<AFFIRM>REWRITERULE..17 30-Sep-81 15:32:31

ApplicableRuleType	7
ApplyRules	16
ArgumentVars	8
CompileRuleIntoLisp	1
computeLispCode	25
ConvertOldType	2
EvalRules	17
EvaluateRule	9
GeneralLHS	10
GenId	18
GetLHSide	19
LeftHandSides	11
MakeFunction	3
MakeSubs	20
Matcher	21
NoteLeftHandSides	4
PatternSub	22
PatternSubs	23
PatternTests	24
RemoveRule	5
RenameArgs	12
Rules	13
RulesThatMightInteract	14
SimplifyRule	15
ZapRule	6

```
(FILECREATED "26-Sep-81 17:20:18" <AFFIRM>REWRITERULE..17 27683

     changes to:  NoteLeftHandSides Rules RulesThatMightInteract REWRITERULECOMS RRuleQueryFns
ApplicableRuleType ArgumentVars GeneralLHS LeftHandSides RenameArgs RRuleModFns RRuleOtherFns

     previous date: "31-Mar-81 19:38:45" <AFFIRM>REWRITERULE..16)


(PRETTYCOMPRINT REWRITERULECOMS)

(RPAQQ REWRITERULECOMS ((* main modification fns. Note that these divisions are rough)
                        (FNS * RRuleModFns)
                        (* main queries)
                        (FNS * RRuleQueryFns)
                        (* other)
                        (FNS * RRuleOtherFns)))
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

(* main modification fns. Note that these divisions are rough)]

```
(RPAQQ RRuleModFns (CompileRuleIntoLisp ConvertOldType MakeFunction NoteLeftHandSides RemoveRule
                                        ZapRule))
(DEFINEQ
```

1

(CompileRuleIntoLisp

```
  [LAMBDA (rule kind dontInteract)                    (* R.Erickson "17-Mar-81 19:41")
```

(* rule is an equation which is put in Lisp code. kind is one of RuleTypes, and is placed in the call to Report, which regulates when it may be applied. Normally stores all the property interactions, but dontInteract suppresses OccursLHSides.)*

(* See RulesThatMightInteract for property format description)*

```
    (if (NLISTP rule:LHS)
        then (printout NIL T "incorrect left hand side: ")
             (PrettyPrint rule T)
             (AffirmError NIL NIL T)
      else (PROG (name id code defn primProp)
                 (name←rule:LHS:Operator)
                 (if (FGETD name)=NIL
                     then (AffirmError '(MakeFunction was called from CompileRuleIntoLisp. Please
                                                      send transcript to the PV group. Everything is
                                                      still ok.)
                                       'mild)
                          (MakeFunction rule:LHS))
                 (id←(GenId name))                    (* rule #)
                 (primProp←(GETPROP name 'PrimaryLHSides))
                                                      (* so we can access and smash)
                 (/RPLACD primProp < ! primProp::1 (create LhsEntry
                                                           eid ← id
                                                           elhs ← rule:LHS)
                                   >)
                 (if dontInteract
                   else (for op in (SecondaryOperators rule:LHS)
                           do (/ADDPROP op 'OccursLHSides rule:LHS T)))
                 (code←(computeLispCode rule id kind))
                 (if (EXPRP name) and (CAADDR (defn←(FGETD name)))='COND
                     then                             (* bypass the editor for speed;
                                                      mark as changed, not undoable for safety)
                          (pushnew CHANGEDFNSLST name)
                          (/RPLACA defn::2 <'COND code ! defn:3::1>)
                   else (CallEDITF name <<'ORR <'(F COND T)
                                               <-2 code>> <'(F if T)
                                                          <1 'if code:1 'then code:2 'elseif >>
                                           <-1 <'MBD <'COND code <'* >>>>>>])
```

2

(ConvertOldType

```
  [LAMBDA (type)                                      (* D.Thompson "19-Aug-80 10:39")
```

(* Given the type name, convert the lhs properties to the new (post Affirm 35) format. Change the fileCOMS if appropriate. Better be sure this happens before NoteLeftHandSides is called.)*

```
    (PROG (fns ifns coms hit?)
          (if type:DeclaredType
            else (AffirmError <"Not a type:" type>))
          (fns←(PACK* type 'FNS))
          (ifns←(ExtendName type 'InterfaceFNS))
          (coms←(PACK* (U-CASE type)
                       'COMS))
          (for fn in < !(EVAL fns) !(EVAL ifns) > bind compl lid
             do                                       (* convert LastId, LeftHandSides, n to OccursLHSides,
                                                      PrimaryLHSides.)
                (compl←(GETPROP fn 'LeftHandSides))   (* nonNIL if this should be interacted)
                (lid←(GETPROP fn 'LastId))
                (if lid
                    then hit?←T
                         (/PUTPROP fn 'PrimaryLHSides
                                   <lid !(for n to lid bind lhs
```

```
                                        collect (lhs←(GETPROP fn n))
                                                (if compl
                                                    then (for op in (SecondaryOperators lhs)
                                                              do (/ADDPROP op 'OccursLHSides lhs T))
                                                    )
                                                (/PUTPROP fn n NIL)
                                                (create LhsEntry
                                                        eid ← n
                                                        elhs ← lhs))
                                    >))
            (/PUTPROP fn 'LastId NIL)
            (/PUTPROP fn 'LeftHandSides NIL))
```

(* change fileCOMS so only desired properties are kept.)*

```
(if (EVALV coms)
    ~='NOBIND
    then (APPLY* 'EDITV coms <'LPQ <'F <'IFPROP 'ALL '& <'*ANY* fns ifns>> 'N >
                                       <2 SavedFnProps>>))
(if hit?
    then (printout NIL T "The type" , type , "has been converted to a new internal form." T
                 "You may wish to save it."
                 T])
```

3

(MakeFunction

```
[LAMBDA (x)                                                    (* R.Erickson "17-Mar-81 19:42")
```

(* Sees to it that x:Operator is defined. If not, uses x:Arguments to determine arity and preferred arg names. Does not deal with LHS, etc.)*

```
(PROG (Args Ex)
      [Args←(for y in x:Arguments collect (if (LITATOM y)
                                              then (Shorten y)
                                             else (GENSYM]
      (if ~BootStrapping or ~(GETD x:Operator)
          then (/PUTD x:Operator <'LAMBDA Args <'COND <<'LIST (KWOTE x:Operator) ! Args>>>>>))
      (Ex←(Extension x:Operator))
      [if Ex
          then (if (IsType Ex)
                   then (if (EVALV x:Operator)=x:Operator
                            then SkolemFunctions← <x:Operator ! SkolemFunctions>
                           else (push CHANGEDFNSLST x:Operator)
                                                    (* notice fn change)
                                (AddToFile x:Operator Ex 'FNS]
      (RETURN x])
```

4

(NoteLeftHandSides

```
[LAMBDA (fns)                                                  (* R.Erickson "25-Sep-81 20:08")
```

(fns should be a list of names of functions of a data type. For each f in fns and each rewrite rule left hand side hung on f and each operator therein, the OccursLHSides property of that operator is updated if necessary.)*

```
(for f in fns do (for lhs in (LeftHandSides f) do (for op in (SecondaryOperators lhs)
                                                       when ~(MEMBER lhs (GETPROP op 'OccursLHSides)
                                                                     )
                                                       do (/ADDPROP op 'OccursLHSides lhs T])
```

5

(RemoveRule

```
[LAMBDA (rule dontPrint)                                       (* D.Thompson "13-Nov-80 13:20")
```

(* Removes rule from RuleList or lisp code. For lisp code uses only lhs for symbolic match. Returns the type of the exrule.)*

```
(PROG (lhs rlEntry name)
      (if dontPrint
        else (printout NIL (if (POSITION)=0
                               then "D"
                              else " d")
```

```
                    "iscarding rule" # (listOneRule rule:LHS RewriteRuleOp rule:RHS)))
        (if rlEntry←(SASSOC rule RuleList)
            then RuleList←(REMOVE rlEntry RuleList)
                 (RETURN RuleTypes:Axiom)
          else name←rule:LHS:Operator
               lhs←rule:LHS
               Unchecked←(for x in Unchecked unless (EQUAL x::1:LHS lhs) collect x)
               (for pair in (GETPROP name 'PrimaryLHSides)::1 until (EQUAL lhs pair:elhs)
                  do NIL finally (ZapRule name pair:eid)
                                 (RuleHistory← <<'discard ZapKind rule> ! RuleHistory>))
               (RETURN ZapKind])
```

6

```
(ZapRule
  [LAMBDA (fname id)                                    (* R.Bates "23-Jan-80 12:58")

          (* * deletes the compiled rule for fname. Sets global ZapKind to the type of the rule. If id = T, we assume there is
          only one candidate; if NIL, there is one or none.)


    (PROG (pair lhs primaryProp ncands)
          (primaryProp←(GETPROP fname 'PrimaryLHSides))
          (if id and id~=T
              then pair←(FASSOC id primaryProp::1)
                   (if ~pair
                       then (AffirmError <"Rule to delete not found" id fname> 'internal))
            else                                        (* id = T or NIL)
                 ncands←(FLENGTH primaryProp::1)
                 (if ncands gt 1
                     then (AffirmError <"Rule id to delete is not unique for" fname> 'internal)
                   elseif ncands=0
                     then (if id
                              then (AffirmError <"No rule to delete for" fname> 'internal)
                            else                        (* optional)
                                 (RETURN)))
                 pair←primaryProp:2
                 id←pair:eid)
          (/RPLACD primaryProp (/DREMOVE pair primaryProp::1))
                                                        (* delete from PrimaryLHSides)
          (lhs←pair:elhs)
          [for op in (SecondaryOperators lhs) do        (* Remove all cross-references to this rule)
                                              (/PUTPROP op 'OccursLHSides
                                                        (REMOVE lhs (GETPROP op 'OccursLHSides]
          (CallEDITF fname
                     <<'LPQ 'F 'if '↑ 'DW >
                       <'ORF <'AND '-- <'Report fname id '-- >> <'Report fname id '-- >> '(S
                         ZapKind
                         (F Report T)
                         4)
                       0 '(:)
                       >])
)
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

(* main queries)]

```
(RPAQQ RRuleQueryFns (ApplicableRuleType ArgumentVars EvaluateRule GeneralLHS LeftHandSides
                                         RenameArgs Rules RulesThatMightInteract SimplifyRule))
(DEFINEQ
```

7

(ApplicableRuleType

```
  [LAMBDA (lhs)                                    (* R.Erickson "26-Feb-81 19:03")
                                                   (* lhs:Operator is extended)

          (* * returns ruletype which can fire on lhs. If several are applicable, just get the first! Failure return NIL)

    (RESETVARS (KindOfRule (ListingFlag T)
                           (REPORTFLAG NIL)
                           (UseRules T))
               (APPLY lhs:Operator lhs:Arguments)
               (RETURN KindOfRule])
```

8

(ArgumentVars

```
  [LAMBDA (typedArgs fnName avoid)                 (* R.Erickson "21-Sep-81 18:16")

          (* given a list of type names and/or EWT records, :Expressions of which are unique extended atoms.
          All variables should be long. fnName is used for error messages.)


          (* * Our caller wants to generate type-correct variable arguments to some function. In some arg positions, we are
          given an ExpressionWithType record; this has the variable the user supplied. which is given priority.
          Otherwise, we are given atomic :Type, and generate a name. All of this is in context of the current type.
          We assume that caller has ensured that none of the EWT's :Expressions conflict; given that, we return a unique list
          of vars.)


    (PROG (taken decls vars)
          (taken←(for ewt in typedArgs when (type? ExpressionWithType ewt) collect ewt:Expression))
                                                   (* variables in use so far)
          (pushlist taken avoid)
          (decls←(getDeclarations CurrentType T))  (* an ALIST from type to declared vars)
          [vars←(for arg in typedArgs bind cands v
                   collect

          (* * the proper var for this position)


                                (if (type? ExpressionWithType arg)
                                    then                    (* ok; use what's given)
                                         arg:Expression
                                  else                      (* given type; pick a name)
                                       cands←(FASSOC arg decls):NEW
                                                            (* list of possible vars)
                                       (if v←(LDIFFERENCE cands taken):1
                                           then             (* some unused; pick first)
                                                (push taken v)
                                                v
                                         elseif cands
                                           then

          (* All taken, but there are vars; rename with primes. Of the declared variables, cands, we pick that one which gives
          the fewest primes. That way, we don't just pile primes onto the first declared var.)


                                                v←(Minimize (for c in cands collect (NewSymbolFrom c taken)
                                                                                    )
                                                            'NPrimes)
                                                (push taken v)
                                                v
                                           else             (* none declared of proper type)
                                                (AffirmError <"Cannot produce a left-hand side for" fnName
                                                              " because no variables of type"
                                                              arg:Type "are declared in current type." >]

          (RETURN vars])
```

9

```
(EvaluateRule
  [LAMBDA (rule)                                        (* R.Erickson "24-Jun-80 13:29")

          (* * Used to simplify a rule, using everything in the system. Besides checking for RuleList, is careful not to just
          EVAL the rule, since that would allow equality axioms of the type to bridge the equal sign. Is careful to return the
          right equality operator (at least, the same as was given))


    (if UsingRuleList
        then (EvalRules rule)
      elseif rule:Operator=EQVOP
        then                                            (* allow equality axioms to apply)
             (EVAL rule)
      else (PROG (lh rh eq)
                 (lh←(EVAL rule:LHS))
                 (rh←(EVAL rule:RHS))
                 (eq←(Equal lh rh))
                 (RETURN (if (NLISTP eq)
                             then eq
                           else <rule:Operator lh rh>])
```

10

```
(GeneralLHS
  [LAMBDA (fn avoid)                                    (* R.Erickson "21-Sep-81 18:23")
                                                        (* Fn is an atomic name or expression, extended with
                                                        type. Avoid a list of vars.)

          (* * Computes a list of variables which are suitable as arguments to fn; produces <fn!args>.
          We prefer the vars used in interface, but always return something suitable for the current ytpe.
          This check is because the original interface vars might have been discarded. -
          We may be given an expression, in which case we fill in the slots for vars. -
          We stay away from any vars in avoid -
          Note that functions like Equal don't have normal interfaces, won't work.)


    (PROG (ilhs iargs vars actuals fnArgs)
          (if (LISTP fn)
              then                                      (* given an expression: first compute just vars)
                   actuals←fn:Arguments
                   fn←fn:Operator)
          [ilhs←(CAR (LeftHandSides (ExtendName fn IEXT]    (* Interface will have only one lhs, this is it.
                                                            ilhs:Arguments are ExpressionWithType records.
                                                            Note that :Arguments:-1 is NIL, used for
                                                            TooManyArguments)
          (if ~ilhs
              then (AffirmError <"no interface" fn> 'internal))
          (iargs←(for tail on ilhs:Arguments when tail::1 collect tail:1))
                                                        (* all interface arguments except the last,
                                                        TooManyArgs.)

          (* * Compute the variables to use, giving precedence to interface vars if still good)


          (vars←(for ewt in iargs collect                   (* :Expression is translated var, :Type its type)
                                    (if (EQUAL ewt (TranslateLitAtom ewt:Expression T))
                                        and ~(MEMB ewt:Expression avoid)
                                      then                  (* still declared, same type;
                                                            not in avoid; pass EWT record.
                                                            Note that TLA shortens its args.)
                                           ewt
                                      else                  (* changed; pass type name)
                                           ewt:Type)))
          (vars←(ArgumentVars vars (Shorten fn)
                              avoid))
          (if actuals
              then

          (* * We were given an expression, so we have to recurse and fill in for nonvariable args. (Assuem:
          (EQ (LENGTH vars) (LENGTH actuals))))


                   (pushlist avoid (for var in vars as act in actuals unless act collect var))
                                                        (* add those vars we will really use here.)
                   fnArgs←(for var in vars as act in actuals bind general
                             collect                    (* in a var position, use what we just computed.
                                                        Otherwise, recurse, making sure we never use the same
                                                        variable twice.)
```

```
                                (if act
                                    then                    (* expr)
                                         general←(GeneralLHS act avoid)
                                         (pushlist avoid (Variables general))
                                         general
                                  else                      (* var)
                                       var))
                        (RETURN <fn ! fnArgs>)
                  else (RETURN <fn ! vars>))                (* unique variable names. suitable as args)
    ])
```

11

```
(LeftHandSides
  [LAMBDA (fn)                                              (* R.Erickson "17-Sep-81 19:56")
                                                            (* fn is atomic extended)

        (* * Returns a list of left-hand sides, without regard for the rule type. Old functionality, newly made a function.)

    (for pair in (GETPROP fn 'PrimaryLHSides)::1 collect pair:elhs])
```

12

```
(RenameArgs
  [LAMBDA (call avoid)                                      (* R.Erickson "24-Aug-81 15:03")
                                                            (* call is of form: "(fn var1 ... varn)" and avoid is a
                                                            list of variables, all long.)

        (* * Rename the variables in call so they aren't in avoid)

    <call:Operator ![for arg in call:Arguments collect      (* the variable, renamed and added to avoid)
                                                          (CAR (push avoid (NewSymbolFrom arg avoid]
    >])
```

13

```
(Rules
  [LAMBDA (Name Kinds)                                      (* R.Erickson "25-Sep-81 20:10")

        (* * Returns all rules with the primary operator Name, such that the rule's kind is in Kinds, or Kinds = T.
        Name may be a list.)

    (if Kinds~=T and (NLISTP Kinds)
        then Kinds← <Kinds>)
    (if (NLISTP Name)
        then (for lhs in (LeftHandSides Name) bind rule eachtime rule←(Rule lhs)
                when Kinds=T or KindOfRule MEMB Kinds collect rule)
      else (for n in Name join (Rules n Kinds])
```

14

```
(RulesThatMightInteract
  [LAMBDA (rule ignoreThese)                                (* R.Erickson " 3-Aug-81 16:44")

        (* * Returns all rules which include rule:LHS:Operator in their LHS, and all rules whose primary operator is in
        rule:LHS. These come from compiled rules' properties, and from RuleList. Any rule will be ignored if it is EQUAL to
        one of ignoreThese.)


        (* * Format descriptions: -
        RuleList is a list of entries. In each one, CDR lists all operators present, and CAR is the rule with an EqualOp.
        -
        LHS properties. -
        PrimaryLHSides: CAR is numeric LastId (we may later extend this to allow (CONS # filterfunction)) CDR is an alist
        from id to rule lhs -
        OccursLHSides: a list of the lhs of all rules in which this operator occurs but is NOT primary.)


    (PROG (secondary primary Ops (mainOp (rule:LHS:Operator)))

        (* * all rules in which mainOp is found as a secondary)
```

```
        (secondary← < !!(for pair in RuleList when mainOp MEMB pair::1 collect pair:1)
                      !(for lhs in (GETPROP mainOp 'OccursLHSides) bind rule eachtime rule←(Rule
                                                                                          lhs)
                          unless KindOfRule=RuleTypes:Define collect rule)
                     >)
        (Ops←(Operators rule:LHS))

        (* * all rules which have a primary operator in Ops -
        including self)


        (primary←(Rules Ops <RuleTypes:Axiom RuleTypes:Lemma ! RuleTypes:OtherRules>))
        (if (UserProfile 'KBAxiomOrder)='Inverse
            then primary←(REVERSE primary))
        (primary← < !!(for pair in RuleList bind rule eachtime rule←pair:1
                          when rule:LHS:Operator MEMB Ops collect rule)
                    ! primary>)
        (RETURN (REMOVEDUPLICATES (for rule in < ! secondary ! primary>
                                     unless (MEMBER rule ignoreThese) collect rule])
```

15

```
(SimplifyRule
  [LAMBDA (rule)                                    (* R.Erickson "20-Jun-80 14:20")
                                                    (* Reduce both sides of rule using all existing rules
                                                    except rule itself)
    (PROG (rlEntry)
          (RETURN (if rlEntry←(SASSOC rule RuleList)
                      then (RESETVARS ((RuleList (REMOVE rlEntry RuleList)))
                                      (RETURN (EvalRules rule)))
                    else (RESETVARS ((ListingFlag ('IgnoreRule))
                                     (RuleToBeIgnored rule)
                                     (REPORTFLAG NIL)
                                     (UseRules T))
                                    (RETURN (EvaluateRule rule])
)
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

```
(* other) ]

(RPAQQ RRuleOtherFns (ApplyRules EvalRules GenId GetLHSide MakeSubs Matcher PatternSub PatternSubs
                          PatternTests computeLispCode))
(DEFINEQ

                                                                                    16

(ApplyRules
  [LAMBDA (op args)                                          (* R.Erickson "12-Dec-79 12:34")
    (RESETVARS (rule MatchBindings)
               (RETURN (if (for pair in RuleList thereis (rule←pair:1)
                                                         (if op=rule:LHS:Operator
                                                             then MatchBindings←NIL
                                                                  (Matcher rule:LHS <op ! args>)))
                           then (EvalRules (SUBLIS MatchBindings rule:RHS))
                           else (PROG (ex)
                                      (ex←(APPLY op args))
                                      (RETURN (if ex:Operator=op and (EQUAL ex:Arguments args)
                                                  then ex
                                                  else (EvalRules ex])

                                                                                    17

(EvalRules
  [LAMBDA (ex)
    (if (NLISTP ex)
        then ex
      else (if ex:Operator=QOP
               then (create Qexpression
                            expr ←(EvalRules ex:expr) using ex)
             else (ApplyRules ex:Operator (for arg in ex:Arguments collect (EvalRules arg])

                                                                                    18

(GenId
  [LAMBDA (Name)                                             (* R.Erickson "20-Dec-79 17:47")
                                                             (* Returns the next rule # for Name, updates counter.)
    (PROG (prop)
          (RETURN (if prop←(GETPROP Name 'PrimaryLHSides)
                      then (/RPLACA prop prop:1+1):1
                   else (/PUTPROP Name 'PrimaryLHSides <1>)
                        1])

                                                                                    19

(GetLHSide
  [LAMBDA (name id)                                          (* R.Bates "23-Jan-80 12:58")

          (* * Returns the left-hand side with main operator Name and the proper rule #)

    (OR (fetch elhs of (FASSOC id (GETPROP name 'PrimaryLHSides)::1))
        (AffirmError <"Rule not found for id" id name> 'internal])

                                                                                    20

(MakeSubs                                                    funname
  [LAMBDA (PS x)                                             (* R.Erickson "27-OCT-78 14:41")
    (if (NUMBERP x)
        then x
      else (PROG (y)
                 (y←(SASSOC x PS))
                 (RETURN (if y
                             then y:NEW
                          elseif (NLISTP x)
                            then x
                            else                             (* changed to do operators too)
                                 (for z in x collect (MakeSubs PS z])

(~~for~~ < x:1 ~~(KWOTE x:1)~~ ! (for z in x::1 collect (MakeSubs PS z ))>>
                                                                                    21

(Matcher                                                     funname
  [LAMBDA (x y)                                              (* R.Erickson "20-Feb-80 12:33")

(if x:1 = funname then funname else (KWOTE x:1))
```

```
        (* * Seeks theta suchthat theta (x) = y. If successful, returns true. Global MatchBindings should be bound by caller
        to NIL, will receive sublist theta. If a function has different arities in x and y, the extra args are simply
        ignored.)


    (if (LITATOM x) and ~(x : IsConstant)
        then                                                    (* is atom x already assigned?)
             (if (FASSOC x MatchBindings)
                 then (EQUAL (FASSOC x MatchBindings)::1 y)
                 else MatchBindings← <<x ! y> ! MatchBindings>)
      elseif (NLISTP x)
        then                                                    (* numeric, etc)
             (EQUAL x y)
      elseif x:Operator=y:Operator
        then (for z in x:Arguments as w in y:Arguments always (Matcher z w])
```

22

```
(PatternSub
  [LAMBDA (x y)                                                 (* Edited by R.Bates on 24-JAN-78;
                                                                from version 16)
    <(create SubPair
             OLD ← x
             NEW ← y)
     !(if (NLISTP x)
          then NIL
          else (for z in x:Arguments bind w first w←y eachtime w← <'CDR w>
                  join (PatternSub z <'CAR w>)))
    >])
```

23

```
(PatternSubs
  [LAMBDA (Patterns Arguments)                                  (* R.Erickson " 6-Dec-79 19:34")

        (* * Given Patterns, the LHS of a rule, and Arguments, the arglist of the operator. Returns an alist, with an :OLD
        entry for each subexpression of each Pattern. :NEW tells how to get to :OLD in terms of the Arguments.
        For example, "<f(i,g(j)).(a b)>" would yield "((i . a)((g j). b)(j . (CADR b))" (except using CAR, CDR))


    (for x in Patterns as y in Arguments join (PatternSub x y])
```

24

```
(PatternTests
  [LAMBDA (Patterns PS)                                         (* Edited by R.Bates on 24-JAN-78;
                                                                from version 16)
    (for x in Patterns join (if (NLISTP x)
                                then (if (NUMBERP x) or (STRINGP x) or x:IsConstant or x=NIL
                                         then <<'EQ (SASSOC x PS):NEW x>>
                                         else NIL)
                                else <<'EQ <'CAR (SASSOC x PS):NEW> <'QUOTE x:Operator>>
                                                                !(PatternTests x:Arguments PS)
                                                                >])
```

25

```
(computeLispCode
  [LAMBDA (rule id kind)                                        (* R.Bates "23-Jan-80 12:58")

        (* * Called by CompileRuleIntoLisp, this returns a segment, suitable for insertion into a COND, which implements the
        rule.)


    (PROG (name formalargs report ps test eqtest result)
          (name←rule:LHS:Operator)
          (formalargs←(ARGLIST name))
          (report← <'Report name id kind>)
          (if (FLENGTH rule:LHS:Arguments) gt (FLENGTH formalargs)
              then (AffirmError <"Cannot create a rule where" name "has" (FLENGTH rule:LHS:Arguments)
                                "arguments, because it already has a rule with"
                                (FLENGTH formalargs)
                                >))
          (ps←(PatternSubs rule:LHS:Arguments formalargs))
                                                                (* pieces of LHS and how to get them)
          (eqtest←(for patts on ps bind patt other eachtime (patt←patts:1)
```

```
                                                    (other←(if (NLISTP patt:OLD)
                                                               or patt:OLD:Arguments=NIL
                                                              then (SASSOC patt:OLD patts::1)
                                                              ))
                        when other collect                    (* test for equality of any duplicated elements which
                                                              are vars or nilary fns)
                                          (<'EQUAL patt:NEW other:NEW>)))
                (ps←(for x on ps unless (SASSOC x:1:OLD x::1) collect x:1))
                                                              (* only unduplicated pieces)
                (test← < !!(PatternTests rule:LHS:Arguments ps) !! eqtest ! <report>>)
                (test←(if test::1=NIL
                          then test:1
                         else <'AND ! test>))
                (result←(MakeSubs ps rule:RHS))
                (result←(UCI\SUBST <'LIST (KWOTE name) ! formalargs> <name ! formalargs> result))
                                                              (* remove any self-evaluation with same arguments)
                (RETURN <test result>])
)
(DECLARE: DONTCOPY
  (FILEMAP (NIL (810 9093 (CompileRuleIntoLisp 822 . 2869) (ConvertOldType 2873 . 4747) (MakeFunction
4751 . 5770) (NoteLeftHandSides 5774 . 6403) (RemoveRule 6407 . 7417) (ZapRule 7421 . 9090)) (9319
20904 (ApplicableRuleType 9331 . 9842) (ArgumentVars 9846 . 12386) (EvaluateRule 12390 . 13275) (
GeneralLHS 13279 . 16681) (LeftHandSides 16685 . 17112) (RenameArgs 17116 . 17668) (Rules 17672 .
18274) (RulesThatMightInteract 18278 . 20255) (SimplifyRule 20259 . 20901)) (21105 27659 (ApplyRules
21117 . 21727) (EvalRules 21731 . 22044) (GenId 22048 . 22461) (GetLHSide 22465 . 22814) (MakeSubs
22818 . 23262) (Matcher 23266 . 24213) (PatternSub 24217 . 24607) (PatternSubs 24611 . 25175) (
PatternTests 25179 . 25681) (computeLispCode 25685 . 27656)))))
STOP
```

(FILECREATED "29-Jun-81 16:55:40" <AFFIRM>RUNGOP..3 1418

previous date: "12-Sep-80 17:16:48" <AFFIRM>RUNGOP..2)

```
(PRETTYCOMPRINT RUNGOPCOMS)

(RPAQQ RUNGOPCOMS ((FNS * RUNGOPFNS)
                   (ADVICE * RUNGOPADVICE)))

(RPAQQ RUNGOPFNS (RUNGOP))
(DEFINEQ

(RUNGOP
  [LAMBDA (InputFile)                                    (* Edited by R.Bates on 31-OCT-78;
                                                            from version 3)
    (COND
      ((PRIN1 (GOPFILE (QUOTE PARSER)
                       InputFile
                       (QUOTE STOP)))
        (PRIN1 ".")
        (LOAD (QUOTE <AFFIRM>PARSERPLUS))
        [EDITE PARSERCOMS (QUOTE (F FNS (E (PROGN (SETQ PARSERFNS (COPY (## 2 UP)))
                                                  NIL))
                                       (DELETE (2 THRU -1))
                                       (N * PARSERFNS)
                                       ↑
                                       (LP I (P (LOAD --))
                                           P UP (1))
                                       ↑ F BLOCKS 2 F ENTRIES UP (BI 1 -1)
                                       !0 2 UP (DELETE (1 THRU -2))
                                       (BO -1)
                                       (COMS (CONS -1 (APPEND PARSERFNS PARSERPLUSFNS)))
                                       !0 F ENTRIES (N PARSEPROGRAM PARSE\ASSERTION]
        (PRIN1 ".")
        (SETQ PARSERCOMS (APPEND PARSERCOMS PARSERPLUSCOMS))
        [EDITE PARSERFNS (QUOTE ((E (PROGN (SORT (## 2 UP))
                                           NIL]
        (PRIN1 PARSERFNS)
        (REMPROP (QUOTE PARSER)
                 (QUOTE FILE))
        (MAKEFILE (QUOTE PARSER)
                  (QUOTE NEW])
)

(RPAQQ RUNGOPADVICE (ADDTOFILE))

(PUTPROPS ADDTOFILE READVICE [NIL (BEFORE NIL (PROGN (SETQ TYPE FILE)
                                                     (SETQ FILE (QUOTE PARSER])
(DECLARE: DONTCOPY
  (FILEMAP (NIL (271 1238 (RUNGOP 283 . 1235)))))
STOP
```