENCE (Frees rule:RHS)
                                                    (Frees rule:LHS))
                      then (AFFIRMMAPRINT (CONCAT "You've used unbound "
                                                  (Plural 'variables unboundVariables))
                                          (for v in unboundVariables collect (Shorten v))
                                          T
                                          (CONCAT " on the right hand side of "
                                                  (SELECTQ kind
                                                           (axiom "an axiom")
                                                           (defn. "a definition")
                                                           ((lemma rulelemma)
                                                             "a rulelemma")
                                                           (schema "a schema")
                                                           (PROGN (Unexpected 'RuleType)
                                                                  (CONCAT "an unexpected kind (" kind
                                                                          ") of rule!!")))
                                                  Period))
                              NIL
                      else T])
```

30

(PexecLists

```
  [LAMBDA (expression notNIL evenAtoms)                 (* R.Erickson " 4-Mar-80 13:48")

          (* If arg is a list, calls pexec. This is useful for prop names and numbers, to bypass the interface mechanism.
          evenAtoms forces us to always call pexec; useful for nilary fns. notNIL causes us to AffirmError rather than return
          NIL if pexec fails.)

    (if (LISTP expression) or evenAtoms
        then (pexec expression) or (AffirmError)
      elseif expression
      elseif notNIL
        then (AffirmError "No parameter!"])
```

31

(PrintDecls

```
  [LAMBDA (alist)                                       (* D.Thompson "31-Oct-79 19:30")

          (* * This routine prints the 'local declarations' provided as its parameter -
          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 ((grouped (< ! '((#))
                    >)))
          (for vt in alist bind t eachtime t←vt::1:Type do (PUTASSOC t < !!(FASSOC t grouped)::1
                                                                        (RenameAtoms vt:1) >
                                                           grouped))
          (for g in grouped unless g:1='# do (printout NIL .TAB0 0 # (MAPRINT g::1 T NIL ": " ", ")
                                             #
                                             (PrettyPrint g:1 T)
```

```
                                                                T])
```

32

```
(RangeType
  [LAMBDA (fn)                                          (* R.Erickson "17-Sep-81 19:51")
                                                        (* atom\TypeName)

        (* * Type which it returns. Assume only 1 interface rule.)

    (fetch Type of (fetch RHS of (CAR (Rules (ExtendName fn IEXT)
                                             RuleTypes:Interface])
```

33

```
(RemoteTypeOf
  [LAMBDA (var)                                         (* R.Erickson "24-Feb-81 21:29")

        (* * Given an extended variable, go off to the declarations of the type to which it belongs
        (determined by its extension) ,and compute its declared type.)

    (PROG (decls)
          [decls←(fetch LocalDeclarations of ((Extension var) or (AffirmError
                                                                   <"No extension:" var> 'internal]
          (RETURN (fetch Type of (CDR (FASSOC (ShortenLessPrimes var)
                                              decls])
```

34

```
(Translate
  [LAMBDA (expression)                                  (* R.Bates "25-Feb-80 14:45")

        (* * put Expression, which has short names. thru the interface mechanism. Get an ExpressionWithType if successful.)

    (if (NLISTP expression)
        then (if (LITATOM expression)
                 then (TranslateLitAtom expression)
               elseif (NUMBERP expression)
                 then <'ExpressionWithType expression (if (FIXP expression)
                                                          then 'Integer
                                                        else 'Real)
                      >
               elseif (STRINGP expression)
                 then <'ExpressionWithType expression 'String >
               else (Unexpected 'ExprType))
      elseif (type? ExpressionWithType expression)
        then expression
      else (TranslateApply 'GenericInterface 'Translate expression:Arguments expression:Operator])
```

35

```
(TranslateLitAtom
  [LAMBDA (atom varProbe)                               (* R.Erickson "24-Feb-81 20:40")

        (* * Puts a short literal atom through the interface mechanism. Checks for variable, nilary interface, or calls
        DWIM. "varProbe" flag causes only var check to be done, returns NIL if not found.)

    (if atom MEMB '(TRUE FALSE)
        then (create ExpressionWithType
                     Type ←('Boolean)
                     Expression ← atom)
      else (PROG (shortless entry nprimes)
                 (shortless←(ShortenLessPrimes atom))       (* Short name of the root variable)
                 (entry←(ASSOC shortless CurrentContext))
                 (if entry
                     then                                   (* hit, root is declared. Add primes as necessary)
                          (RETURN (create ExpressionWithType
                                          Expression ←(PrimesToMatch entry:NEW:Expression atom)
                                  using entry:NEW))
                   elseif varProbe
                     then                                   (* not declared as variable, fail, return NIL)
                          NIL
                   else                                     (* See if it's a parameterless function or ask to
```

declare.)

```
                    (RETURN (GenericInterface atom])
```

36

```
(TypeOfExpression
  [LAMBDA (ex foreignType)                          (* R.Erickson "29-Jan-80 13:12")

          (* * puts ex thru the interface process, causing errors as necessary. Returns the type of the result -
          all this is in the current type. Does not know about IMPLIST. -
          foreignType may be specified to cause us to use a different variable context. This might cause trouble in case we
          call DWIMUSERFN. This option is used by Rule to compute EqualOp for anomolous cases.)


    (PROG (trans)
          (if foreignType and foreignType~=CurrentType
              then (RESETVARS ((CurrentContext (foreignType:LocalDeclarations)))
                              (trans←(Translate ex)))
            else trans←(Translate ex))
          (if (type? ExpressionWithType trans)
              then (RETURN trans:Type)
            else (printout NIL T "Syntax error:" ,)
                 (PrettyPrint trans)
                 (ERROR!])
```

37

```
(pexec
  [LAMBDA (Expression notNil)                       (* R.Erickson "18-Dec-79 11:30")
    (PROG (Tran)
          (Tran←(Translate Expression))
          (if Tran:1='ExpressionWithType
              then Tran←Tran:2
                   (for i in IMPLIST do Tran←(APPLY* 'Using Tran (NameSubList i)))
                   (RETURN Tran)
            else (Heading "**** Interface error")
                 (PrettyPrint Tran)
                 (if notNil
                     then (AffirmError)             (* else return NIL to caller)])
```

38

```
(typeify
  [LAMBDA (Expression)                              (* R.Erickson "28-Sep-79 15:23")

          (* * called by the VCGen package: typeifyStatement.)


    (if Expression
        then (pexec Expression])
)

(RPAQQ InterfaceMods (AddDeclarations MakeInterfaceRule))
(DEFINEQ
```

39

```
(AddDeclarations
  [LAMBDA (newlocals)
    (CLISP: UNDOABLE)                               (* D.Thompson " 3-Jan-80 14:20")
                                                    (* given a declaration or list of such, add them to the
                                                    declarations of the current type, and to the current
                                                    context.)
    (PROG ((ctname (Name CurrentType)))
          (if newlocals:2='ExpressionWithType
              then newlocals← <newlocals>)
          (/SET 'CurrentContext ctname:LocalDeclarations← < !! ctname:LocalDeclarations
                                                            ! newlocals>)
          (AddNeeds 'Types (for e in newlocals collect e::1:Type)
                    CurrentType T])
```

40

```
(MakeInterfaceRule
  [LAMBDA (function type context translated DontPrint)   (* R.Bates "26-Sep-80 10:20")
    (PROG (ifun)
```

```
        (ifun←(ExtendName (MakeExtension function:Operator context)
                          'Interface))
        [if ifun MEMB (interfaceDefined (Shorten ifun))
            then (if DontPrint
                     then (discardInterface ifun DontPrint)
                   else (while (FGETD ifun) do (PRINTBELLS)
                                               (DOBE)
                                               (CLEARBUF T T)
                                               (printout NIL .TAB0 0
                                                         "Already have an interface for"
                                                         ,
                                                         (Shorten ifun)
                                                         Period T
                                "To proceed, discard interface (then type ok;), or abort; to stop."
                                                         T)
                                               (if (Dtvs Terminal)='aborted
                                                   then (ERROR!]
        (/PUTD ifun <'LAMBDA < ! function:Arguments 'TooManyArguments > <'COND <'NIL >>>)
        (AddToFile ifun (Extension ifun)
                   'FNS)
        (if function:Arguments
          else ((InsertExtensions function context):Operator:IsConstant←T))
        (RETURN (create RewriteRule
                        LHS ←(<ifun ! < !! translated NIL>>)
                        RHS ←(create ExpressionWithType
                                     Expression ←(InsertLISTs (MakeFunction
                                                                 (InsertExtensions function context))
                                                              )
                                     Type ← type])
)

(RPAQQ InterfaceInternals (ExpressionWithType GenericInterface InsertExtensions InterfaceError
                                              cases\Interface cases\Schema))
(DEFINEQ

                                                                                          41

(ExpressionWithType
  [LAMBDA (Expression Type)
    <'ExpressionWithType Expression Type>])

                                                                                          42

(GenericInterface
  [LAMBDA N                                                  (* D.Thompson " 3-Sep-80 16:00")

        (* * This routine attempts to resolve user input to type-specific references, using a bottom-up walk of the
        expression parse tree. If no interfaces match, the interface error routine is called, which tries out a few
        heuristsics, maybe asks the user, and perhaps fails.)


    (PROG (arg1 default interfaces keyList listTypes optionList question second temp value)
      TOP (interfaces←(GETPROP arg1←(ARG N 1)
                               'NoteInterface))              (* find all the interface declaration that apply.)
          (for fun in interfaces
             do (if (FNTYP fun)
                    then (if temp←(NoSpreadApply* fun N 8)
                             then (if value=NIL
                                      then value←temp
                                    elseif second=NIL
                                      then value← <temp value> second←T
                                    else value← <temp ! value>))
                  else (AffirmError <"The name" fun "should be a defined interface, but isn't!" >
                                    'mild)))
          (if value=NIL
              then (RETURN (InterfaceError (for i from 1 to N collect (ARG N i))
                                           interfaces))
            elseif second=NIL
              then (RETURN value)
            else (if (for v in value::1 always (EQUAL value:1 v))
                     then                                    (* all the values were the same so it doesn't matter
                                                             which one we pick.)
                          (RETURN value:1)
                   else (printout NIL .TAB0 0 # (PrettyPrint value:1:Expression T)
                                  , "is defined in types" , #
                                  (AFFIRMMAPRINT NIL listTypes←(for expr in value
                                                                  collect (if (Extension
                                                                                expr:Expression:Operator)
                                                                              else
```

```
                                                        expr:Expression:Operator))
                                        T T))
                optionList←NIL
                question←"Which type do you mean here? " keyList←
                < !(for expr in value as typeName in listTypes collect
                                                        <typeName "" 'RETURN
                                                          (KWOTE expr)
                                                          >)
                  <'pause " (in a lower exec)" >>
                default←(if CurrentType MEMB listTypes
                           then (for typeName in listTypes as expr in value
                                   bind currentExpr eachtime currentExpr←expr
                                   until typeName=CurrentType do T
                                   finally (RETURN currentExpr))
                         else NIL)
                value←(AFFIRMUSER NIL default question keyList NIL NIL optionList)
                (if value~='pause
                    then (RETURN value))
                (printout NIL .TAB0 0 LeftParenthesis "Type  ok;  to continue"
                          RightParenthesis T)
                (if (Dtvs Terminal)='aborted
                    then (ERROR!))
                (GO TOP])
```

43

(InsertExtensions

```
  [LAMBDA (Expression Extension)                    (* R.Bates "21-Feb-80 15:29")
    <(ExtendName Expression:1 Extension) !(for argument in Expression::1
                                             collect (TranslateLitAtom argument):Expression)
    >])
```

44

(InterfaceError

```
  [LAMBDA (expression interfaces)                   (* R.Erickson "24-Jun-81 17:22")
                                                    (* return a result as translated as possible.)

        (* * This routine attempts to "fix" interface errors. These arise from several sources: -
        (1) fn is undefined or misspelled; -
        (2) type mismatch; -
        (3) to few or too many parameters; -
        (4) some horrible combination of the above. -
        This routine currently doesn't try anything, but just falls through to the bottom line processing.)


    (PROG (arguments expectedLength givenLength interface interfaceOperator operator solution
                 varEntry)

        (* (solution← (if interfaces then (* fn is defined. May or may not be misspelled, but for now assume is spelled OK.)
        (if (EQLENGTH interfaces 1) then interface←interfaces:1 (if expectedLength← (FLENGTH interface) -2- = givenLength←
        (FLENGTH expression) -
        2 then (* too few or too many parameters) (* * dummy for now: (printout NIL .TAB0 0 interface:1 , "expects",
        expectedLength , (Plural "arguments" expectedLength) Colon T # (listOneRule interface) T "but it was given",
        givenLength Colon T # # (INFIX\PRINT3 expression) T)) NIL else (* type mismatch) NIL) else (* more than one
        possibility for this interface) NIL) else (* fn is undefined or misspelled) NIL)))

                                                    (* We assume that caller has taken care of nilary
                                                    interface conversion)
        (RETURN (if solution
                  else                              (* We were unable to spelling corect above, so go to a
                                                    lower exec to ask user to declare.
                                                    Will ERROR! if unwilling.)
                       operator←expression:1
                       arguments←expression::1
                       (if (EQLENGTH expression 1)
                           then                     (* undeclared variable)
                                (PleaseDeclare operator 'Variable)
                                (Translate operator)
                          elseif varEntry←(ASSOC (Shorten operator)
                                                 CurrentContext)
                           then                     (* user entered a Skolem expression, for some reason)
                                (create ExpressionWithType
                                        Expression ←(<varEntry:NEW:Expression
                                                     !(for a in arguments
                                                         collect (if a:1='ExpressionWithType
                                                                     then a:Expression
                                                                  else (AffirmError "Not typed"
                                                                                    'internal)))
```

```
                        >)
                    Type ← varEntry:NEW:Type)
            elseif (GETPROP operator 'NoteInterface)
              then                              (* already has some interface.
                                                Generate error.)
                   <(MakeExtension operator 'Interface) ! arguments>
            else                                (* missing interface)
                 (PleaseDeclare operator 'Interface)
                 (Translate expression])
```

45

```
(cases\Interface
  [LAMBDA c
    (if (for i from 1 to c always (ARG c i):Operator='ExpressionWithType)
        then <'ExpressionWithType <'cases\Schema !(for i from 1 to c collect (ARG c i):Expression) >
                                  ' Boolean >])
```

46

```
(cases\Schema
  [LAMBDA c <'cases\Schema !(for i from 1 to c collect (ARG c i))
            >])
)
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

(* other utilities)]

```
(RPAQQ SpecFiles (InitializeLoad NoteDeclarations NoteInterfaces SetupFile))
(DEFINEQ
```

47

```
(InitializeLoad
  [NLAMBDA (category%  name%  affirmversion%  list% )        (* R.Erickson "25-Feb-81 19:39")

        (* * This routine performs the initialization functions for types loaded from files. It also worries about files
        saved from OLD systems, with different formats.)


    (if affirmversion%  lt 35
        then                                                 (* compatibility of infix operator data moved to
                                                             property list as of AFFIRM-35)
             (PROG (infixOps infixVar)
                   (infixVar←(PACK* name%  'InfixOps))
                   (if infixOps←(EVALV infixVar)='NOBIND
                       then infixOps←NIL)
                   (name% :Infix←(MKLIST infixOps))
                   (initInfix name% )))
    (if affirmversion%  lt 36
        then                                                 (* The data type has the old form for its left-hand-side
                                                             properties. Convert it, telling the user if
                                                             appropriate.)
             (ConvertOldType name% ))
    (if affirmversion%  lt 122 and category% ='TYPE
        then                                                 (* LocalDeclarations has extended names. convert to
                                                             short and remove primeds)
             name% :LocalDeclarations←(for pair in name% :LocalDeclarations when (NPrimes pair:OLD)=0
                                         collect (create SubPair
                                                         OLD ←(Shorten pair:OLD) using pair)))
    (SELECTQ category%
             (TYPE (pushnew KnownNames:Types name% )
                   (for l in list%  do (EVAL l)))
             (SHOULDNT])
```

48

```
(NoteDeclarations
  [LAMBDA (type)                                             (* R.Erickson " 7-Feb-80 15:26")
    (for v in type:LocalDeclarations do (/SET v:3 v:3])
```

49

```
(NoteInterfaces
  [LAMBDA (ifns)                                             (* R.Bates "26-Jun-80 21:07")
    (for (f shortF) in ifns do (if f ~MEMB (GETPROP shortF←(Shorten f)
                                                    'NoteInterface)
                                       then (/ADDPROP shortF 'NoteInterface f])
```

50

```
(SetupFile
  [LAMBDA (typeName)                                         (* R.Erickson "17-Mar-81 19:11")

        (* * This routine creates the data structure that remembers all the parts of the type. The routine currently spreads
        this data structure over many variables, rather than concentrating it on the property list of the type.)


    (PROG (coms constants file fns icons ifns ivars vars)
          (file←(U-CASE typeName))
          (coms←(PACK* file 'COMS))
          (if (LISTP (EVALV coms)) or (GETPROP file 'FILEDATES) or (GETPROP file 'FILE)
              then (printout NIL .TAB0 0 "(Redefining" , typeName , ")" T)
                   (/REMPROP file 'FILEDATES)
                   (/REMPROP file 'FILE))
          (fns←(PACK* typeName 'FNS))
          (ifns←(ExtendName typeName 'InterfaceFNS))
          (constants←(PACK* typeName 'Constants))
          (icons←(ExtendName typeName 'InterfaceConstants))
          (/SET coms <<'P <'CheckLoad '(QUOTE TYPE)
                                      (KWOTE <AFFIRMVERSION ! AFFIRMSYSOUTFILE>)
                                      (KWOTE typeName)
```

```
                        >> <'FNS
                      '* fns> <'FNS '* ifns> <'VARS '* constants> <'VARS '* icons>
                      <'IFPROP 'ALL '* constants> <'IFPROP SavedFnProps '* fns>
                      <'IFPROP SavedFnProps '* ifns> <'P <'InitializeLoad 'TYPE typeName
                                                          AFFIRMVERSION
                                                          <<'NoteInterfaces ifns>
                                                            <'initInfix (KWOTE typeName) >
                                                            <'initNeeds (KWOTE typeName) >
                                                            <'NoteDeclarations (KWOTE typeName)
                                                              > <'NoteLeftHandSides fns>>>>>)
        (/SET fns (if typeName MEMB '(Boolean BUILTIN)
                      then                                  (* Builtin types treated specially: Boolean uses
                                                            EQV\Boolean; BUILTIN uses Equal, on file Ceval)
                           NIL
                    else <(GetEqualOp typeName)
                      >))
        (/SET ifns NIL)
        (/SET constants <typeName>)
        (/SET icons NIL)
        (if file ~MEMB FILELST
            then (/SET 'FILELST <file ! FILELST>))
        (/PUTPROP file 'FILE <<coms ! T> file>)          (* beats me why this is done -- RE)
        (/PUTPROP file 'FILEDATES NIL)                   (* Causes MAKEFILE to be NEW)
      ])
)

(RPAQQ SpecOther (AtomsOfType Infix RenameAtoms RetrieveRules ShortenAllAtoms initInfix))
(DEFINEQ
```

51

```
(AtomsOfType
  [LAMBDA (ex type)                                      (* Edited by R.Bates on 24-JAN-78;
                                                         from version 43)
    (if (NLISTP ex)
        then (if ~(ex : IsConstant) and (fetch Type of (Translate (RenameAtoms ex)))=type
                 then <ex>
               else NIL)
      else (for a in ex:Arguments join (AtomsOfType a type])
```

52

```
(Infix
  [LAMBDA (opList typeName)                              (* D.Thompson "27-Aug-80 11:29")

        (* * This routine adds the ops in opList to the data structures recording the fact that each op is an infix
        operator.)


    (PROG (remainingOps renamedOps)
          (if typeName~=T
              then renamedOps←(for op in opList collect (if (MapToInternal op)
                                                            else op))
                   typeName:Infix←(REMOVEDUPLICATES < ! typeName:Infix ! renamedOps>)
                   remainingOps←(INTERSECTION opList renamedOps)
                   (SetHierarchy remainingOps 2)
                   [for op in remainingOps
                      do (if op MEMB #LIST\OF\SYSTEM\OPS
                             else #LIST\OF\SYSTEM\OPS← < ! #LIST\OF\SYSTEM\OPS op>)
                         (if op MEMB #NO\LEFT\ASSOC\OPS
                           else #NO\LEFT\ASSOC\OPS← < ! #NO\LEFT\ASSOC\OPS op>)
                         (if (GETPROP 'TTYCHARSET op)
                           else (if (NCHARS op)=1
                                    then (PUTPROP 'TTYCHARSET op op)
                                  else (PUTPROP 'TTYCHARSET op (PACK* Blank op Blank]
            else

        (* * The case where typeName = T corresponds to the old (pre version 35) method of initializing infix operators, and
        is now obsolete)


                NIL])
```

53

```
(RenameAtoms
  [LAMBDA (x)                                            (* R.Erickson "19-Dec-79 13:21")
```

```
(* * This is like Shorten. except it returns the OriginalName property. if it exists. This prop is set by hand. for
things like ExpressionWithType. Equal. = \Interface. and IfThenElse; it is used for printing ONLY.)

    (if (LITATOM x)
        then (if x:OriginalName
                elseif x=NIL
                  then NIL
                else (Shorten x))
      elseif (NLISTP x)
        then x
      else (for y in x collect (RenameAtoms y])
```

54

```
(RetrieveRules
  [LAMBDA (TypeName Kind)
    (EVAL (ExtendName Kind TypeName])
```

55

```
(ShortenAllAtoms
  [LAMBDA (x)                                          (* Edited by R.Bates on 24-JAN-78;
                                                       from version 43)

    (if (LITATOM x)
        then (Shorten x)
      elseif (NLISTP x)
        then x
      else (for y in x collect (ShortenAllAtoms y])
```

56

```
(initInfix
  [LAMBDA (typeName)                                   (* D.Thompson "18-Dec-79 17:22")
    (Infix typeName:Infix typeName])
)

(RPAQQ SpecLimbo (AssociatedType InsertInterfaces InsertLISTs LegalityChecks Macro NameSubList Quote
                                 SubstTypes Using declareMacro))
(DEFINEQ
```

57

```
(AssociatedType
  [LAMBDA (Imp)
    (GETPROP Imp 'Type])
```

58

```
(InsertInterfaces
  [LAMBDA (Expression)                                 (* R.Erickson "26-Jan-80 14:09")
    NIL])
```

59

```
(InsertLISTs
  [LAMBDA (x)                                          (* Edited by R.Bates on 24-JAN-78;
                                                       from version 16)

    (if (NLISTP x)
        then x
      else <'LIST (KWOTE x:Operator) !(for y in x:Arguments collect (InsertLISTs y))
           >])
```

60

```
(LegalityChecks
  [LAMBDA (x)                                          (* Edited by R.Bates on 24-JAN-78;
                                                       from version 43)

    (if (NLISTP x)
        then TRUE
      elseif x:Operator=IFOP
        then <IFOP x:Test (LegalityChecks x:ThenPart)
                   (LegalityChecks x:ElsePart) >
      else (Checks x 'OkToCall])
```

61

```
(Macro
  [LAMBDA (x)                                          (* D.Thompson "19-Aug-80 11:14")
    (PROG (y (AskUserToDefine NIL))
          (y←(Translate x))
          (if y:Operator='ExpressionWithType
              then (Heading " *** Already defined to be:")
                   (PrettyPrint y:Expression:LHS)
            elseif y:RHS:Operator~='ExpressionWithType
              then (Heading " **** Right hand side is not legal")
                   (PrettyPrint y:RHS)
            else y:RHS←(InsertLISTs y:RHS)
                 (if (LISTP y:LHS)
                     then (y:LHS:Operator←(MakeExtension (MakeExtension (Shorten y:LHS:Operator)
                                                                        (Name CurrentType))
                                                         'Interface)))
                 (CompileRuleIntoLisp y 'macro)
                 (if (LISTP y:LHS)
                     then (NoteInterfaces <y:LHS:Operator>])
```

62

```
(NameSubList
  [LAMBDA (Imp)                                        (* Edited by R.Bates on 7-FEB-78;
                                                       from version 54)
    (for n in (Functions (AssociatedType Imp)) collect (create SubPair
                                                           OLD ← n
                                                           NEW ←(ExtendName n Imp])
```

63

```
(Quote
  [NLAMBDA (a1% )                                      (* Edited by R.Bates on 24-JAN-78;
                                                       from version 43)
    (if (NLISTP a1% )
        then <'Quote a1% >
      else <'Quote <a1% :1 !(for a%  in a1% ::1 collect (EVAL a% ))
                   >>])
```

64

```
(SubstTypes
  [LAMBDA (NewType OldType x)                          (* Edited by R.Bates on 24-JAN-78;
                                                       from version 43)
    (if (NLISTP x)
        then (if OldType=x
                 then NewType
               elseif (Extension x)=OldType
                 then (ExtendName (RenameAtoms x)
                                  NewType)
               else x)
      else (for y in x collect (SubstTypes NewType OldType y])
```

65

```
(Using
  [NLAMBDA (Exp%  Imp% )                               (* R.Bates "14-Dec-79 14:43")
    (if (NLISTP Exp% )
        then (if (FASSOC Exp%  Imp% ):NEW
                 else Exp% )
      else (MakeFunction <(if (FASSOC Exp% :Operator Imp% ):NEW
                              else Exp% :Operator)
                          !(for x in Exp% :Arguments collect (APPLY* 'Using x Imp% ))
                          >])
```

66

```
(declareMacro
  [LAMBDA (File Stop)                                  (* R.Bates "10-Jan-80 15:51")
    (PROG (x)
          (x←(parse 'expressionSeq File Stop))
          (if x
              then (for z in x:expression do (Macro z))
```

```
                (RETURN x:expression])
)

(RPAQQ SavedFnProps (PrimaryLHSides EqualOp EQOP))

(RPAQQ KnownTypes (Basis TypeParameter Boolean BUILTIN Integer Induction))

(ADDTOVAR NOSAVESETVARS KnownTypes)

(PUTPROPS InitializeLoad INFO EVAL)
(DECLARE: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS

(ADDTOVAR NLAMA )

(ADDTOVAR NLAML Using Quote InitializeLoad)

(ADDTOVAR LAMA cases\Schema cases\Interface GenericInterface)
)
(DECLARE: DONTCOPY
  (FILEMAP (NIL (1144 24777 (Axiom 1156 . 1841) (Axioms 1845 . 1948) (Define 1952 . 3220) (Distinct 3224
 . 5972) (Edit 5976 . 6421) (NoChange 6425 . 8402) (Schema 8406 . 8908) (adopt 8912 . 9531) (
declareVar 9535 . 13124) (discard 13128 . 17564) (discardInterface 17568 . 18303) (discardVariable
18307 . 19240) (distinct 19244 . 21724) (nochange 21728 . 24411) (set 24415 . 24774)) (24943 28309 (
Constants 24955 . 25115) (Constructors 25119 . 26071) (Functions 26075 . 26351) (IsType 26355 . 26611)
 (Name 26615 . 26823) (NonConstructorsUsed 26827 . 27836) (Selectors 27840 . 28306)) (28411 33954 (
DeclareFun 28423 . 30483) (DeclareType 30487 . 32552) (GenerateReflexiveRule 32556 . 33132) (
MakeConstant 33136 . 33415) (MinimalTypeSpec 33419 . 33951)) (34222 41518 (InterfaceDefined 34234 .
34468) (NoFreeVarsInRule 34472 . 35302) (PexecLists 35306 . 35906) (PrintDecls 35910 . 36702) (
RangeType 36706 . 37097) (RemoteTypeOf 37101 . 37651) (Translate 37655 . 38515) (TranslateLitAtom
38519 . 39773) (TypeOfExpression 39777 . 40735) (pexec 40739 . 41268) (typeify 41272 . 41515)) (41585
43648 (AddDeclarations 41597 . 42310) (MakeInterfaceRule 42314 . 43645)) (43796 50314 (
ExpressionWithType 43808 . 43905) (GenericInterface 43909 . 46662) (InsertExtensions 46666 . 46940) (
InterfaceError 46944 . 49926) (cases\Interface 49930 . 50195) (cases\Schema 50199 . 50311)) (50465
54784 (InitializeLoad 50477 . 52083) (NoteDeclarations 52087 . 52275) (NoteInterfaces 52279 . 52561) (
SetupFile 52565 . 54781)) (54883 57729 (AtomsOfType 54895 . 55304) (Infix 55308 . 56553) (RenameAtoms
56557 . 57160) (RetrieveRules 57164 . 57248) (ShortenAllAtoms 57252 . 57568) (initInfix 57572 . 57726)
) (57878 61308 (AssociatedType 57890 . 57952) (InsertInterfaces 57956 . 58086) (InsertLISTs 58090 .
58393) (LegalityChecks 58397 . 58760) (Macro 58764 . 59562) (NameSubList 59566 . 59873) (Quote 59877 .
 60170) (SubstTypes 60174 . 60604) (Using 60608 . 61006) (declareMacro 61010 . 61305)))))
STOP
```