<AFFIRM>PASCAL..22 17-Jun-81 16:36:07

```
(FILECREATED "31-Mar-81 19:51:16" <AFFIRM>PASCAL..22 10365

     previous date: "28-Mar-81 15:17:18" <AFFIRM>PASCAL..21)


(PRETTYCOMPRINT PASCALCOMS)

(RPAQQ PASCALCOMS ((FNS * PASCALFNS)))

(RPAQQ PASCALFNS (InitializeAffirmReadTable PARSE genvcs readp setupDeclarations skipPascalComment
                                            typecheck typeifyProgramUnit typeifyStatement))
(DEFINEQ
```

1

```
(InitializeAffirmReadTable
  [LAMBDA NIL                                          (* D.Thompson "27-Jun-79 12:06")
    (PROG NIL
          (PASCAL\READ\TABLE←(COPYREADTABLE 'ORIG))
          (SETBRK '(4 5 6 14 16 17 18 19 20 21 27 28 29 30 ! %" & %( %) * + , - %. / : ; < = > @ %[
                      %] ↑ { %| } ~)
                  NIL PASCAL\READ\TABLE)
          (SETSYNTAX '{ '(SPLICE IMMEDIATE skipPascalComment)
                     PASCAL\READ\TABLE])
```

2

```
(PARSE
  [LAMBDA (FILE)                                       (* D.Thompson "5-Oct-79 15:31")
    !VALUE ←(PARSEPROGRAM (INPUT (INFILE FILE)))
    (CLOSEF FILE)
    ~(EQUAL !VALUE NIL])
```

3

```
(genvcs
  [LAMBDA (unitnames)                                  (* R.Erickson "3-Dec-80 17:59")

          (* * This routine generates the verification conditions for the Pascal unit names provided as its parameter
          The units must have been previously parsed -
          An AFFIRM command function *)


    (PROG (computes foundOne legalNames parent trans uname unit vcs node lemmas #vcs)
          (legalNames←(for p in LastProgramUnits collect p:1))
          (for u in unitnames do (uname←(AFFIRMSpellingCorrect u legalNames NIL T NIL))
                                 (unit←(FASSOC uname LastProgramUnits)::1)
                                 (if unit
                                     then foundOne←T
                                          vcs←(SETUPCALLVCGEN unit)
                                          computes←vcs:1
                                          vcs←vcs::1

          (* * Do the computes lemma)


                                          node←(ExprToNode computes)
                                          (if ~(True? node)
                                              then (Assume node)
                                                   (SetName (PACK* 'computes uname)
                                                            node))

          (* * Do the verification conditions )


                                          vcs←(REMOVE TRUE vcs)
                                          #vcs←(LENGTH vcs)
                                          (printout NIL .TAB0 0 T ('The)
                                                    , #vcs , ('verification)
                                                    ,
                                                    (Plural 'condition #vcs)
                                                    ,
                                                    ('for)
                                                    , uname , (Plural 'is #vcs)
                                                    (':))

          (* * Set up the kludgey tree structure.)
```

```
                                    (for vc in vcs as n from 1 do (node←(ExprToNode vc))
                                                                  (MakeTheorem node)
                                                                  (SetName (PACK* uname '# n)
                                                                           node)
                                                                  (printPropn node))
                                    parent←(ExprToNode <'verification\ProcedureCall
                                                        (FindSubexpressions
                                                             'computes\ProcedureCall
                                                                computes)
                                                        :1:Arg1:Operator>)
                                    (/SET 'CurrentTheorem (MakeTheorem parent))
                                    (Annotate uname parent 'PROGRAM)
                                    (SetName uname parent)
                                    (MakeCurrent parent)
                                    lemmas←(if computes=TRUE
                                               then vcs
                                              else < ! vcs computes>)
                                    trans←(create Transformation
                                                  uses ← lemmas)
                                          (* no longer use the labels: vclabels←
                                          (for i to (FLENGTH vcs) collect
                                          (PACK* 'VC i Colon)))
                                    (Transform trans parent)
                               else (printout NIL .TAB0 0 u "??" T))
                          (printout NIL T))
        (if ~foundOne
            then (AffirmError "(No parameter! Expecting a list of Pascal program unit names.)"])
```

4

```
(readp
  [LAMBDA (fileName)                                    (* D.Thompson "22-Sep-80 12:26")

        (* * This routine parses the pascal unit contained the file provided as the parameter -
        An AFFIRM command function.)


    (PROG (unitNames)
          (if fileName←(getFileName fileName NIL "File of Pascal programs: ")
              then (printout NIL .TAB0 0 "(Reading Pascal program units from" . fileName ")" T)
                   (RESETLST (RESETSAVE NIL <'CLOSEF? fileName>)
                             (PARSE fileName))
                   (if !VALUE
                       then unitNames←(for x in LastProgramUnits collect x:1)
                            (AFFIRMMAPRINT (Plural "The program units are" unitNames)
                                           unitNames T T)
                            (typecheck)!VALUE←NIL
                     else (ParsingError T))
            else (AffirmError])
```

5

```
(setupDeclarations
  [LAMBDA (programUnit)                                 (* R.Erickson "23-Feb-81 18:06")

        (* * declares the formal parameters and the local vars)


    (PROG (fun funType tc type vars)
          (for fps in <programUnit:formalParameterSection programUnit:formalParameterSection#>
             do (for fps1 in fps bind pg do (pg←fps1:parameterGroup)
                                            (type←pg:type:LEXEME)
                                            (type←(CheckForType type NIL 'Declare))
                                            (vars←pg:identifier)
                                            (vars← < ! vars !(for v in vars
                                                                 collect (PACK* v SingleQuote))
                                                     >)
                                            (declareVar vars type 'vc)))
          (for dop in programUnit:block:declareopt when dop:SYNTACTICTYPE='varDeclaration
             do (type←dop:type:LEXEME)
                (type←(CheckForType type NIL 'Declare))
                (declareVar (for v in dop:varDeclarePart collect v:identifier)
                            type 'vc))
          (fun← <programUnit:identifier ! < !!(formalParameters programUnit:formalParameterSection)
                  !(formalParameters programUnit:formalParameterSection#)
                  >>)
          (funType←(if programUnit:unitKind:LEXEME='FUNCTION
                       then programUnit:type:LEXEME
```

```
                else 'ProcedureCall))
        (if programUnit:unitKind:LEXEME='FUNCTION
            then (declareVar <(if programUnit:identifier#
                                  else programUnit:identifier)
                              >
                             funType 'vc))
        (DeclareFun fun funType T)
        (programUnit:identifier←(ExtendName programUnit:identifier CurrentType))
        (if programUnit:identifier#
            then (programUnit:identifier#←(ExtendName programUnit:identifier# CurrentType)))
        [for fps in <programUnit:formalParameterSection programUnit:formalParameterSection#>
           do (for s in fps do (for i on s:parameterGroup:identifier do (i:1←(typeify i:1]
        (programUnit:identifier##←(for i in programUnit:identifier## collect (typeify i])
```

6

```
(skipPascalComment
  [LAMBDA (commandFile)                                   (* D.Thompson " 2-Sep-80 11:30")
    (until (READC commandFile)=RightCurlyBracket do])
```

7

```
(typecheck
  [LAMBDA NIL                                             (* D.Thompson "15-Sep-79 04:37")
    (for pu in LastProgramUnits do (setupDeclarations pu::1))
    (for pu in LastProgramUnits first (printout NIL .TAB0 0 "Type checking") finally (printout NIL T)
       do (printout NIL . pu:1 "...")
          (typeifyProgramUnit pu::1])
```

8

```
(typeifyProgramUnit
  [LAMBDA (pu)                                            (* Edited by D.Musser on 6-APR-78:
                                                             no file)
                                                          (* Type check and translate the entry and exit
                                                             assertions and the compound statement of the block or
                                                             program unit pu)
    (PROG (blk)
          (blk←pu:block)
          (blk:assertion←(typeify blk:assertion))
          (blk:assertion#←(typeify blk:assertion#))
          (typeifyStatement blk:compoundStatement])
```

9

```
(typeifyStatement
  [LAMBDA (x)                                             (* D.Musser "16-Jul-79 11:40"
                                                          (* Type check and translate the statement x:
                                                             each expression and variable is type checked and its
                                                             type-translation replaces the original in the statement)
    (SELECTQ x:SYNTACTICTYPE
             ((assertStatement assumeStatement proveStatement)
               x:assertion←
               (typeify x:assertion))
             (assignmentStatement x:variable← (typeify x:variable)
                                  x:expression←
                                  (typeify x:expression))
             (caseStatement x:expression← (typeify x:expression)
                            (for y in x:caseElementList do (y:caseLabel←(for z in y:caseLabel
                                                                           collect (typeify z)))
                                                           (typeifyStatement y:statement))
                            (typeifyStatement x:statement))
             (compoundStatement (for y in x:statement do (typeifyStatement y)))
             (concurrentAssignmentStatement x:variable← (for y in x:variable collect (typeify y))
                                            x:expression←
                                            (for y in x:expression collect (typeify y)))
             (forStatement x:assertion← (typeify x:assertion)
                           x:identifier←
                           (typeify x:identifier)
                           x:expression←
                           (typeify x:expression)
                           x:expression#←
                           (typeify x:expression#)
                           (typeifyStatement x:statement)
                           x:assertion#←
                           (typeify x:assertion#))
             (goToStatement x:assertion← (typeify x:assertion))
```

```
            (ifStatement x:expression← (typeify x:expression)
                         (typeifyStatement x:statement)
                         (typeifyStatement x:statement#))
            (labelStatement (typeifyStatement x:simpleStatement))
            [procedureStatement (PROG (ty)
                                      (ty←(typeify <x:identifier ! x:expression ! x:identifier#>))
                                      (if ty
                                          then x:identifier←ty:Operator
                                               x:expression←(for y in x:expression
                                                                 collect (typeify y))
                                               x:identifier#←(for y in x:identifier#
                                                                  collect (typeify y))
                                               x:variable←(for y in x:variable
                                                                  collect (typeify y]
            (repeatStatement (for y in x:statement do (typeifyStatement y))
                             x:expression←
                             (typeify x:expression)
                             x:assertion←
                             (typeify x:assertion))
            (returnStatement x:expression← (typeify x:expression)
                             x:assertion←
                             (typeify x:assertion))
            (whileStatement x:assertion← (typeify x:assertion)
                            x:expression←
                            (typeify x:expression)
                            (typeifyStatement x:statement)
                            x:assertion#←
                            (typeify x:assertion#))
            x])
)
(DECLARE: DONTCOPY
  (FILEMAP (NIL (374 10341 (InitializeAffirmReadTable 386 . 827) (PARSE 831 . 1034) (genvcs 1038 . 3540)
 (readp 3544 . 4377) (setupDeclarations 4381 . 6432) (skipPascalComment 6436 . 6619) (typecheck 6623 .
 7024) (typeifyProgramUnit 7028 . 7597) (typeifyStatement 7601 . 10338)))))
STOP
```