```
   let Expression[] be
   §E
       SubExpression[0]
       OuttoBuffer[DUMMY, DUMMY]
       unless FHN = UNSET do CoercetoVal[]
   $E


   and LeftExpression[] be
   §LE
       SubExpression[0]
       OuttoBuffer[ASS, ASS]
       FHN := UNSET
   $LE


   and CondExpression[x] = valof
   §CE
       SubExpression[0]
       OuttoBuffer[DUMMY, DUMMY]
       test FHN = UNSET
         ifso  CoercetoBool[x]
         ifnot if IsRevBoolType[x] do ReverseBoolHops[]
     § let f = FHN
       FHN := UNSET
       resultis LastRelExt → f, -f
   $CE


   and ConstantExpression[] = valof
   §CE
       ClearStack[]
       StartOutputMode[CONSTANT]
       Expression[]
       unless NormalOutput = NullProgram do  || see ConstOut
           unless StackPtr = 1 do CompilerError[3201]
       EndOutputMode[]
       StackPtr := 0
       SSP := SSP - 1
       resultis Stack↓1
   $CE


   and SubExpression[p] be
   §E
       let ExpOp = RAND
       §r
           Read[]
         § let x = Ch
           switchon ChType into
           §s
               case SBRA:
                       unless ExpOp = RATOR do
                       §   Error[192, HARD, Ch, BackUp, TRUE]
                           endcase
                       $
                       OuttoBuffer[DUMMY, DUMMY]
                       Application[FNEXIT]
                       break
```

```
            case RBRA:
                    unless ExpOp = RAND do
                    §   Error[191, HARD, Ch, BackUp, PLUS]
                        endcase
                    §
                    Addto[ShuntVec, Ch]
                    SubExpression[0]
                    unless Ch = RKET do CompilerError[3202]
                    ExpOp := RATOR
                    break

            case PLUS:
                    if ExpOp = RAND do
                    §   UpdateCh[UPLUS]
                        endcase
                    §

            case MINUS:
                    if ExpOp = RAND do
                    §   UpdateCh[UMINUS]
                        endcase
                    §
            case MULT:
            case DIV:
            case REM:
            case LSHIFT:
            case RSHIFT:
            case VECOP:
            case EQV:
            case NEQV:
            case EQ:
            case NE:
            case LT:
            case LE:
            case GT:
            case GE:unless ExpOp = RATOR do
                    §   Error[192, HARD, Ch, BackUp, TRUE]
                        endcase
                    §
                    ExpOp := RAND
                    EmptyShunt[]
                    if Prec[LEFT, Ch] ≤ p return
                    unless FHN = UNSET do CoercetoVal[]
                    Addto[ShuntVec, Ch]
                    break

            case LOGAND:
            case LOGOR:
                  § let x = Ch
                    let DefVal = (x = LOGAND → FALSE, TRUE)
                    unless ExpOp = RATOR do
                    §   Error[192, HARD, Ch, BackUp, TRUE]
                        endcase
                    §
                    EmptyShunt[]
                    if Prec[LEFT, Ch] ≤ p return
                    Addto[ShuntVec, SUBEXP]
                    test FHN = UNSET
                      ifso
                        §   SubExpression[Prec[RIGHT, x]]
                            OuttoBuffer[DUMMY, DUMMY]
                            unless Top[ShuntVec] = SUBEXP do
                                CompilerError[3203]
                            Subtractfrom[ShuntVec]
                            test FHN = UNSET
```

```
                                ifso  Addto[ShuntVec, x]
                                ifnot
                                  §   if x = LOGAND do ReverseBoolHops[]
130                                   UpdateS[SSP - 1]
                                      RightVersion[DefVal, DefVal]
                                      ForHopPt[FHN]
                                      if LastRelExt do UpdateS[SSP]
                                      FHN := UNSET
135                               §
                            §
                          ifnot
                            §1  if x = LOGOR do ReverseBoolHops[]
                              § let f = FHN
140                             let LRE = LastRelExt
                                FHN := UNSET
                                SubExpression[Prec[RIGHT, x]]
                                OuttoBuffer[DUMMY, DUMMY]
                                unless Top[ShuntVec] = SUBEXP do
145                                 CompilerError[3204]
                                Subtractfrom[ShuntVec]
                                test FHN = UNSET
                                  ifso
                                    §   let HN1 = HopNo[]
150                                     ForHop[HN1]
                                        ForHopPt[f]
                                        test LRE
                                          ifso  UpdateS[SSP - 1]
                                          ifnot SSP := SSP - 1
155                                     RightVersion[DefVal, DefVal]
                                        ForHopPt[HN1]
                                    §
                                  ifnot
                                    §   if x = LOGAND do ReverseBoolHops[]
160                                     ForHopPt[f]
                                        if LRE do UpdateS[SSP]
                                        if x = LOGAND do ReverseBoolHops[]
                                    §
                            §1
165                   endcase
                    §

                case UPLUS:
                case UMINUS:
170             case LV:
                case RV:
                        unless ExpOp = RAND do
                        §   Error[191, HARD, Ch, BackUp, PLUS]
                            endcase
175                     §
                        Addto[ShuntVec, Ch]
                        break

                case NOT:
180                     unless ExpOp = RAND do
                        §   Error[191, HARD, Ch, BackUp, PLUS]
                            endcase
                        §
                        Addto[ShuntVec, SUBEXP]
185                     SubExpression[Prec[RIGHT, NOT]]
                        OuttoBuffer[DUMMY, DUMMY]
                        unless Top[ShuntVec] = SUBEXP do CompilerError[3205]
                        Subtractfrom[ShuntVec]
                        test FHN = UNSET
                          ifso  Addto[ShuntVec, NOT]
190                       ifnot ReverseBoolHops[]
```

```
                    ExpOp := RATOR
                    endcase

195             case DO:
                case LET:
                case AND:
                case COMMA:
                case RKET:
200             case SKET:
                case ASS:
                case TO:
                case BY:
                case INTO:
205             case CCOLON:
                case SECTBRA:|| in case a global type def follows a let type
                case SECTKET:
                case SEMICOLON:
                case OR:
210             case IFSO:
                case IFNOT:
                case REPEAT:
                case REPEATUNTIL:
                case REPEATWHILE:
215             case ENDBODY:
                case ENDFOR:
                case ENDVALOF:
                    unless ExpOp = RATOR ∨ Ch = SKET do
                    §   Error[192, HARD, Ch, BackUp, TRUE]
220                     endcase
                    §
                    EmptyShunt[]
                    return

225             case COND:
                    unless ExpOp = RATOR do
                    §   Error[192, HARD, Ch, BackUp, TRUE]
                        endcase
                    §
230                 EmptyShunt[]
                    if Prec[LEFT, Ch] ≤ p return
                    OuttoBuffer[DUMMY, DUMMY]
                    if FHN = UNSET do CoercetoBool[COND]
                  § let f, LRE, h = FHN, LastRelExt, HopNo[]
235                 let s = SSP
                    FHN := UNSET
                    Addto[ShuntVec, COND]
                    SubExpression[0]
                    OuttoBuffer[DUMMY, DUMMY]
240                 if BoolValueonStack[] do CoercetoBool[COND]
                    SSP := s    ||  reset after left branch
                    unless Ch = COMMA do CompilerError[3206]
                    ForHop[h]
                    ForHopPt[f]
245                 if LRE do UpdateS[SSP]
                    test FHN = UNSET
                      ifso
                        §   Expression[]
                            OuttoBuffer[DUMMY, DUMMY]
250                         ForHopPt[h]
                        §
                      ifnot
                        §   let F2, H2 = FHN, HopNo[]
                            and LRE2 = LastRelExt
255                         FHN := UNSET
                            SubExpression[0]
```

```
                                OuttoBuffer[DUMMY, DUMMY]
                                if BoolValueonStack[] do
                                    CoercetoBool[COND]
260                             ForHop[H2]
                                test FHN = UNSET
                                  ifso
                                    §   ForHopPt[h]
                                        FHN := F2
265                                     LastRelExt := LRE2
                                        SSP := SSP - 1
                                        CoercetoVal[]
                                    §
                                  ifnot
270                                 §   ForHopPt[F2]
                                        ForHop[FHN]
                                        LastRelExt := LastRelExt ∨ LRE2
                                        ForHopPt[h]
                                    §
275                             ForHopPt[H2]
                            §
                        unless Top[ShuntVec] = COND do
                            CompilerError[3207]
                        Subtractfrom[ShuntVec]
280                     ExpOp := RATOR
                        endcase
                    §

                case VECAP:
285                     unless ExpOp = RATOR do
                        §   Error[192, HARD, Ch, BackUp, TRUE]
                            endcase
                        §
                        Expression[]
290                     unless Ch = SKET do CompilerError[3208]
                        ExpOp := RATOR
                        break

                case VALOF:
295                     unless ExpOp = RAND do
                        §   Error[191, HARD, Ch, BackUp, PLUS]
                            endcase
                        §
                        OuttoBuffer[DUMMY, DUMMY]
300                     Valof[]
                        ExpOp := RATOR
                        break

                case NAME:
305             case NUMBER:
                case CHARACTER:
                case STRING:
                case TRUE:
                case FALSE:
310                     unless ExpOp = RAND do
                        §   Error[191, HARD, Ch, BackUp, PLUS]
                            endcase
                        §
                        ExpOp := RATOR
315                     OuttoBuffer[Ch, ChType]
                        break

                default:CompilerError[3209]
            $s repeat
320     $r repeat
    $E
```

```
    and Valof[] be
325 §V
        let RVHN = VHN
        and RVSSP = VSSP
        and l, g = LineNo, GetNo
        VSSP := SSP + 1
330     VHN := UnusedHopNo[]
        Addto[ShuntVec, Ch]
        Command[]
        unless Ch = ENDVALOF do CompilerError[3210]
        unless HopNoUsed[VHN] do
335     §   let Line, Get = LineNo, GetNo
            LineNo, GetNo := l, g
            Error[260, HARD, DUMMY, NullProgram]
            LineNo, GetNo := Line, Get
        $
340     ForHopPt[VHN]
        SSP := VSSP
        unless Top[ShuntVec] = VALOF do CompilerError[3211]
        Subtractfrom[ShuntVec]
        VSSP := RVSSP
345     VHN := RVHN
    $V


    and EmptyShunt[] be
350 §ES
        let n = Prec[LEFT, ChType]
        while n < Prec[RIGHT, Top[ShuntVec]] do
            test IsRelop[Top[ShuntVec]]
              ifso  BoolRelation[]
355           ifnot ShunttoBuffer[]
        if n = Prec[RIGHT, Top[ShuntVec]] do
            test Ch = RKET ∨ Ch = SKET
              ifso  Subtractfrom[ShuntVec]
              ifnot ShunttoBuffer[]
360 $ES


    and BoolRelation[] be
    §BR
365     let Relop = Top[ShuntVec]
        and Result = true
        Subtractfrom[ShuntVec]
        OuttoBuffer[DUMMY, DUMMY]
        FHN := HopNo[]
370     LastRelExt := false
        §r
            switchon StackSize[] into
            §s
                case 1: if Result = false do
375                     §   StackPtr := StackPtr - 1
                            SSP := SSP - 1
                            endcase
                        §

380             case 0: if Result = false do
                        §   UpdateS[SSP - 1]
                            endcase
                        §

385                     test IsRelop[Top[ShuntVec]]
                          ifso  §   Code[RPsm, SSP - 1]
```

```
                                CondForHop[BoolType[Relop], FHN]
                                LastRelExt := true
                            §
390                   ifnot §   SpecCondForHop[BoolType2[Relop], FHN]
                                return
                            §
                    endcase

395         default:
                    Result := Result ∧
                        ConstOp[Relop, Stack↓(StackPtr - 1),
                            Stack↓StackPtr]
                    StackPtr := StackPtr - 1
400                 SSP := SSP - 1
                    endcase
        §s

        Relop := Top[ShuntVec]
405     test IsRelop[Relop]
          ifso  Subtractfrom[ShuntVec]
          ifnot §   FHN := UNSET
                    test StackSize[] > 0
                      ifso  Stack↓StackPtr := Result
410                   ifnot §   UpdateS[SSP - 1]
                                CodeNsm[Result]
                            §
                    return
                §
415 §r repeat
§BR


and SpecCondForHop[Type, n] be
420 §SCFH
    if (Type = HOPIFEQsm ∨ Type = HOPIFNEsm) do
        if StackSize[] > 0 ∧ Stack↓StackPtr = 0 do
        §   Type := (Type = HOPIFEQsm → HOPIFZEsm, HOPIFNZsm)
            StackPtr := StackPtr - 1
425         SSP := SSP - 1
        §
    CondForHop[Type, n]
§SCFH

430
and IsRelop[CType] = (Prec[RIGHT, CType] = 11)


and BoolType[Op] = valof
435 §BT
    switchon Op into
    §s
        case EQ:resultis HOPIFNEsm
        case NE:resultis HOPIFEQsm
440     case LT:resultis HOPIFGEsm
        case GE:resultis HOPIFLTsm
        case LE:resultis HOPIFGTsm
        case GT:resultis HOPIFLEsm
    §s
445 §BT


and BoolType2[Op] = valof
§BT
450 switchon Op into
    §s
```

```
            case EQ:resultis HOPIFNEsm
            case NE:resultis HOPIFEQsm
            case LT:resultis HOPIFLEsm
            case GE:resultis HOPIFGTsm
            case LE:resultis HOPIFLTsm
            case GT:resultis HOPIFGEsm
        $s
    $BT

    and CoercetoBool[x] be
    §CB
        FHN := HopNo[]
        LastRelExt := false
        OuttoBuffer[DUMMY, DUMMY]
        test StackSize[] > 0
          ifso  §   let Bool = Stack↓StackPtr
                    StackPtr := StackPtr - 1
                    SSP := SSP - 1
                    if IsRevBoolType[x] ≠ (Bool = 0) do ForHop[FHN]
                §
          ifnot CondForHop[(IsRevBoolType[x] → HOPIFNZsm, HOPIFZEsm), FHN]
    $CB

    and CoercetoVal[] be
    §CV
        OuttoBuffer[DUMMY, DUMMY]
        ClearStack[]
        NormalOutput[3, COERCETOVAL, FHN, LastRelExt → SSP, UNSET]
        UpdateSSP[COERCETOVAL]
        FHN := UNSET
    $CV

    and ReverseBoolHops[] be
    §RBH
        let h = FHN
        FHN := HopNo[]
        ForHop[FHN]
        ForHopPt[h]
        if LastRelExt do
        §   UpdateS[SSP]
            LastRelExt := false
        §
    $RBH


and IsRevBoolType[x] = valof
    §IR
        switchon x into
        §s
            case IF:
            case WHILE:
            case COND:
            case REPEATWHILE:
                    resultis false

            case UNLESS:
            case UNTIL:
            case REPEATUNTIL:
                    resultis true

            case TEST:
                    resultis (Ch = IFNOT)
```

```
            default:CompilerError[3212]
        $s
$IR


    and ShunttoBuffer[] be
    §SB
        let t = Top[ShuntVec]
        unless FHN = UNSET do CoercetoVal[]
        OuttoBuffer[t, t]
        Subtractfrom[ShuntVec]
    $SB


    and OuttoBuffer[C, CType] be
    §OB
        let l = LineNo
        LineNo := BufferLine
        BufferLine := l
        switchon CType into
        §s
            case LV:LeftVersion[Buffer, BufferType]
                    endcase

            case ASS:
                    StoreVersion[Buffer, BufferType]
                    endcase

            default:RightVersion[Buffer, BufferType]
                    endcase
        $s
        Buffer := C
        BufferType := CType
        LineNo := BufferLine
    $OB

****
```