(FILECREATED " 7-Oct-88 10:25:18" <AFFIRM>CEVAL..25 51964

changes to: Equal

previous date: "22-Sep-88 19:16:29" <AFFIRM>CEVAL..24)

(PRETTYCOMPRINT CEVALCOMS)

```
(RPAQQ CEVALCOMS [(* ↑ Ceval ("Conditional Evaluator")
                     uses if-then-else expressions to represent all propositional calculus operators
                     (not, and, or , implies, equiv)
                     %. Ceval implements only a small set of transformations on conditional expressions,
                     similar to those discussed by McCarthy and implemented previously in the Boyer-Moore
                     Theorem Prover. However, it is still "complete" with respect to propositional calculus
                     in that any valid formula in the propositional calculus (i.e., provable by truth table)
                     will be reduced to TRUE by the transformations. Ceval also incorporates a
                     "case analysis"
                     rule and basic rules for equality.)
                   (* Variables, properties and functions for interfacing with rest of XIVUS system.)
                   (PROP IsConstant TRUE FALSE)
                   (PROP OriginalName IfThenElse Equal)
                   (FNS * CevalInterfaceFNS)
                   (* (BLOCKS (CevalPropCalcBLOCK AssumedAndDenied IfThenElse IfThenElse0 IfThenElse1
                                                  IfThenElse2 SpecialIf addFact addIntegerFacts falseBranch
                                                  trueBranch (ENTRIES IfThenElse IfThenElse0 SpecialIf)
                                                  (GLOBALVARS ContinuousEval)
                                                  (SPECVARS p q r)
                                                  (NOLINKFNS . T))
                              (CevalEqualityBLOCK TestSubst UseEqHypsInPriorHyps UseEqualities OccursIn
                                                  Hypotheses ApplyEqInPriorHyps
                                                  (ENTRIES TestSubst UseEqHypsInPriorHyps UseEqualities
                                                           OccursIn)
                                                  (NOLINKFNS . T)
                   (* ↑ Functions which implement the transformations on propositional calculus expressions.)
                   (RECORDS AssumedAndDenied)
                   (FNS * CevalPropCalcFNS)
                   (* ↑ Functions which implement the "case analysis" rule, f (a1,..., (if p then x else y)
                                                                                        ,...,an)
                      =
                      (if p then f (a1,...,x,...,an)
                          else f (a1,...,y,...,an))
                      when f is not if-then-else.)
                   (FNS * CevalCaseAnalFNS)
                   (* ↑ Functions which implement equality rules.)
                   (FNS * CevalEqualityFNS)
                   (RECORDS * SEPTHEORYRECORDS)
                   (FNS * septheory)
                   (DECLARE: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS
                             (ADDVARS (NLAMA)
                                      (NLAML IfThenElse SpecialIf)
                                      (LAMA])
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

```
(* ↑ Ceval ("Conditional Evaluator")
    uses if-then-else expressions to represent all propositional calculus operators
    (not, and, or , implies, equiv)
    %. Ceval implements only a small set of transformations on conditional expressions, similar to those
    discussed by McCarthy and implemented previously in the Boyer-Moore Theorem Prover. However, it i
    "complete"
    with respect to propositional calculus in that any valid formula in the propositional calculus
    (i.e., provable by truth table)
    will be reduced to TRUE by the transformations. Ceval also incorporates a "case analysis" rule and bas
    rules for equality.) )

(DECLARE: DONTEVAL@LOAD DONTCOPY
```

(* Variables, properties and functions for interfacing with rest of XIVUS system.)]

```
(PUTPROPS TRUE IsConstant T)

(PUTPROPS FALSE IsConstant T)

(PUTPROPS IfThenElse OriginalName IFELSE)

(PUTPROPS Equal OriginalName EQ)

(RPAQQ CevalInterfaceFNS (AssociativeAndMatch MakeAnd MakeEqv MakeImplies MakeNot RemoveCommonParts MakeOr
                                              RemoveIfs RemoveIfs1 FindEqvs RemoveIfsCommonSubExps
                                              RemoveIfsFromIf RemoveIfsHelper SpecialIf))
(RPAQQ INFO
```

1

```
(AssociativeAndMatch
  (LAMBDA (x y)                                          (* D. Thompson "12-Feb-80 10:47")

        (* * This routine ... -
        Just what DOES this routine do?? I don't have the slightest idea. (DHT))


    (if y:Operator=ANDOP
        then x:Operator=ANDOP and (EQUAL x:Arg2 y:Arg2) and (AssociativeAndMatch x:Arg1 y:Arg1)
      else (if x:Operator=ANDOP
              then (EQUAL x:Arg2 y) and x:Arg1
            else (if (EQUAL x y)
                     then TRUE)))
```

2

```
(MakeAnd
  (LAMBDA (x y)                                          (* D. Thompson "28-Jul-80 13:58")

        (* * This routine creates AND expressions, keeping left associativity.)


    (if x=TRUE
        then y
      elseif y=TRUE
        then x
      elseif x=FALSE or y=FALSE
        then FALSE
      elseif y:Operator=ANDOP
        then (create Expression
                     Operator ← ANDOP
                     Arguments ←(<(MakeAnd x y:Arg1)
                       y:Arg2>))
      else (create Expression
                   Operator ← ANDOP
                   Arguments ←(<x y>])
```

3

```
(MakeEqv
  (LAMBDA (x y)                                          (* D. Thompson "28-Jul-80 13:54")

        (* * This routine creates EQV expressions, keeping left associativity.)


    (if x=TRUE
        then y
      elseif y=TRUE
        then x
      elseif x=FALSE
        then (MakeNot y)
```

```
      elseif y=FALSE
        then (MakeNot x)
      elseif y:Operator=EQVOP
        then (create Expression
                     Operator ← EQVOP
                     Arguments ←(<(MakeEqv x y:Arg1)
                       y:Arg2>))
      else (create Expression
                   Operator ← EQVOP
                   Arguments ←(<x y>))
```

4

```
(MakeImplies
  (LAMBDA (x y)                                   (* D. Thompson "28-Jul-80 14:02")

        (* * This routine creates IMP expressions.)


    (if x=TRUE
       then y
     elseif x=FALSE
       then TRUE
     elseif y=TRUE
       then TRUE
     elseif y=FALSE
       then (MakeNot x)
     elseif y:Operator=IMPOP
       then (create Expression
                    Operator ← IMPOP
                    Arguments ←(<(MakeAnd x y:Arg1)
                      y:Arg2>))
     else (create Expression
                  Operator ← IMPOP
                  Arguments ←(<x y>))
```

5

```
(MakeNot
  (LAMBDA (expression)                            (* D. Thompson "28-Jul-80 14:11")
    (PROG (extension operator)
          (operator←expression:Operator)
          (RETURN (if (NLISTP expression)
                      then (if expression=TRUE
                               then FALSE
                             elseif expression=FALSE
                               then TRUE
                               else (create Expression
                                            Operator ← NOTOP
                                            Arguments ←(<expression>)))
                    elseif operator=LTOP
                      then (create Expression
                                   Operator ← LEOP
                                   Arguments ←(<expression:Arg2 expression:Arg1>))
                    elseif operator=LEOP
                      then (create Expression
                                   Operator ← LTOP
                                   Arguments ←(<expression:Arg2 expression:Arg1>))
                    elseif operator=GTOP
                      then (create Expression
                                   Operator ← LEOP
                                   Arguments ←(<expression:Arg1 expression:Arg2>))
                    elseif operator=GEOP
                      then (create Expression
                                   Operator ← LTOP
                                   Arguments ←(<expression:Arg1 expression:Arg2>))
                    elseif operator=ANDOP
                      then (MakeOr (MakeNot expression:Arg1)
                                   (MakeNot expression:Arg2))
                    elseif operator=IMPOP
```

```
      then (MakeAnd expression:Arg1 (MakeNot expression:Arg2))
    elseif operator=NOTOP
      then expression:Arg1
    elseif operator=OROP
      then (MakeAnd (MakeNot expression:Arg1)
                    (MakeNot expression:Arg2))
    else (create Expression
                 Operator ← NOTOP
                 Arguments ←(<expression>))
```

6

```
(RemoveCommonParts
  (LAMBDA (expression)                                   (* D. Thompson "17-Sep-80 10:23")

        (* * This routine attempts to find commonality in the THEN and ELSE branches of an IF-expression, and
        simplifies the proposition accordingly.)

    expression←(RemoveIfsCommonSubExps expression)
    (bind done until done do (if (type? IfExpression expression:ThenPart)
                                 and (EQUAL expression:ThenPart:ElsePart expression:ElsePart)
                               then

        (* (If B1 then (If B2 then X else Y) else Y) == (If (B1 and B2) then X else Y))

                                    expression←(create IfExpression
                                                      Test ←(MakeAnd expression:Test
                                                                     expression:ThenPart:Test)
                                                      ThenPart ← expression:ThenPart:ThenPart
                                                      ElsePart ← expression:ElsePart)
                               elseif (type? IfExpression expression:ElsePart)
                                      and (EQUAL expression:ThenPart expression:ElsePart:ThenPart)
                                 then

        (* (If B1 then X else (If B2 then X else Y)) == (If (B1 or B2) then X else Y))

                                    expression←(create IfExpression
                                                      Test ←(MakeOr expression:Test
                                                                    expression:ElsePart:Test)
                                                      ThenPart ← expression:ThenPart
                                                      ElsePart ← expression:ElsePart:ElsePart)
                               elseif (type? IfExpression expression:ThenPart)
                                      and (EQUAL expression:ThenPart:ThenPart expression:ElsePart)
                                 then

        (* (If B1 then (If B2 then X else Y) else X) == (If (B1 imp B2) then X else Y))

                                    expression←(create IfExpression
                                                      Test ←(MakeImplies expression:Test
                                                                         expression:ThenPart:Test)
                                                      ThenPart ← expression:ThenPart:ThenPart
                                                      ElsePart ← expression:ThenPart:ElsePart)
                               elseif (type? IfExpression expression:ElsePart)
                                      and (EQUAL expression:ThenPart expression:ElsePart:ElsePart)
                                 then

        (* (If B1 then X else (If B2 then Y else X)) == (If (B2 imp B1) then X else Y))

                                    expression←(create IfExpression
                                                      Test ←(MakeImplies expression:ElsePart:Test
                                                                         expression:Test)
                                                      ThenPart ← expression:ThenPart
                                                      ElsePart ← expression:ElsePart:ThenPart)
                               else done←T))
    expression))
```

7

```
(MakeOr
  [LAMBDA (x y)                                          (* D. Thompson "28-Jul-80 14:05")

        (* * This routine creates OR expressions, keeping left associativity.)


    (if x=TRUE or y=TRUE
       then TRUE
     elseif x=FALSE
       then y
     elseif y=FALSE
       then x
     elseif y:Operator=IMPOP
       then (MakeImplies (MakeAnd (MakeNot x)
                                  y:Arg1)
                         y:Arg2)
     elseif y:Operator=OROP
       then (create Expression
                    Operator ← OROP
                    Arguments ←(<(MakeOr x y:Arg1)
                      y:Arg2>))
     else (create Expression
                  Operator ← OROP
                  Arguments ←(<x y>])
```

8

```
(RemoveIfs
  [LAMBDA (expression needNotImpForm)                    (* D. Thompson "10-Sep-80 16:32")

        (* * This routine translates expressions in the internal if-then-else form into a form more suitable for
        output, involving the normal Boolean operators.)


    (if needNotImpForm or ~(UserProfile 'UseORinProps T)
       then (RemoveIfs1 expression T)
     else (FindEqvs (RemoveIfs1 expression NIL])
```

9

```
(RemoveIfs1
  [LAMBDA (expression needNotImpForm)                    (* D. Thompson "16-Sep-80 11:26")

        (* * This routine translates expressions in the internal if-then-else form into a form more suitable for
        output, involving the normal Boolean operators.)


    (if (NLISTP expression)
       then                                              (* a simple value. Just return it)
            expression
     elseif (type? IfExpression expression)
       then                                              (* operator is if-then-else.)
            (if needNotImpForm
                then (RemoveIfsHelper (RemoveIfs1 expression:Test T)
                                      (RemoveIfs1 expression:ThenPart T)
                                      (RemoveIfs1 expression:ElsePart T))
              else (RemoveIfsFromIf expression))
     elseif (type? Qexpression expression)
       then                                              (* a Q-expression)
            expression←(create Qexpression
                               expr ←(RemoveIfs1 expression:expr needNotImpForm) using expression)
            (if expression:expr=TRUE
                then TRUE
              else expression)
     else                                              (* operator is not if-then-else.
                                                        Reformulate any if-then-elses in the
                                                        arguments)

          (create Expression
```

```
        Operator ← expression:Operator
        Arguments ←(for argument in expression:Arguments collect (RemoveIfs1 argument
                                                                  needNotImpForm))
```

10

(FindEqvs

```
(LAMBDA (expression)                                   (* D. Thompson "16-Sep-80 11:20")

    (* * This routine searches an expression for possible equivalences, and performs the appropriate
    substitutions.)


  (if (NLISTP expression)
      then expression
    elseif (type? Qexpression expression)
      then (create Qexpression
                   expr ←(FindEqvs expression:expr) using expression)
    elseif ~(type? IfExpression expression)
      then (create Expression
                   Operator ← expression:Operator
                   Arguments ←(for a in expression:Arguments collect (FindEqvs a)))
    elseif (EQUAL expression:ThenPart (MakeNot expression:ElsePart))
      then (MakeEqv (FindEqvs expression:ThenPart)
                    (FindEqvs expression:Test))
    else (create IfExpression
                 Test ←(FindEqvs expression:Test)
                 ThenPart ←(FindEqvs expression:ThenPart)
                 ElsePart ←(FindEqvs expression:ElsePart))
```

11

(RemoveIfsCommonSubExps

```
(LAMBDA (expression)                                   (* D. Thompson "17-Sep-80 11:14")

    (* * This routine uses AssociativeAndMatch to look for common subexpressions in the then and else part of
    if-then-else expressions. -
    I think. I really don't know. (DHT))


  (PROG (aam boolPart elsePart thenPart)
        (RETURN (if (type? IfExpression expression)
                    then boolPart←expression:Test
                         thenPart←expression:ThenPart
                         elsePart←expression:ElsePart
                         (if aam←(AssociativeAndMatch thenPart elsePart)
                             then (MakeAnd (MakeImplies boolPart aam)
                                           elsePart)
                           elseif aam←(AssociativeAndMatch elsePart thenPart)
                             then (MakeAnd (MakeOr boolPart aam)
                                           thenPart)
                           elseif thenPart:Operator=IMPOP
                                  and (if (EQUAL thenPart:Arg2 elsePart)
                                          then (MakeImplies (MakeImplies boolPart thenPart:Arg1)
                                                            elsePart)
                                        elseif elsePart:Operator=IMPOP
                                               and (EQUAL thenPart:Arg2 elsePart:Arg2)
                                               and aam←(AssociativeAndMatch thenPart:Arg1 elsePart:Arg1)
                                          then (MakeImplies (MakeAnd (MakeImplies boolPart aam)
                                                                     elsePart:Arg1)
                                                            elsePart:Arg2))
                           elseif elsePart:Operator=IMPOP
                                  and (if (EQUAL elsePart:Arg2 thenPart)
                                          then (MakeImplies (MakeOr boolPart elsePart:Arg1)
                                                            thenPart)
                                        elseif thenPart:Operator=IMPOP
                                               and (EQUAL elsePart:Arg2 thenPart:Arg2)
                                               and aam←(AssociativeAndMatch elsePart:Arg1 thenPart:Arg1)
                                          then (MakeImplies (MakeAnd (MakeOr boolPart aam)
                                                                     thenPart:Arg1)
```

```
                                                    thenPart:Arg2))
                        else expression)
                else expression]]
```

12

(RemoveIfsFromIf

```
[LAMBDA (expression)                                    (* D. Thompson "17-Sep-80 11:20")
```

(* * This routine rewrites if-then-else expressions into a form more suitable for output, involving normal Boolean operators.)

```
(PROG (boolPart elsePart thenPart)
      (RETURN (if (type? IfExpression expression)
                  then boolPart←(RemoveIfs1 expression:Test)
                       thenPart←(RemoveIfs1 expression:ThenPart)
                       elsePart←(RemoveIfs1 expression:ElsePart)
                       [SELECTQ boolPart
                                (TRUE                          (* if T then X else Y == X)
                                      thenPart)
                                (FALSE                         (* if F then X else Y == Y)
                                       elsePart)
                                (SELECTQ thenPart
                                         [TRUE (SELECTQ elsePart
                                                        (TRUE     (* if B then T else T == T)
                                                              TRUE)
                                                        (FALSE    (* if B then T else F == B)
                                                              boolPart)
                                                        (PROGN    (* if B then T else Y == B or Y)
                                                              (MakeOr boolPart elsePart]
                                         [FALSE (SELECTQ elsePart
                                                         (TRUE    (* if B then F else T == ~B)
                                                              (MakeNot boolPart))
                                                         (FALSE
                                                              (* if B then F else F == F)
                                                              FALSE)
                                                         (PROGN
                                                              (* if B then F else Y == ~B and Y)
                                                              (MakeAnd (MakeNot boolPart)
                                                                       elsePart]
                                (SELECTQ elsePart
                                         (TRUE          (* if B then X else T == B or X)
                                               (MakeImplies boolPart thenPart))
                                         (FALSE         (* if B then X else F == B and X)
                                               (MakeAnd boolPart thenPart))
                                         (PROGN         (* Nothing else worked: try to remove common
                                                        sub-pieces.)
                                               (RemoveCommonParts (create IfExpression
                                                                          Test ← boolPart
                                                                          ThenPart ← thenPart
                                                                          ElsePart ← elsePart]
                  else expression]]
```

13

(RemoveIfsHelper

```
[LAMBDA (Test ThenPart ElsePart)                        (* D. Thompson "28-Jul-80 14:42")
  (if ElsePart=TRUE
      then (MakeImplies Test ThenPart)
    elseif ElsePart=FALSE
      then (MakeAnd Test ThenPart)
    elseif ThenPart=TRUE
      then (MakeImplies (MakeNot Test)
                        ElsePart)
    elseif ThenPart=FALSE
      then (MakeAnd (MakeNot Test)
                    ElsePart)
    elseif [PROG (aam)
                 (aam←(AssociativeAndMatch ThenPart ElsePart))
```

```
                (if aam
                   then (RETURN (MakeAnd (MakeImplies Test aam)
                                         ElsePart))
                  else aam←(AssociativeAndMatch ElsePart ThenPart)
                       (RETURN (if aam
                                   then (MakeAnd (MakeImplies (MakeNot Test)
                                                              aam)
                                                 ThenPart]
      elseif ThenPart:Operator=IMPOP and (if (EQUAL ThenPart:Arg2 ElsePart)
                                             then (MakeImplies (MakeImplies Test ThenPart:Arg1)
                                                               ElsePart)
                                           else ElsePart:Operator=IMPOP
                                                and (EQUAL ThenPart:Arg2 ElsePart:Arg2)
                                                and (PROG (aam)
                                                          (aam←(AssociativeAndMatch ThenPart:Arg1
                                                                                    ElsePart:Arg1))
                                                          (RETURN (if aam
                                                                      then
                                                                       (MakeImplies
                                                                         (MakeAnd (MakeImplies Test aam)
                                                                                  ElsePart:Arg1)
                                                                         ElsePart:Arg2]
      elseif ElsePart:Operator=IMPOP
             and (if (EQUAL ElsePart:Arg2 ThenPart)
                     then (MakeImplies (MakeImplies (MakeNot Test)
                                                    ElsePart:Arg1)
                                       ThenPart)
                   else ThenPart:Operator=IMPOP and (EQUAL ElsePart:Arg2 ThenPart:Arg2)
                        and (PROG (aam)
                                  (aam←(AssociativeAndMatch ElsePart:Arg1 ThenPart:Arg1))
                                  (RETURN (if aam
                                              then (MakeImplies (MakeAnd (MakeImplies (MakeNot Test)
                                                                                      aam)
                                                                         ThenPart:Arg1)
                                                                ThenPart:Arg2]
      else (create IfRecord
                   Test ← Test
                   ThenPart ← ThenPart
                   ElsePart ← ElsePart))
```



```
(SpecialIf
  [NLAMBDA (p% q% r% )                                    (* R.Bates "25-Jun-79 14:58")
    p% ←(EVAL p% )
    (if p% =TRUE
        then (EVAL q% )
      elseif p% =FALSE
        then (EVAL r% )
      else (RESETVARS ((ContinuousEval))
                      (q% ←(EVAL q% ))
                      (r% ←(EVAL r% ))
                      (RETURN (if (EQUAL q% r% )
                                  then q%
                                elseif q% =TRUE and r% =FALSE
                                  then p%
                                else (IfThenElse1 p% q% r% ])
)
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

```
[* (BLOCKS (CevalPropCalcBLOCK AssumedAndDenied IfThenElse IfThenElse0 IfThenElse1 IfThenElse
                    addFact addIntegerFacts falseBranch trueBranch
                    (ENTRIES IfThenElse IfThenElse0 SpecialIf)
                    (GLOBALVARS ContinuousEval)
                    (SPECVARS p q r)
                    (NOLINKFNS . T))
          (CevalEqualityBLOCK TestSubst UseEqHypsInPriorHyps UseEqualities OccursIn Hypotheses
                    ApplyEqInPriorHyps (ENTRIES TestSubst UseEqHypsInPriorHyps UseEqualitie
                              OccursIn)
                    (NOLINKFNS . T] )

(DECLARE: DONTEVAL@LOAD DONTCOPY
```

```
(* ↑ Functions which implement the transformations on propositional calculus expressions.) ]

[DECLARE: EVAL@COMPILE

(RECORD AssumedAndDenied (assumedTrue deniedTrue assumedFalse deniedFalse))
]

(RPAQQ CevalPropCalcFNS (AssumedAndDenied If1? IfThenElse IfThenElse0 IfThenElse1 IfThenElse2 MergeQexsForIf0
                                         OccursIn TopLevelIf addFact addIntegerFacts falseBranch trueBranch))
(DEFINEQ
```

15

```
(AssumedAndDenied
  [LAMBDA (p)                                                    (* R. Bates "20-Mar-80 15:45")
    (PROG (a\d temp xop)
          (temp← <p>)
          (a\d←(create AssumedAndDenied
                       assumedTrue ← temp
                       deniedFalse ← temp))
          (if (xop←p:Operator):EQOP
              then temp←(ReverseEquality p)
                   a\d:assumedTrue← <temp ! a\d:assumedTrue> a\d:deniedFalse← <temp ! a\d:deniedFalse>
                   (if xop='Equal\Integer
                       then a\d:assumedTrue← <<LEOP p:Arg1 p:Arg2> <LEOP p:Arg2 p:Arg1> ! a\d:assumedTrue>
                            a\d:deniedTrue← <<LTOP p:Arg1 p:Arg2> <LTOP p:Arg2 p:Arg1> ! a\d:deniedTrue>)
            elseif xop=LEOP
              then a\d:deniedTrue← <<LTOP p:Arg2 p:Arg1> ! a\d:deniedTrue> a\d:assumedFalse←a\d:deniedTrue
                   a\d:deniedFalse← <<LTOP p:Arg1 p:Arg2> <'Equal\Integer p:Arg1 p:Arg2>
                                    <'Equal\Integer p:Arg2 p:Arg1> ! a\d:deniedFalse>
                   a\d:assumedFalse← <<LEOP p:Arg2 p:Arg1> ! a\d:assumedFalse>
            elseif xop=LTOP
              then a\d:deniedTrue← <<LEOP p:Arg2 p:Arg1> ! a\d:deniedTrue> a\d:assumedFalse←a\d:deniedTrue
                   a\d:assumedTrue← <<LEOP p:Arg1 p:Arg2> ! a\d:assumedTrue> a\d:deniedTrue←
                   <<LTOP p:Arg2 p:Arg1> <'Equal\Integer p:Arg1 p:Arg2> <'Equal\Integer p:Arg2 p:Arg1>
                     ! a\d:deniedTrue>)
          (RETURN a\d))
```

16

```
(If1?
  [LAMBDA (p q r)                                                (* R. Erickson "28-Mar-80 11:43")

        (* * called by IfThenElse0, who has extracted a Qexpression. If Assumed= Denied= NIL, we want to continue
        normally, by calling IfThenElse1. Otherwise, we will wait; some ancestor will rename us
        (so as not to conflict with A/D) and renormalize. In this latter case, we just construct an IFOP expression.)


    (if Assumed or Denied
        then <IFOP p q r>
      else                                                       (* we ask for reeval by IfThenElse1)
           (IfThenElse1 p q r T))
```

17

```
(IfThenElse
  [NLAMBDA (p% q% r% )                                           (* R. Bates " 8-Apr-80 08:47")

        (* * IfThenElse tests for the trivial p= TRUE or FALSE, evals args under Assumed/Denied)


    p% ←(EVAL p% )
    (if p% :Operator=IFOP
        then (IfThenElse1 p%  (EVAL q% )
                          (EVAL r% ))
      elseif p% =TRUE or (MEMBER p%  Assumed)
        then (EVAL q% )
      elseif p% =FALSE or (MEMBER p%  Denied)
        then (EVAL r% )
```

```
    else (PROG (assume\denied)
               (assume\denied←(AssumedAndDenied p% ))
               (q% ←(trueBranch assume\denied q%  r% ))
               (r% ←(falseBranch assume\denied r% ))
               (RETURN (if q% =TRUE and r% =FALSE and p% :Operator~=QOP
                           then <IFOP p%  q%  r% >
                         elseif (EQUAL q%  r% )
                           then q%
                         else (IfThenElse0 p%  q%  r% ))
```

18

(IfThenElse0

```
  (LAMBDA (p q r)                                          (* R. Erickson "28-Mar-80 12:53")
```

(* * IfThenElse0 peels Qexpressions out of the test part (may recurse), renames each Qex to avoid conflicts with the other parts, then calls IfThenElse1. Arguments have been evaluated, using Assumed/Denied. If we find a Qex, we want to be careful not to reevaluate the :expr, if it might have name conflicts with A/D. IfI? does this stuff. Recursion is safe, since we show the caller the Qex.)

```
    (if p:Operator=QOP
       then (if q=TRUE or q=FALSE                          (* (Q T  ))
                then
                     (MergeQexsForIf0 p q r)
              elseif r=TRUE or r=FALSE                     (* (Q   T))
                then                                       (* T tells Merge that q,r are reversed.)
                     (MergeQexsForIf0 p r q T)
              else                                         (* p:Operator=QOP, q,r nonsimple.
                                                           We punt, convert to "P imp Q and not(P) imp R"
; P may get renamed separately, but that's OK.)
                   (IfThenElse0 (IfThenElse0 p q TRUE)
                                (IfThenElse0 p TRUE r)
                                FALSE))
       elseif q:Operator~=QOP and r:Operator~=QOP
         then                                              (* no Qexpressions at all.
                                                           No need to worry if IfThenElse1 decides to
                                                           reevaluate.)
              (IfThenElse1 p q r)
       elseif q=TRUE or q=FALSE                            (* r a Qex, q trivial.)
         then
              r←(RenameBoundVariables r (Frees p))
              (create Qexpression
                      free ←(UNION (Frees p)
                                   r:free)
                      expr ←(IfI? p q r:expr) using r)
       elseif r=TRUE or r=FALSE                            (* q a Qex, r trivial.)
         then
              q←(RenameBoundVariables q (Frees p))
              (create Qexpression
                      free ←(UNION (Frees p)
                                   q:free)
                      expr ←(IfI? p q:expr r) using q)
       else
```

(* One of q or r is a Qex, and the other may contain variables. Rather than be clever and rename the Qex against the other and against p, we split up.)

```
        (IfThenElse0 (IfThenElse0 p q TRUE)
                     (IfThenElse0 p TRUE r)
                     FALSE))
```

19

(IfThenElse1

```
  (LAMBDA (p q r reeval)                                   (* R. Bates " 8-Apr-80 08:47")
```

(* * IfThenElse1 peels IFOPs out of the test, calls IfThenElse2)

```
(* * reeval= T means p:test has changed, so we need to redo the EVAL in RESETVARS)

(if p:Operator=IFOP
   then (IfThenElse p:Test (IfThenElse p:ThenPart (EVAL q)
                                       (EVAL r))
                           (IfThenElse p:ElsePart (EVAL q)
                                       (EVAL r)))
 elseif reeval
   then (if p=TRUE or (MEMBER p Assumed)
           then (EVAL q)
         elseif p=FALSE or (MEMBER p Denied)
           then (EVAL r)
         else (PROG (assume\denied)
                    (assume\denied←(AssumedAndDenied p))
                    (RETURN (IfThenElse2 p (trueBranch assume\denied q r)
                                           (falseBranch assume\denied r]
 else (IfThenElse2 p q r))
```

20

```
(IfThenElse2
  (LAMBDA (p q r)                                   (* D. Musser "16-Jul-79 16:15")
    (if (EQUAL q r)
        then q
      else <IFOP p q r>))
```

21

```
(MergeQexsForIf0
  (LAMBDA (test simple other reverse?)              (* R. Erickson "28-Mar-80 18:11")

        (* * Called by IfThenElse0 to do some of the repetitious work in merging Qexpressions into an IfThenElse.
        test is a Qexpr. We have <IFOP test simple other>. simple is TRUE of FALSE. We rename test, other;
        merge into a <QOP -- <IFOP --»». reverse? means simple and other are switched in the IfThenElse.)


    (PROG ((sense (simple=TRUE)))
          (if reverse?
              then sense←~sense)

          (* * "sense" is the logical sense of test. We have the following cases: -
          "test T other -- test or other" "test F other -- not(test) and other" "test other T -- not(test) or other"
          "test other F -- test and other")


          (test←(RenameBoundVariables test (Frees other)))
          (RETURN (if other:Operator=QOP
                      then other←(RenameBoundVariables other (Variables test))
                           (create Qexpression
                                   given ←(< !(if sense
                                                  then test:given
                                                else test:find)
                                              ! other:given>)
                                   find ←(< !(if sense
                                                  then test:find
                                                else test:given)
                                              ! other:find>)
                                   free ←(UNION test:free other:free)
                                   expr ←(if reverse?
                                             then (If1? test:expr other:expr simple)
                                           else (If1? test:expr simple other:expr)))
                    else (create Qexpression
                                 given ←(if sense
                                             then test:given
                                           else test:find)
                                 find ←(if sense
                                            then test:find
                                          else test:given)
                                 free ←(UNION test:free (Frees other))
```

```
                    expr ←(if reverse?
                              then (If1? test:expr other simple)
                            else (If1? test:expr simple other))
```

22

```
(OccursIn
  (LAMBDA (p q)                                    (* R. Bates "29-Mar-79 13:46")
    (if (EQUAL p q)
        then T
      elseif (NLISTP q)
        then NIL
      else (for x in q:Arguments thereis (OccursIn p x))
```

23

```
(TopLevelIf
  (LAMBDA (p)                                      (* R. Erickson " 2-Nov-79 17:17")

        (* * Ensure that p, unless it is degerate, has an IFOP at the top level. This is necessary if p is to be
        embedded inside a larger expression, because of the way we handle Assumedn and Denied.
        Note that this call is inappropriate for unnormalized exprs, which might have ANDOP etc on top.)


    (if p=TRUE or p=FALSE
        then p
      elseif (NLISTP p)
        then <IFOP p TRUE FALSE>
      elseif p:Operator=IFOP
        then p
      elseif p:Operator=OOP
        then p← < ! p> p:expr←(TopLevelIf p:expr)
             p
      else <IFOP p TRUE FALSE>))
```

24

```
(addFact
  (LAMBDA (fact)                                   (* R. Bates "23-May-80 10:43")
    (PROG (assume\denied)
          ((fact←(APPLY fact:1 fact::1))
          (if fact=FALSE or (MEMBER fact Denied)
              then Assumed← <fact> Denied← <fact> (RETURN FALSE))
          (if fact=TRUE or (MEMBER fact Assumed)
              then (RETURN T))
          (assume\denied←(AssumedAndDenied fact))
          (Denied← < ! assume\denied:deniedTrue ! Denied>)
          (Assumed← < ! assume\denied:assumedTrue ! Assumed>)
          (RETURN T))
```

25

```
(addIntegerFacts
  (LAMBDA (assume denied)                          (* R. Bates "17-Apr-80 13:11")
    (PROG NIL
          (for (p temp) in assume do (if p:Operator=LEOP
                                         then (if (MEMBER <'Equal\Integer ! p:Arguments> Denied)
                                                  then (addFact <LTOP ! p:Arguments>)
                                                                (* i<=j and i~=j imp i<j))
                                              (if (LISTP p:Arg2) and p:Arg2:Operator=ADDOP
                                                    and p:Arg2:Arg2=1
                                                  then (if (MEMBER <LTOP p:Arg2:Arg1 p:Arg1> Assumed)
                                                           then       (* j le i+1 and i<j imp i=j-1 and i+1=j)
                                                                (addFact <'Equal\Integer p:Arg2:Arg1
                                                                         <DIFFOP p:Arg1 1>>)
                                                                (addFact <'Equal\Integer
                                                                         <ADDOP p:Arg2:Arg1 1> p:Arg1>)))
                                              (for x in Assumed do (addFact <LEOP x:Arg1 p:Arg2>)
```

```
                                                    (* i<= j and j<= k imp i<= k)
                                          when x:Operator=LEOP and (EQUAL x:Arg2 p:Arg1))
                                        (for x in Assumed do (addFact <'Equal\Integer ! x:Arguments>)
                                                    (* i<= j and j<= i imp i= j)
                                          when x:Operator=LEOP and (EQUAL x:Arg2 p:Arg1)
                                                and (EQUAL x:Arg1 p:Arg2))
                                    elseif p:Operator=LTOP
                                      then (if (MEMBER <LEOP p:Arg2 <ADDOP p:Arg1 1>> Assumed)
                                               then                 (* i<j and j le i+1 imp i= j-1 and i+1= j)
                                                    (addFact <'Equal\Integer p:Arg1 <DIFFOP p:Arg2 1>>)
                                                    (addFact <'Equal\Integer <ADDOP p:Arg1 1> p:Arg2>)))
    when (LISTP p))
  (for p in denied do (if p:Operator='Equal\Integer
                          then (if (MEMBER <LEOP ! p:Arguments> Assumed)
                                   then (addFact <LTOP ! p:Arguments>)
                                                    (* i~= j and i<= j imp i<j)))
    when (LISTP p))
  (if (for a in Assumed thereis (MEMBER a Denied))
      then (RETURN FALSE))
```

26

```
(falseBranch
  [LAMBDA (assume\denied rr% )                                  (* R.Bates " 8-Apr-80 08:48")
    (RESETVARS ((Denied (< ! assume\denied:deniedFalse ! Denied>))
                (Assumed (< ! assume\denied:assumedFalse ! Assumed>)))
               (RETURN (if (addIntegerFacts assume\denied:assumedFalse assume\denied:deniedFalse)
                           then FALSE
                          else (EVAL rr% ))
```

27

```
(trueBranch
  [LAMBDA (assume\denied qq% rr% )                              (* R.Bates "30-May-80 12:19")
    (if (RESETVARS ((Denied (< ! assume\denied:deniedTrue ! Denied>))
                    (Assumed (< ! assume\denied:assumedTrue ! Assumed>)))
                   (RETURN (if (addIntegerFacts assume\denied:assumedTrue assume\denied:deniedTrue)
                               then NIL
                              else (EVAL qq% )
        else (EVAL rr% ))
)
(DECLARE: DONTEVALeLOAD DONTCOPY
```

```
(* ↑ Functions which implement the "case analysis" rule, f (a1,..., (if p then x else y)
                                                                ,...,an)
    =
    (if p then f (a1,...,x,...,an)
       else f (a1,...,y,...,an))
    when f is not if-then-else.) ]

(RPAQQ CevalCaseAnalFNS (AnyInnerIfs RaiseIfs RaiseIfsHelper))
(DEFINEQ
```

28

```
(AnyInnerIfs
  (LAMBDA (x OtherOpFound)                                    (* Edited by R. Bates on 23-JAN-78;
                                                              from version 12)
    (AND (LISTP x)
         (OR (AND x:Operator=IFOP (OR OtherOpFound (AnyInnerIfs x:Test T)
                                                   (AnyInnerIfs x:ThenPart NIL)
                                                   (AnyInnerIfs x:ElsePart NIL)))
             (AND x:Operator~=IFOP (for y in x:Arguments thereis (AnyInnerIfs y T))
```

29

```
(RaiseIfs
  (LAMBDA (x)                                                 (* D. Musser " 9-Aug-79 15:16")
    (if (NLISTP x)
        then x
      elseif x:Operator=IFOP
        then (IfThenElse (RaiseIfs x:Test)
                         (RaiseIfs x:ThenPart)
                         (RaiseIfs x:ElsePart))
      elseif x:Operator=QUOTEOP
        then x
      else (RaiseIfsHelper x (for y in x:Arguments collect (RaiseIfs y))
                           NIL T))
```

30

```
(RaiseIfsHelper
  (LAMBDA (Original OldArgs NewArgs AnyIfFound)               (* D. Musser " 9-Aug-79 15:18")
    (if OldArgs=NIL
        then (if AnyIfFound
                 then (APPLY Original:Operator (REVERSE NewArgs))
               else Original)
      elseif OldArgs:1:Operator=IFOP
        then (IfThenElse OldArgs:1:Test (RaiseIfsHelper Original <OldArgs:1:ThenPart ! OldArgs::1> NewArgs T)
                         (RaiseIfsHelper Original <OldArgs:1:ElsePart ! OldArgs::1> NewArgs T))
      else (RaiseIfsHelper Original OldArgs::1 <OldArgs:1 ! NewArgs> AnyIfFound])
)
(DECLARE: DONTEVALeLOAD DONTCOPY
```

```
(* † Functions which implement equality rules.) ]

(RPAQQ CevalEqualityFNS (Equal TestSubst UseEqHypsInPriorHyps UseEqualities Hypotheses ApplyEqInPriorHyps))
(DEFINEQ
```

31

```
(Equal
  (LAMBDA (x y)                                    (* R. Erickson " 1-Oct-80 16:30")

          (* * This function seems to be only used by EvaluteRule, except as a placeholder all over the place.
          It is the remnant from before we had separate equality operators. Used to check IsConstant, but not any more.)


    (if (EQUAL x y)
        then TRUE
      elseif (NUMBERP x) and (NUMBERP y)
        then FALSE
      else <'Equal x y>))
```

32

```
(TestSubst
  (LAMBDA (new old x)
    (if (OccursIn old x)
        then (if (EQUAL x old)
                 then new
               else (SUBST new old x))
      else x))
```

33

```
(UseEqHypsInPriorHyps
  (LAMBDA (x)
    (if x:Operator=IFOP
        then (if x:ThenPart=TRUE or x:ElsePart=TRUE
                 then (PROG (y)
                            (y←(Hypotheses x))
                            (for h in y::1 when h:Operator=EQOP do x←(ApplyEqInPriorHyps h x))
                            (RETURN x))
               else (PROG (y z)
                          (y←(UseEqHypsInPriorHyps x:ThenPart))
                          (z←(UseEqHypsInPriorHyps x:ElsePart))
                          (RETURN (if y=x:ThenPart and z=x:ElsePart
                                      then x
                                    else <IFOP x:Test y z>)
    else x))
```

34

```
(UseEqualities
  (LAMBDA (x)
    (if x:Operator~=IFOP
        then x
      else (PROG (y z)
                 (if x:Test:Operator=EQOP
                     then y←(UseEqualities (TestSubst x:Test:RHS x:Test:LHS x:ThenPart))
                          z←(UseEqualities (TestSubst FALSE (create Equation
                                                                    LHS ← x:Test:RHS
                                                                    RHS ← x:Test:LHS)
                                                            x:ElsePart))
                   else y←(UseEqualities x:ThenPart)
                        z←(UseEqualities x:ElsePart))
                 (RETURN (if y=x:ThenPart and z=x:ElsePart
                             then x
                           else <IFOP x:Test y z>)
```

35

```
(Hypotheses
  [LAMBDA (x)
    (if x:Operator=IFOP
        then (if x:ThenPart=TRUE
                 then <<NOTOP x:Test> !(Hypotheses x:ElsePart) >
               elseif x:ElsePart=TRUE
                 then <x:Test !(Hypotheses x:ThenPart)
                        >])
```

36

```
(ApplyEqInPriorHyps
  [LAMBDA (h x)
    (if h=x:Test
        then x
      else (PROG (y z)
                 (y←(TestSubst h:RHS h:LHS x:Test))
                 (if x:ThenPart=TRUE
                     then z←(ApplyEqInPriorHyps h x:ElsePart)
                          (RETURN (if y=x:Test and z=x:ElsePart
                                      then x
                                    else <IFOP y TRUE z>))
                   else z←(ApplyEqInPriorHyps h x:ThenPart)
                        (RETURN (if y=x:Test and z=x:ThenPart
                                    then x
                                  else <IFOP y z TRUE>])
)

(RPAQQ SEPTHEORYRECORDS (Arc Sepnode))
[DECLARE: EVAL@COMPILE

(RECORD Arc (Name Val))

(RECORD Sepnode (Name . Outarcs))
]

(RPAQQ septheory (AddArc Separation SeparationContradiction UseSeparations normint))
(DEFINEQ
```

37

```
(AddArc
  [LAMBDA (g i c j)                                   (* D. Musser "29-Apr-80 07:34")

          (* Assuming g is a transitively closed graph of nodes representing integer variables, and arcs labeled by
          integer constants representing "separations" between the variables, add the nodes i and j with separation c
          and close the graph, returning the closed graph. Nodes i and j and the separation c correspond to the
          inequality relation i - c <= j %. It is assumed that each node k in g is connected to itself with an arc
          labeled with separation 0. Only arcs with maximal separation are retained in the graph.)


    (if ~(SASSOC i g)
        then g← <(create Sepnode
                         Name ← i
                         Outarcs ←(<(create Arc
                                            Name ← i
                                            Val ← 0)
                           >))
                 ! g>)
    (if ~(SASSOC j g)
        then g← <(create Sepnode
                         Name ← j
                         Outarcs ←(<(create Arc
                                            Name ← j
                                            Val ← 0)
                           >))
                 ! g>)
    (for node in g bind (jarcs ←(SASSOC j g):Outarcs)
```

```
        collect (create Sepnode
                        Name ← node:Name
                        Outarcs ←(PROG (oldarcs newarcs)
                                       [newarcs←(for arc in node:Outarcs
                                                   join (if (EQUAL arc:Name 1)
                                                            then (for arc1 in jarcs bind check
                                                                    eachtime check←(SASSOC arc1:Name
                                                                                           node:Outarcs)
                                                                    when ~check
                                                                         or (LESSP check:Val
                                                                                   arc:Val+c+arc1:Val)
                                                                    collect (create Arc
                                                                                   Name ← arc1:Name
                                                                                   Val ←(arc:Val+c+arc1:Val]
                                       (oldarcs←(for arc in node:Outarcs unless (SASSOC arc:Name newarcs)
                                                   collect arc))
                                       (RETURN < | oldarcs | newarcs>])
```

38

(Separation

```
  [LAMBDA (p)                                                    (* R. Bates "26-Jun-80 12:52")

        (* Given a predicate expression p in the form produced by the XEVAL simplifier, convert it to a
        "separation triple" (x c y) representing the relation x + c rel y, where "rel" is the less-than-or-equal,
        equality, or strictly-less-than relation between integers.  If p cannot be put in this form, NIL is returned.
        In the triple, x and y are integer terms or the constant 0 (but both cannot be 0) and c is an integer
        constant.  By "term" is meant any nonconstant integer expression in which the main operator is neither + nor
        -.)


    (if p:Operator MEMB <LEQP LTQP 'Equal\Integer >
        then (PROG (difference)
                   (difference←(XEVAL <DIFFOP p:LHS p:RHS>))

        (* The above could be (SIMP\DIFF p:LHS p:RHS) if p:LHS and p:RHS were guaranteed to be in XEVAL canonical
        form)


                   (RETURN (if difference:Operator=ADDOP
                               then (if difference:Arg1:Operator=NEGOP
                                        then                          (* form is (+ (-x) --))
                                             (if (FIXP difference:Arg2)
                                                 then                 (* form is (+ (-x) c))
                                                      <0 difference:Arg2 difference:Arg1:Arg1>
                                               elseif difference:Arg2:Operator~=NEGOP
                                                 then                 (* form is (+ (-x) y --))
                                                      (if difference:Arg3
                                                          then (if (FIXP difference:Arg3)
                                                                   then
                                                                   (* form is (+ (-x) y c) where c is an integer
                                                                   constant)
                                                                     <difference:Arg2 difference:Arg3
                                                                       difference:Arg1:Arg1>)
                                                                 else <difference:Arg2 0 difference:Arg1:Arg1>))
                                      elseif (FIXP difference:Arg2)
                                        then                          (* form is (+ x c))
                                             <difference:Arg1 difference:Arg2 0>)
                                    elseif difference:Operator=NEGOP
                                      then <0 0 difference:Arg1>
                                    else <difference 0 0>])
```

39

(SeparationContradiction

```
  [LAMBDA (g conjuncts lastpredicate negated)                    (* D. Thompson "27-Aug-80 11:38")
    (for node in g thereis (for arc in node:Outarcs
                              thereis (if (EQUAL arc:Name node:Name) and (LESSP 0 arc:Val)
                                          then (if ShowIntegerSimplification
                                                   then (printout NIL .TAB0 0 "Contradiction found:" #
```

```
                                                    (PrettyPrint (N2BINARY
                                                                  <ANDOP ! <
                                                                    !!(REVERSE conjuncts)
                                                                    (if negated
                                                                        then
                                                                         <NOTOP lastpredicate>
                                                                      else lastpredicate)
                                                                    >>]
                                                T))
```

40

```
(UseSeparations
  [LAMBDA (predicate graph conjuncts)                    (* R. Bates "26-Jun-80 12:54")
    (if (NLISTP predicate) OR predicate:Operator~=IFOP
        then predicate
      else (PROG (sep x c y thenGraph elseGraph equality? useEq)
                 (sep←(Separation predicate:Test))
                 (RETURN (if sep=NIL
                             then <IFOP predicate:Test (UseSeparations predicate:ThenPart graph conjuncts)
                                        (UseSeparations predicate:ElsePart graph conjuncts) >
                           else x←sep:1
                                c←sep:2
                                y←sep:3
                                (if predicate:Test:Operator=LEOP
                                    then thenGraph←(AddArc graph x c y)
                                         elseGraph←(AddArc graph y 1-c x)
                                  elseif predicate:Test:Operator=LTOP.
                                    then thenGraph←(AddArc graph x c+1 y)
                                         elseGraph←(AddArc graph y (-c)
                                                           x)
                                  else equality?←T)
                                (if ~equality?
                                    then (if (SeparationContradiction thenGraph conjuncts predicate:Test)
                                             then (UseSeparations predicate:ElsePart elseGraph
                                                                  <<NOTOP predicate:Test> ! conjuncts>)
                                           elseif (SeparationContradiction elseGraph conjuncts predicate:Test
                                                                           T)
                                             then (UseSeparations predicate:ThenPart thenGraph
                                                                  <predicate:Test ! conjuncts>)
                                           else <IFOP predicate:Test (UseSeparations predicate:ThenPart
                                                                                    thenGraph
                                                                                    <predicate:Test
                                                                                      ! conjuncts>)
                                                                     (UseSeparations predicate:ElsePart elseGraph
                                                                                     <<NOTOP predicate:Test> ! conjuncts>)
                                                                     >)
                                  else thenGraph←(AddArc (AddArc graph x c y)
                                                         y
                                                         (-c)
                                                         x)
                                       (if (SeparationContradiction thenGraph conjuncts predicate:Test)
                                           then (if (SeparationContradiction (AddArc graph x c+1 y)
                                                                             conjuncts
                                                                             <LEOP <ADDOP x c+1> y>)
                                                    then (UseSeparations predicate:ElsePart
                                                                         (AddArc graph y 1-c x)
                                                                         <<NOTOP predicate:Test>
                                                                           ! conjuncts>)
                                                  else (UseSeparations predicate:ElsePart
                                                                       (AddArc graph x c+1 y)
                                                                       <<NOTOP predicate:Test> ! conjuncts>))
                                         else useEq←(UseSeparations predicate:ThenPart thenGraph
                                                                    <predicate:Test ! conjuncts>)
                                              <IFOP predicate:Test useEq
                                                (if (SeparationContradiction (AddArc graph x c+1 y)
                                                                             conjuncts
                                                                             <LEOP <ADDOP x c+1> y>)
                                                    then (if (SeparationContradiction (AddArc graph y 1-c x)
                                                                                      conjuncts
                                                                                      <LEOP <ADDOP y (1-c) >
```

```
                                                                x>)
                                then useEq
                                else (UseSeparations predicate:ElsePart
                                                     (AddArc graph y 1-c x)
                                                     <<NOTOP predicate:Test>
                                                       ! conjuncts>))
                            elseif (SeparationContradiction (AddArc graph y 1-c x)
                                                            conjuncts <LEOP
                                                                        <ADDOP y (1-c)
                                                                          >
                                                                        x>)
                              then (UseSeparations predicate:ElsePart
                                                   (AddArc graph x c+1 y)
                                                   <<NOTOP predicate:Test>
                                                     ! conjuncts>)
                            else <IFOP <LEOP <ADDOP x c+1> y>
                                       (UseSeparations predicate:ElsePart
                                                       (AddArc graph x c+1 y)
                                                       <<LEOP <ADDOP x c+1> y>
                                                         ! conjuncts>)
                                       (UseSeparations predicate:ElsePart
                                                       (AddArc graph y 1-c x)
                                                       <<LEOP <ADDOP y (1-c) > x>
                                                         ! conjuncts>)
                                       >
                          >])
```

41

```
(normint
  (LAMBDA NIL                                        (* D. Thompson " 8-Sep-80 16:30")

        (* * This routine implements the NORMINT command, a simple integer inequality solver.)


    (PROG (result)
          (CheckForQexpression CurrentPropn)
          (result←(create Qexpression
                          expr ←(UseSeparations CurrentPropn:expr) using CurrentPropn))
          (if (EQUAL result CurrentPropn) or (EQUAL result←(EVAL result)
                                                    CurrentPropn)
              then (if InAutoMechanism
                       else (printout NIL .TAB0 0 "Normint had no effect." T))
                   (RETURN NIL)
            else (RETURN (Descend (Transform (create Transformation
                                                     command ←('normint)
                                                     children ←(<result>))
                                             NIL 'normint])
)
(DECLARE: DONTEVALeLOAD DOEVALeCOMPILE DONTCOPY COMPILERVARS

(ADDTOVAR NLAMA )

(ADDTOVAR NLAML IfThenElse SpecialIf)

(ADDTOVAR LAMA )
)
(DECLARE: DONTCOPY
  (FILEMAP (NIL (3614 21674 (AssociativeAndMatch 3626 . 4222) (MakeAnd 4226 . 4846) (MakeEqv 4850 . 5524) (
MakeImplies 5528 . 6168) (MakeNot 6172 . 7893) (RemoveCommonParts 7897 . 10352) (MakeOr 10356 . 11101) (
RemoveIfs 11105 . 11609) (RemoveIfs1 11613 . 13315) (FindEqvs 13319 . 14451) (RemoveIfsCommonSubExps 14455 .
16522) (RemoveIfsFromIf 16526 . 18752) (RemoveIfsHelper 18756 . 21091) (SpecialIf 21095 . 21671)) (22677 36735
 (AssumedAndDenied 22689 . 24087) (If1? 24091 . 24778) (IfThenElse 24782 . 25806) (IfThenElse0 25810 . 28487)
(IfThenElse1 28491 . 29521) (IfThenElse2 29525 . 29732) (MergeQexsForIf0 29736 . 31701) (OccursIn 31705 .
32017) (TopLevelIf 32021 . 32796) (addFact 32800 . 33455) (addIntegerFacts 33459 . 35817) (falseBranch 35821 .
 36247) (trueBranch 36251 . 36732)) (37078 38736 (AnyInnerIfs 37090 . 37584) (RaiseIfs 37588 . 38095) (
RaiseIfsHelper 38099 . 38733)) (38950 41842 (Equal 38962 . 39498) (TestSubst 39502 . 39708) (
UseEqHypsInPriorHyps 39712 . 40354) (UseEqualities 40358 . 40997) (Hypotheses 41001 . 41267) (
ApplyEqInPriorHyps 41271 . 41839)) (42102 51774 (AddArc 42114 . 44048) (Separation 44052 . 46297) (
SeparationContradiction 46301 . 47037) (UseSeparations 47041 . 50871) (normint 50875 . 51771)))))
STOP
```