```
(FILECREATED "26-Sep-81 18:12:33" <AFFIRM>THEOREMPROVER..37 43804

     changes to:  denote

     previous date: "10-Jul-81 16:50:10" <AFFIRM>THEOREMPROVER..36)


(PRETTYCOMPRINT THEOREMPROVERCOMS)

(RPAQQ THEOREMPROVERCOMS ((VARS ExpandInductors)
                          (FNS * THEOREMPROVERFNS)
                          (VARS (VariablesInUse NIL)
                                (SkolemFunctions NIL)
                                (ProofBasis NIL)
                                (INDENTATION 2))
                          (PROP INFO choice)
                          (PROP MACRO choice)))

(RPAQQ ExpandInductors NIL)

(RPAQQ THEOREMPROVERFNS (CheckAndSplit ComputeInductionExpression EqualInvoke FLATTEN FindAllDefs
                                       InductionCases MakeAlist MakeAlistError Normalize PickDefs
                                       PickPostiveNumber ShortNamesAndNames SplitIntoSubgoals augment
                                       cases choice choose complete denote employ enter getPrettyNorm
                                       invoke invokeP keep let printvars recheckLemma? replaceCommand
                                       search split suppose swapCommand use))
(DEFINEQ

                                                                                              1

(CheckAndSplit
  [LAMBDA (qex subs)                                    (* R.Erickson "27-Sep-79 17:19")

          (* given a parent Qexpression and a list of (non-qexpr) subgoals. check if their find sets are disjoint.
          If so. set up the subgoals.)


    (PROG (qfind qgiven finds sofar common)
          (qfind←(for f in qex:find collect f:1))           (* extract var. names)
          (qgiven←(for f in qex:given collect f:1))
          (finds←(for s in subs collect (INTERSECTION (FreeVars s)
                                                      qfind)))
          (sofar←NIL)
          (common←(for f in finds bind dup eachtime (PROGN dup←(INTERSECTION sofar f)
                                                           sofar← < !! sofar ! f>)
                      when dup collect dup))
          (if common
              then (AffirmError <"Unable to split; these 'find' variables are used in >1 subgoal:"
                                common>)
            else (RETURN (for s in subs as f in finds collect (create Qexpression
                                                                      expr ← s using qex])

                                                                                              2

(ComputeInductionExpression
  [LAMBDA (expr indVar indVal)                          (* R.Erickson "19-Jun-80 18:15")

          (* * Substitute indVal for indVar in expr, a normalized Qexpression. Equivilant to "Prop(indVal)", where
          "Prop(indVar) = = expr." Called by PROPOP, IH3OP.)


    (PROG (rhs inductvar)
          (DECLARE (SPECVARS inductvar))                    (* used in EVAL, below)
          (CheckForQexpression expr)
          (if (OccursAsOperatorIn indVar expr:expr) or (OccursIn indVar expr:expr)
            else (AffirmError <"induction variable not found" indVar> 'internal))
          (rhs←(create Qexpression
                       given ←(REMOVE <indVar> expr:given)
                       find ←(for sk in expr:find collect (if (NLISTP sk)
                                                              then sk
                                                            else (REMOVE indVar sk)))
                       free ←(<'inductvar >)
                       expr ←(SUBST 'inductvar indVar (SUBST 'inductvar <indVar> expr:expr))
                  using expr))

          (* * rhs of rule. where (QUOTE indvar) replaces indVar. The old employ (with a single IH) didn't bother renaming
          indVar, since it called CompileRuleIntoLisp, which used PROPOP's actual argument name, "a"; since we bind the
          actual-argument value ourselves, we do the renaming here.)
```

```
        (inductvar←indVal)                          (* set value)
        (RETURN (EVAL rhs))                         (* evalute rule)
    ])
```

3

```
(EqualInvoke
  [LAMBDA (Name Expr)                               (* R.Bates "18-Sep-80 09:44")
    (MatchOp Expr:1 Name])
```

4

```
(FLATTEN
  [LAMBDA (x)
    (if (NLISTP x)
        then <x>
      elseif x::1
        then <!!(FLATTEN x:1) !(FLATTEN x::1) >
      else (FLATTEN x:1])
```

5

```
(FindAllDefs
  [LAMBDA (Name Expr Fun)                           (* R.Bates "17-Sep-80 10:34")
    (for x on Expr do (if (APPLY* Fun Name x)
                          then FOUND← < !! FOUND x>)
                      (FindAllDefs Name x:1 Fun])
```

6

```
(InductionCases
  [LAMBDA (type)
    (GETPROP (Name type)
             'InductionCases])
```

7

```
(MakeAlist
  [LAMBDA (ex)                                      (* D.Musser " 7-Aug-79 22:52")
    (if ex:Operator:EQOP
        then <<ex:Arg1 ! ex:Arg2>>
      elseif ex:Operator=ANDOP
        then <!!(MakeAlist ex:Arg1) !(MakeAlist ex:Arg2) >
      else (MakeAlistError ex])
```

8

```
(MakeAlistError
  [LAMBDA (ex)
    (PRINTLINES T "*** incorrect substitution " (INFIX\PRINT3 ex)
                T)
    (ERROR!])
```

9

```
(Normalize
  [LAMBDA (expr normalizeCommand)                   (* R.Erickson "16-Sep-80 14:48")

        (* * normalize expr, print it out, and do all the other stuff formerly done by CevalHelper. expr = NIL-> do nothing;
        else = T-> use CurrentPropn)


    (if expr=T
        then expr←CurrentPropn)
    (if expr
        then (PROG ((result expr))
                   (ZapSkolemFunctions)
                   (if LessOutputDesired
                     else (PrettyPrint expr)
                          (printout NIL T "Normalization:"))
                   (if normalizeCommand and (type? Qexpression result)
                       then                         (* try to fix a normalization problem)
                            result←(SkolemizeProperly result))
```

```
        (result←(TreatFreeAsGiven (EVAL result)))
        (if (type? Qexpression result) and result:expr=TRUE
            then result←TRUE)
        (/SET 'CurrentPropn result)                    (* T->saves PrettyNorm)
        (if result=TRUE
            then                                       (* normalized to TRUE)
                 (PrettyPrint result NIL T)
                 (if normalizeCommand
                     then                              (* record success of explicit command)
                          (Transform (create Transformation
                                             command ←('normalize)
                                             children ←(<TRUE>)))
                   else                                (* implicit)
                        (TransformToTrue)))
        (ReferencedInterfaces←(Operators CurrentPropn))
        (ShortenedReferencedInterfaces←(Shorten ReferencedInterfaces))
        (RETURN CurrentPropn])
```

10

```
(PickDefs
  [LAMBDA (Range Expr ErrorExpr)                       (* R.Bates "19-Sep-80 08:53")
    (PROG (startNumber endNumber)
          (if Range:2=NIL
              then Range←Range:1
                   (RETURN (if Range='ALL
                               then Expr
                             else <(CAR (NTH Expr (PickPostiveNumber Range Expr ErrorExpr)))
                                  >)))
          (startNumber←(PickPostiveNumber Range:1 Expr ErrorExpr))
          (endNumber←(PickPostiveNumber Range:2 Expr ErrorExpr))
          (if endNumber lt startNumber
              then (printout NIL "Didn't select any element with " Range:1 " : " Range:2)
                   (AffirmError))
          (RETURN (for i from startNumber to endNumber collect (NTH Expr i):1])
```

11

```
(PickPostiveNumber
  [LAMBDA (Range Expr ErrorExpr)                       (* R.Bates "19-Sep-80 12:25")
    (PROG ((LengthExpr (LENGTH Expr)))
          (RETURN (SELECTQ Range
                           (LAST LengthExpr)
                           (FIRST 1)
                           (PROGN (if (NUMBERP Range)
                                      then (if (MINUSP Range)
                                               then (if LengthExpr lt (-Range)
                                                        then (printout NIL .TAB0 0 "There "
                                                                   (Plural 'are LengthExpr)
                                                                   " only " LengthExpr " "
                                                                   (Plural 'occurrences
                                                                           LengthExpr)
                                                                   " of " #
                                                                   (PrettyPrint ErrorExpr T)
                                                                   T)
                                                             (AffirmError))
                                                   LengthExpr+Range+1
                                             else (if (ILEQ Range LengthExpr)
                                                      then Range
                                                    else (printout NIL .TAB0 0 "There "
                                                               (Plural 'are LengthExpr)
                                                               " only " LengthExpr " "
                                                               (Plural 'occurrences LengthExpr)
                                                               " of " # (PrettyPrint ErrorExpr T)
                                                               T)
                                                         (AffirmError)))
                                  else (printout NIL .TAB0 0 "Can not figure out: " Range T)
                                       (AffirmError])
```

12

```
(ShortNamesAndNames
  [LAMBDA (ex)
    (if ex=NIL
        then NIL
      elseif (LITATOM ex)
        then <<(Shorten ex) ! ex>>
      elseif (NLISTP ex)
```

```
    then NIL
  else (for x in ex join (ShortNamesAndNames x])
```

13

```
(SplitIntoSubgoals
  [LAMBDA (p qex)                                   (* R.Erickson "21-Feb-80 11:45")

        (* * Given a proposition and its Qexpression environment, return a list of (nonquantified) subgoals whose
        conjunction implies p. Must ensure that, if >1 subgoal is produced, they meet restrictions on free variables.)


    (if qex=NIL
        then (if (type? Qexpression p)
                 then                                (* make single arg permissible)
                      qex←p
                      p←p:expr
               else (AffirmError "missing arg to SplitIntoSubgoals!" 'internal)))
    (if p:Operator=IFOP
        then (if p:ElsePart=TRUE
                 then (choice (finds p:Test qex)
                              <p>
                              (for s in (SplitIntoSubgoals p:ThenPart qex)
                                 collect <IFOP p:Test s TRUE>)
                              p)
               elseif p:ElsePart=FALSE
                 then (if p:ThenPart=TRUE
                          then                       (* (if p T F))
                               <p>
                        else                         (* conjunction. Pull out the text, but remember to embed
                                                     in IFOP, since we don't allow naked predicates.)
                             (choice (finds p:Test qex)
                                     <p> <<IFOP p:Test TRUE FALSE>
                                           !(for s in (SplitIntoSubgoals p:ThenPart qex)
                                               collect <IFOP p:Test s TRUE>)
                                     >
                                     p))
               elseif p:ThenPart=TRUE
                 then (choice (finds p:Test qex)
                              <p>
                              (for s in (SplitIntoSubgoals p:ElsePart qex)
                                 collect <IFOP p:Test TRUE s>)
                              p)
               elseif p:ThenPart=FALSE
                 then (choice (finds p:Test qex)
                              <p> <<IFOP p:Test FALSE TRUE> !(for s in (SplitIntoSubgoals p:ElsePart
                                                                                          qex)
                                                                  collect <IFOP p:Test TRUE s>)
                                  >
                              p)
               else (choice < !!(finds p:Test qex) !(INTERSECTION (FreeVars p:Test)
                                                                  (InvalidDependencies
                                                                    (INTERSECTION (finds p:ThenPart
                                                                                         qex)
                                                                                  (finds p:ElsePart
                                                                                         qex))
                                                                    qex))
                            >
                           <p> <<IFOP p:Test p:ThenPart TRUE> <IFOP p:Test TRUE p:ElsePart>> p))
      else <p>])
```

14

```
(augment
  [LAMBDA (ex)                                      (* R.Erickson "16-Sep-80 14:52")

        (* * This routine provides a capability like use and suppose, but is more specialized. H imp C goesto (H and A imp
        C, H imp A). Quantification is like suppose -
        An AFFIRM command function.)


    (PROG (impform children trans)
          (CheckForQexpression CurrentPropn)        (* any free vars must be given)
          (ex←(EVAL ex))
          (CheckIntroducedExpr ex CurrentPropn 'given)
          (impform←(RemoveIfs CurrentPropn:expr T))
          [children←(for g in <<IFOP ex (TopLevelIf CurrentPropn:expr)
                                     TRUE>
                                     (if impform:Operator=IMPOP
```

```
                                then <IFOP impform:Arg1 (TopLevelIf ex)
                                       TRUE>
                                else ex)
                              >
                        collect (SkolemizeProperly (EVAL (create Qexpression
                                                                  expr ← g using CurrentPropn]
        [trans←(create Transformation
                       command ←('augment)
                       parameters ←(<ex>)
                       children ← children
                       labels ←('(main: thesis:]
        (RETURN (Descend (Transform trans])
```

15

(cases

```
  [LAMBDA NIL                                          (* D.Thompson " 4-Sep-80 15:53")

        (* * This routine uses the embedded if case analysis rule to rearrange the internal form of the current proposition.
        -
        An AFFIRM command function.)


    (PROG (ContinuousEval trans result)
          (DECLARE: (SPECVARS ContinuousEval))
          (CheckForQexpression CurrentPropn)
          (result←(create Qexpression
                          expr ←(RaiseIfs (TopLevelIf CurrentPropn:expr)) using CurrentPropn))
          (if (EQUAL result CurrentPropn)
              then (if InAutoMechanism
                       else (printout NIL .TAB0 0 "Cases had no effect." T))
                   (RETURN NIL)
            else trans←(create Transformation
                               command ←('cases)
                               children ←(<result>))
                 (RETURN (Descend (Transform trans])
```

16

(choice

```
  [LAMBDA (shouldnil failure success context)          (* Edited by Erickson on 28-AUG-78;
                                                       from version 6)
                                                       (* has a macro. return failure or success, depending on
                                                       whether shouldnil = NIL. fail -> print message + context
                                                       (expression) . binding)

    (if shouldnil
        then (TERPRI)
             (PRIN1 "unable to split up the (sub)expression       ")
             (PrettyPrint context)
             (TERPRI)
             (PRIN1 "because of restrictions involving: ")
             (if (NLISTP shouldnil)
                 then (PRIN1 shouldnil)
               else (printvars shouldnil))
             (TERPRI)
             failure
      else success])
```

17

(choose

```
  [LAMBDA (choice)                                     (* R.Erickson "19-Sep-80 14:44")

        (* * This routine implements the AFFIRM choose command. -
        An AFFIRM command function.)


    (CheckForQexpression CurrentPropn)
    (ChooseChainingsAndNarrowings (SkolemizeProperly CurrentPropn:expr CurrentPropn)
                                  0 choice NIL])
```

18

(complete

```
  [LAMBDA (retrycom)                                   (* R.Erickson "19-Sep-80 17:38")

        (* * This routine attempts to find a contradiction in the current proposition by making all the hypotheses and
```

conclusion temporary rewrite rules and using the rewrite rule machine. -
An AFFIRM command function.)

```
(RESETVARS (ContradictionFound (UsingRuleList T)
                               (RuleList RuleList)
                               (Unchecked NIL)
                               impform hlist conclusion (rulelimit retrycom)
                               (rulecount 0)
                               trans result)
           (CheckForQexpression CurrentPropn)
           (impform←(RemoveIfs (SkolemizeProperly CurrentPropn:expr CurrentPropn)
                               T))
           (if impform:Operator=IMPOP
               then hlist←impform:Arg1
                    conclusion←impform:Arg2
             else conclusion←impform
                  hlist←NIL)
           (printout NIL .TAB0 0 "Using hypotheses and negated conclusion as rewrite rules" T
                     "in an attempt to prove by contradiction:"
                     T)
           (if ~[UNDONLSETQ (Completer (for predicate
                                       in
                                        <(if conclusion:Operator=NOTOP
                                             then conclusion:Arg1
                                            elseif conclusion:Operator=NEOP
                                              then <EQVOP ! conclusion::1>
                                             else <EQVOP conclusion FALSE>)
                                          !(if hlist
                                               then (ListOfConjuncts hlist))
                                        >
                                       collect (if predicate:Operator:EQOP
                                                   then predicate
                                                  elseif predicate:Operator=NOTOP
                                                   then <EQVOP predicate:Arg1 FALSE>
                                                  else <EQVOP predicate TRUE>]
              then (if ContradictionFound
                       then result←TRUE
                    else                                    (* aborted)
                         (ERROR!))
             else (PROG ((avoid (REMOVEDUPLICATES < ! CurrentPropn:free
                                                  !(for v in CurrentPropn:given
                                                      join (Frees v))
                                                  !(for v in CurrentPropn:find
                                                      join (Frees v))
                                                  >))
                         newrulevars)
                        (conclusion←FALSE)
                        (hlist←(for pair in RuleList bind r eachtime r←pair:1
                                  when (if r:RHS=FALSE
                                           then conclusion←(IfThenElse (rulequan <r:LHS>
                                                                                 CurrentPropn):1
                                                                       TRUE conclusion)
                                                NIL
                                         else T)
                                  collect (if r:RHS=TRUE
                                              then r:LHS
                                             else r)))
                        (for h in (rulequan hlist) do conclusion←(IfThenElse h conclusion TRUE))
                        (result←(create Qexpression
                                        expr ← conclusion
                                        find ←(< ! CurrentPropn:find ! newrulevars>)
                                 using CurrentPropn))
                                                    (* new vars from rules are find)
                   ))
           (trans←(create Transformation
                          command ←('complete)
                          children ←(<result>)))
           (RETURN (Descend (Transform trans)
                            T])
```



```
(denote
  [LAMBDA (pairs)                                   (* R.Erickson "24-Jun-81 19:29")
```

(* Command function. Pairs is a list of elements "(expr var)"; for each pair we add the hypothesis "expr = var" and replace occurances of expr in the proposition with var. Later pairs may refer to earlier variables.*
Restrictions: -
all free vars must be in proposition or earlier vars -

```
    var doesn't occur in expr -
    var is declared or renamed if necessary, must be of compatible type.)


(PROG (equations newBody goal trans)
      (CheckForQexpression CurrentPropn)

      (* * Do type and freevar checking for each pair, collecting the translated, expanded, renamed versions.)


      (equations←(for pair in pairs bind (sofar frees exp var trVar trExp)
                    collect                                  (* pair is "(expr var)" -
                                                               sofar is vars of earlier pairs)
                            (exp←pair:1)
                            (var←pair:Arg1)

      (* * We do Translates so we get both :Expression and :Type.)


                            (if trExp←(Translate exp)
                              else                           (* interface error; generate error message and abort)
                                   (pexec exp T)
                                   (SHOULDNT))
                            (if trVar←(TranslateLitAtom var T)
                                then                         (* var already declared; check type)
                                     (if trVar:Type~=trExp:Type
                                         then (AffirmError <"Denote type mismatch:" '< trExp:Type
                                                                trVar:Type '> >))
                              else                           (* auto declare)
                                   (declareVar <var> trExp:Type 'internal)
                                   trVar←(Translate var)
                                                             (* so we have extended form))
                            (frees←(CheckIntroducedExpr trExp:Expression CurrentPropn NIL sofar))
                                                             (* free vars must be found in CP or earlier pairs)
                            (if trVar:Expression MEMB frees
                                then (AffirmError <"Denote variable occurs in expression:"
                                                  (Shorten trVar:expression)
                                                  >))
                            (var←trVar:Expression)
                            (var←(NewSymbolFrom var < ! sofar !(Variables CurrentPropn)
                                                    >))
                            (push sofar trVar:Expression)
                                                             (* user can't anticipate renaming)

      (* * We collect (LIST ▪ expr var))


                            (<(GetEqualOp trExp:Type)
                              (SkolemizeProperly trExp:Expression CurrentPropn)
                              var>)))

      (* * Use each equation to reduce later equations so we can do them sequentially.)


      (for tail on equations bind eqn do (eqn←tail:1)
                                         (tail::1←(for tailEq in tail::1
                                                     collect
                                                      <tailEq:1 (SUBST eqn:Arg2 eqn:Arg1
                                                                       tailEq:Arg1)
                                                        tailEq:Arg2>)))

      (* * We want the equational hypotheses to appear in the order given, but need for earlier substitutions to be done
      first. So we first SUBST newBody, then build up the hypotheses and ALLOPs. We let ALLOP take care of Skolemization
      issues)


      (newBody←(SkolemizeProperly CurrentPropn:expr CurrentPropn))
      (for eqn in equations do newBody←(SUBST eqn:Arg2 eqn:Arg1 newBody))
      [for eqn in (REVERSE equations) do                     (* "all fa(f(a) ▪ fa imp newbody")
                                         (newBody←(create Expression
                                                          Operator ← ALLOP
                                                          Arguments ←(<eqn:Arg2
                                                            <IFOP eqn newBody TRUE>>]
      (goal←(create Qexpression
                    expr ← newBody using CurrentPropn))
      (trans←(create Transformation
                     command ←('denote)
                     parameters ←(Separate (for eqn in equations collect (<eqn:Arg1 'by eqn:Arg2>)
                                           )
                                  T)
                     children ←(<goal>)))
```

```
        (RETURN (Descend (Transform trans])
```

20

```
(employ
  [LAMBDA (schema)                                    (* D.Thompson " 4-Sep-80 15:04")

        (* * This routine uses the induction schema provided as its parameter to split the current proposition into the
        cases indicated. -
        An AFFIRM command function.)


    (PROG (IndTarget IndVar ExpandInductors cases labels children trans)
          (DECLARE (SPECVARS IndTarget ExpandInductors indVar))

          (* globals used by Prop, IH. IndTarget is the node# at which employ is done. ExpandInductors turns on the rules;
          when off it allows "cases" to be expanded first.)


          (CheckForQexpression CurrentPropn)
          (IndVar←schema:Arg1)
          (if ~(MEMBER <IndVar> CurrentPropn:given)
              then                                    (* insist that var be given, w/ no dependencies.
                                                      (?))
                   (AffirmError <"Illegal induction variable" (Shorten IndVar)
                                "." >))
          (IndTarget←CurrentNode:prop#)
          (cases←(EVAL schema))
          (ExpandInductors←T)
          (labels←(SchemaCaseNames cases))
          (children←(for s in (if cases:Operator='cases\Schema
                                  then cases:Arguments
                                else (LIST cases))
                       as l in labels bind p collect (p←(EVAL s))
                                                      (* evaluate the case, with Prop, IH active)
                                                     (printout NIL .TAB0 0 "Case" , l , #
                                                               (PrettyPrint s T)
                                                               .
                                                               (if p=TRUE
                                                                   then "proven."
                                                                 else "remains to be shown.")
                                                               T)
                                                     p))
          (trans←(create Transformation
                         command ←('employ)
                         parameters ←(<schema>)
                         children ← children
                         labels ← labels))
          (RETURN (Descend (Transform trans])
```

21

```
(enter
  [LAMBDA (listnames)                                 (* R.Bates "13-Feb-80 16:26")

        (* * This routine enters the propositions contained on each of the groups of the list of groups provided as the
        parameter into the proof forest. The status of each proposition is also set according to the value saved when the
        proposition was added to the group. -
        An AFFIRM command function.)


    (AffirmError "***** This command has problems, please do not use" 'mild)
    (for z in listnames do (for x in (EVAL z) bind node do (if ~(type? groupmember x)
                                                               then (AffirmError
                                                                      <z "isn't a group name!" >))
                                                           (node←(ExprToNode x:grpmprop))
                                                           (MakeTheorem node)
                                                           (if ~(x : grpmstat) or x:grpmstat MEMB '(
                                                                                        proved assumed)
                                                               then (Assume node))
                                                           (if x:grpmname
                                                               then (SetName x:grpmname node])
```

22

```
(getPrettyNorm
  [LAMBDA NIL                                         (* D.Thompson " 9-Mar-80 18:33")
                                                      (* PrettyNorm = either NIL or CONS
```

```
                                        (CurrentPropn (RemoveIfs CurrentPropn)). Nobody should
                                        clobber CurrentPropn so its eq to old value!)
    (if PrettyNorm:1~=CurrentPropn
        then (/SET 'PrettyNorm <CurrentPropn !(RemoveIfs CurrentPropn)
                               >))
    PrettyNorm::1])
```

23

```
(invoke
  [LAMBDA (defnRangePairs)                                    (* R.Erickson " 2-Feb-81 18:50")

          (* The algorithm for the invoke command is: (1) for each thing you want to invoke do a tree walk, gather up the
          instances. (2) Filter out which occurences the user ask for and sort the remaining list by count.
          (3) Invoke each occurence using destructive functions. The reason for the sort is so that inner instances of a given
          function are expanded before outer ones.)

                                                                (* A command function.)
    (if defnRangePairs
        then (PROG ((pairs (defnRangePairs:range))
                    interfaces defnToExpand result trans PropnList answer missing FOUND)
                   (CheckForQexpression CurrentPropn)
                   (interfaces←(Shorten (LDIFFERENCE (Operators CurrentPropn:expr)
                                                     <IFOP !(for var in CurrentPropn:given
                                                                 collect (firstElement var))
                                                       >)))
                                                                (* Possible things to invoke.)

          (* * check that we always have a function target)


                   (for pair in pairs unless pair:1 do (CannedMessage 'invokeNoFn T
                                                                      <(DefnRangeConvert <pair>)
                                                                        >))

          (* * Spelling correct names)


                   (pairs←(for pair in pairs bind fn when fn←(AFFIRMSpellingCorrect pair:1 interfaces)
                             collect                            (* If the user responds "none" to spelling correct. we
                                                                just ignore that parm.)
                                      (<fn pair:2>)))
                   (result← <(COPY (getPrettyNorm))
                            >)                                  (* operate on pretty form, so ordinals are right.
                                                                The copy gets smashed)
                   (PropnList←(for defn in pairs collect (FOUND←NIL)
                                                         (FindAllDefs defn:1 result
                                                                      (if (LISTP defn:1)
                                                                          then
                                                                           (FUNCTION EqualInvoke)
                                                                        else (FUNCTION invokeP)))
                                                         FOUND))
                   (missing←(for x in pairs as y in PropnList collect (Shorten x:1) when y=NIL))
                   (if missing
                       then (AFFIRMMAPRINT "Can not find" missing T " in the current expression.")
                            (AffirmError))
                   [defnToExpand←(for defnRange in pairs as propnList in PropnList
                                    join (for def in defnRange:2 or '((ALL NIL))
                                              join                  (* default is all occs)
                                                   (PickDefs def propnList defnRange:1]
                   (if answer←(for x on defnToExpand thereis (for y on x::1 thereis x:1=y:1))
                       then (printout NIL .TAB0 0 "Can not invoke " # (PrettyPrint answer:1:1 T)
                                      " twice !!" T)
                            (AffirmError))
                   [defnToExpand←(SORT defnToExpand (FUNCTION (LAMBDA (A B)
                                                              ((COUNT A:1)
                                                               lt
                                                               (COUNT B:1]
                   (for expr in defnToExpand do (answer←(ExpandVisibleDefs <expr:1:Operator> expr:1 T)
                                                        )
                                                (if (EQUAL answer expr:1)
                                                    then (printout NIL .TAB0 0
                                                                   "invoke had no effect on "
                                                                   (Shorten expr:1:Operator)
                                                                   T)
                                                  else (expr:1←answer)))
                   (trans←(create Transformation
                                  command ←('invoke)
                                  parameters ←(DefnRangeConvert pairs)
                                  children ← result))
```

```
                (RETURN (if (EQUAL result (getPrettyNorm))
                            then (printout NIL .TAB0 0 "invoke had no effect." T)
                                 NIL
                          else (Descend (Transform trans])
```

24

```
(invokeP
  [LAMBDA (What Expr)                                    (* R.Bates "15-Sep-80 09:44")
    (AND (LISTP Expr:1)
         (Shorten Expr:1:Operator)=What])
```

25

```
(keep
  [LAMBDA (groupName elementList)                        (* R.Bates "13-Feb-80 16:26")

          (* * This routine adds the propositions in the list provided as the second parameter to the group indicated by the
          first parameter. -
          An AFFIRM command function.)


    (AffirmError "***** This command has problems, please do not use" 'mild)
    (/SET groupName (for x in elementList bind xval node
                       join (if (LITATOM x) and xval←(EVALV x)
                                            ~='NOBIND
                                and (LISTP xval) and (for m in xval always (type? groupmember m))
                            then                                (* a preexisting group)
                                 xval
                          else node←(GetNode x)
                               <(create groupmember
                                        grpmprop ←(GetExpression node:prop#)
                                        grpmname ←(GetName node)
                                        grpmannotation ← node:annotation
                                        grpmstat ←(NodeToThm node):status)
                                >)))
    (/SET (PACK* (U-CASE groupName)
                 'COMS)
          <<'VARS groupName>>])
```

26

```
(let
  [LAMBDA (subs dontKeepVariables Command)               (* R.Bates " 4-Sep-80 15:36")

          (* * This routine performs instantiations of existential quantifiers, according to the instantiations provided by
          the first parameter. The second parameter indicates whether the existential quantifiers are to be retained for
          further instantiation. -
          An AFFIRM command function.)


    (PROG (alist circl ok result trans)
          (CheckForQexpression CurrentPropn)
          (if (NUMBERP subs)
              then trans←(GetTransformation subs)
                   (if trans and trans:command MEMB '(let put)
                       then subs←trans:parameters:1
                     else (AffirmError <subs "is not an instantiation." >)))
          (alist←(MakeAlist subs))
          (if circl←(CircularSubs alist)
              then (printout NIL .TAB0 0 "Can't make the substitutions" .PPVTL
                             (for a in alist collect <(Shorten a:1)
                                                      "="
                                                      (Shorten a::1)
                                                      >)
                             #
                             (AFFIRMMAPRINT "because of circularity involving"
                                            (for c in circl collect (Shorten c))
                                            T T))
                   (AffirmError))
          (ok←T)
          [for tail on alist bind (CPfinds CPgivens excess pair)
             first (CPfinds←(for x in CurrentPropn:find collect (firstElement x)))
                   (CPgivens←(for x in CurrentPropn:given collect (firstElement x)))
             eachtime pair←tail:1 do                       (* a bad var?)
                              (if pair:1 MEMB CPfinds
                                  then CPfinds←(REMOVE pair:1 CPfinds)
                                else                       (* not FIND)
```

```
                                    ok←NIL
                                    (printout NIL .TAB0 0 (Shorten pair:1)
                                                , "doesn't appear in the find list." T))
                                (if (FASSOC pair:1 tail::1)
                                    then                (* var appears twice in the list.)
                                         ok←NIL
                                         (printcut NIL .TAB0 0 (Shorten pair:1)
                                                     , "is to be instantiated TWICE?" T))
                                (if excess←(LDIFFERENCE (Frees pair::1)
                                                        < ! CPfinds ! CPgivens>)
                                    then                (* some vars in the proposed instantiation are free in
                                                        outer context.)
                                         ok←NIL
                                         (AFFIRMMAPRINT (CONCAT "The " (Plural 'variables excess)
                                                                )
                                                        (Shorten excess)
                                                        T
                                                        (CONCAT Blank (if (EQLENGTH excess 1)
                                                                          then 'is
                                                                        else 'are)

                                                       "n't bound in the current proposition."]
        (if ok
            then result←(if dontKeepVariables
                             then (create Qexpression
                                          find ←(for v in CurrentPropn:find
                                                   unless (FASSOC v:1 alist) collect v)
                                          expr ←(Instance alist CurrentPropn) using CurrentPropn)
                         else (create Qexpression
                                      expr ←(<IFOP (Instance NIL CurrentPropn)
                                                   TRUE
                                                   (Instance alist CurrentPropn)
                                                   >)
                                      using CurrentPropn))
                 trans←(create Transformation
                               command ←(if dontKeepVariables
                                            then 'put
                                          else 'let)
                               parameters ←(<subs>)
                               children ←(<result>))
                 (if Command
                     then (Annotate Command trans))
                 (RETURN (Descend (Transform trans)))
          else (AffirmError])
```

27

```
(printvars
  [LAMBDA (x)
    (for e in x do (PRINTLINES (if (LITATOM e)
                                   then (Shorten e)
                                 else e)
                               "  "])
```

28

```
(recheckLemma?
  [LAMBDA (lemno thno)                  (* Edited by Erickson on 16-AUG-78;
                                        from version 1)
                                        (* if desired,we run thru subgoals numerically higher ->
                                        proven later)
    (if RECHECKLEMMAS
        then lemno gt thno
      else NIL])
```

29

```
(replaceCommand
  [LAMBDA (parms)                       (* R.Erickson "19-Sep-80 17:13")

        (* * This routine implements the theorem prover command REPLACE, which causes substitutions to be made in the
        current proposition, based on the equalities that occur in the hypotheses. Form is REPLACE: or REPLACE a,b,c,...;
        where the arguments form the ReplaceList used to control exactly which substitutions get made.
        If the ReplaceList is empty, all equality hypotheses are used to make RHS for LHS substitutions.
        If ReplaceList is not empty, and LHS = RHS is a hypothesis to an implication, then if LHS occurs in ReplaceList, the
        substitution RHS for LHS is made; otherwise, if RHS occurs in ReplaceList, the substitution LHS for RHS is made;
        otherwise neither substitution is made. -
        An AFFIRM command function.)
```

```
    (PROG (ReplaceList (result CurrentPropn))
          (CheckForQexpression CurrentPropn)
          (ReplaceList←(for r in parms collect (SkolemizeProperly r result)))
          (result←(SkolemizeProperly result))
          (result←(create Qexpression
                          expr ←(ApplyAllEqHyps (RemoveIfs result:expr T)) using result))
          (if (EQUAL result CurrentPropn) or (EQUAL result←(EVAL result)
                                                    CurrentPropn)
              then (if InAutoMechanism
                       else (printout NIL .TAB0 0 "Replace had no effect." T))
                   (RETURN NIL)
            else (RETURN (Descend (Transform (create Transformation
                                                    command ←('replace)
                                                    parameters ← parms
                                                    children ←(<result>))
                                             NIL 'replace])
```

30

(search

```
  [LAMBDA NIL                                          (* R.Erickson "19-Sep-80 14:43")

          (* * This routine uses the Chaining and Narrowing algorithm of Lankford and Musser to find a set of instantiations
          of the existential quantifiers of the current proposition that will reduce the proposition to TRUE.
          -
          An AFFIRM command function.)


    (CheckForQexpression CurrentPropn)
    (if (TryChainingsAndNarrowings (SkolemizeProperly CurrentPropn:expr CurrentPropn)
                                   0)
      else (printout NIL .TAB0 0 "Unsuccessful." T)
           NIL])
```

31

(split

```
  [LAMBDA NIL                                          (* D.Thompson " 5-Sep-80 14:20")

          (* * This routine splits the current proposition into two or more subsidiary propositions, as is documented in the
          AFFIRM reference manual. -
          An AFFIRM command function.)


    (PROG (PN PE children trans)
          (RESETVARS ((LessOutputDesired T))
                     (Normalize (RemoveIfs CurrentPropn)))
          (PN←(getPrettyNorm))
          (if (NLISTP PN)
              then (printout NIL .TAB0 0 "Can't split" , # (PrettyPrint PN T)
                             T)
                   (RETURN NIL))
          (PE←(if PN:Operator=QOP
                  then PN:expr
                else PN))
          (if PE:Operator=ANDOP
              then                                     (* user sees conjunction. Use that to split.)
                   children←(CheckAndSplit PN PE:Arguments)
            else                                       (* split using regular if-then-else form.)
                 children←(MakeQexpressions (SplitIntoSubgoals CurrentPropn)
                                            CurrentPropn))
          [trans←(create Transformation
                         command ←('split)
                         children ← children
                         labels ←(if children:1 and ~(children : : 8)
                                     then                  (* label the children if possible)
                                          (to (FLENGTH children) as l
                                             in '(first: second: third: fourth: fifth: sixth:
                                                         seventh: eighth:)
                                             collect l]
          (RETURN (Descend (Transform trans])
```

32

(suppose

```
  [LAMBDA (supp)                                       (* R.Erickson "17-Sep-80 13:46")
```

```
(* * This routine splits the current proposition into two propositions, based on the supposition and contradiction
of the proposition supplied as the parameter. -
An AFFIRM command function.)


(PROG (p children trans)
      (CheckForQexpression CurrentPropn)
      (CheckIntroducedExpr supp CurrentPropn)
      (supp←(SkolemizeProperly supp CurrentPropn))

(* * ensure embedded item has ifthenelse)


      (p←(SkolemizeProperly (TopLevelIf CurrentPropn)))
      (children←(MakeQexpressions (SplitIntoSubgoals <IFOP supp p:expr p:expr> CurrentPropn)
                                  CurrentPropn))
      [trans←(create Transformation
                     command ←('suppose)
                     parameters ←(<supp>)
                     children ← children
                     labels ←('(yes: no:]                    (* what if ~2 children?)
      (RETURN (Descend (Transform trans])
```

33

```
(swapCommand
  [LAMBDA (parsedRangePairs parameters)                      (* R.Erickson "19-Sep-80 11:18")

(* * Command function. Reverses selected equalities. They may be specified by ordinal number, possibly giving one of
the arguments to equal.)


   (PROG (ScanFilter Occurances targets [result (COPY (SkolemizeProperly (getPrettyNorm]
                trans)
         (DECLARE: (SPECVARS ScanFilter Occurances))
         (parsedRangePairs←parsedRangePairs:range)           (* format of :range is "(...(op ((begin end)...))...)"
                                                             where end is NIL if omitted.)
         (for rangedOp in parsedRangePairs
            do

(* * compute the targets of swapping for all ranged pairs.)


               (rangedOp:1 or rangedOp:2 or (AffirmError "Swap what?"))
               (ScanFilter←(if rangedOp:1
                               then (SkolemizeProperly (pexec rangedOp:1 T)
                                                       result)))
               (Occurances←NIL)                                  (* set Occurances to all equalities one of whose
                                                                 arguments equals ScanFilter. Occurances will be formed
                                                                 reversed.)
               (MapExpr result [FUNCTION (LAMBDA (x)             (* is it an equality? If filtering, does it match?)
                           (if x:Operator:EQOP and (~ScanFilter or (EQUAL ScanFilter x:Arg1)
                                                                   or (EQUAL ScanFilter x:Arg2))
                               then (push Occurances x]
                           T)
               (if ~Occurances
                   then                                          (* no hits at all)
                        (if ScanFilter
                            then (AffirmError <"Can't find an equality with" (Shorten ScanFilter)
                                              >)
                          else (AffirmError "Can't find any equalities.")))
               (Occurances←(DREVERSE Occurances))
               (if rangedOp:2
                   then                                          (* do ordinal filtering. targets is where we collect
                                                                 what to swap. PickDefs does own errors, never returns
                                                                 NIL.)
                        [pushlist targets (for range in rangedOp:2
                                             join (PickDefs range Occurances
                                                            (OR rangedOp:1 'equalities]
                 else                                          (* no ordinal: do all)
                      (pushlist targets Occurances)))
               (for tail on targets do                         (* check for duplicates)
                                (if tail:1 MEMB tail::1
                                    then (AffirmError <"Can't swap twice:" (Shorten tail:1)
                                                      >)))

(* * do the actual swapping)
```

```
        (for targ in targets do (swap targ:Arg1 targ:Arg2))
                                                        (* this is why we copied)
        (trans←(create Transformation
                       command ←('swap)
                       parameters ← parameters
                       children ←(<result>)))
        (RETURN (Descend (Transform trans])
```

34

```
(use
  [LAMBDA (node applyCommand)                           (* R.Erickson "17-Nov-80 14:37")

        (* * This routine performs the "use lemma" function of AFFIRM. -
        input may be node or name , node -
        An AFFIRM command function.)


    (PROG (Assumption effect id result trans)           (* anything but NIL for current)
        (MakeTheorem node)                              (* we choose: use -> make a thm if not already)
        (Assumption←(NodeToExpr node))                  (* since node may be given by name, etc.)
        (Assumption←(EVAL Assumption))
        (if Assumption=TRUE
            then (AffirmError "Lemma reduces to True!"))
        (if (FIXP id←(NodeId node))
            then                                        (* no name was supplied; invent one.)
                 id←(if CurrentTheorem
                        then (TheoremId CurrentTheorem))
                 (if (FIXP id)
                     then id←NIL)                       (* id is now NIL or a name of parent theorem)
                 id←(NewNodeName 'lemma (if id
                                            then (PACK* 'Of id)))
                 (SetName id node)
                 (printout NIL "Since you didn't supply a name, that's called" , id T))
        (Assumption←(RenameBoundVariables (TreatFreeAsGiven Assumption)
                                          (Variables CurrentPropn)))
        (PrettyPrint (create Qexpression
                             given ← Assumption:find
                             find ← Assumption:given using Assumption))
        (result←(IfThenElse Assumption (TopLevelIf CurrentPropn)
                            TRUE))                      (* normalizes)
        (trans←(create Transformation
                       command ←(if applyCommand
                                    then 'apply
                                  else 'use)
                       parameters ←(NodeId node)
                       uses ←(<node:prop#>)
                       children ←(<result>)))
        (effect←(Descend (Transform trans)
                         T))
        (RETURN (if applyCommand
                    then effect
                  else (RESETVARS ((LessOutputDesired T))
                                  (RETURN (Normalize result])
)

(RPAQ VariablesInUse NIL)

(RPAQ SkolemFunctions NIL)

(RPAQ ProofBasis NIL)

(RPAQ INDENTATION 2)

(PUTPROPS choice INFO EVAL)

(PUTPROPS choice MACRO [x (LIST (QUOTE PROG)
                                (QUOTE (shld))
                                (LIST (QUOTE SETQ)
                                      (QUOTE shld)
                                      (CAR x))
                                (LIST (QUOTE COND)
                                      (LIST (QUOTE shld)
                                            (LIST (QUOTE PRINTLINES)
                                                  T "unable to split up the (sub)expression  " T
                                                  (LIST (QUOTE PrettyPrint)
                                                        (CADDDR x))
                                                  T "because of restrictions involving: ")
                                            [LIST (QUOTE COND)
                                                  (LIST (LIST (QUOTE NLISTP)
```

```
                                        (QUOTE shld))
                                    (LIST (QUOTE PRIN1)
                                        (QUOTE shld)))
                                (LIST T (LIST (QUOTE printvars)
                                            (QUOTE shld]
                            (LIST (QUOTE TERPRI))
                            (LIST (QUOTE RETURN)
                                (CADR x)))
                        (LIST T (LIST (QUOTE RETURN)
                                    (CADDR x])
(DECLARE: DONTCOPY
  (FILEMAP (NIL (950 42873 (CheckAndSplit 962 . 2037) (ComputeInductionExpression 2041 . 3542) (
EqualInvoke 3546 . 3685) (FLATTEN 3689 . 3867) (FindAllDefs 3871 . 4123) (InductionCases 4127 . 4215)
(MakeAlist 4219 . 4534) (MakeAlistError 4538 . 4661) (Normalize 4665 . 6291) (PickDefs 6295 . 7053) (
PickPostiveNumber 7057 . 8210) (ShortNamesAndNames 8214 . 8471) (SplitIntoSubgoals 8475 . 10730) (
augment 10734 . 11872) (cases 11876 . 12757) (choice 12761 . 13510) (choose 13514 . 13891) (complete
13895 . 16821) (denote 16825 . 20771) (employ 20775 . 22662) (enter 22666 . 23681) (getPrettyNorm
23685 . 24175) (invoke 24179 . 27887) (invokeP 27891 . 28059) (keep 28063 . 29100) (let 29104 . 32537)
 (printvars 32541 . 32699) (recheckLemma? 32703 . 33084) (replaceCommand 33088 . 34779) (search 34783
. 35398) (split 35402 . 36936) (suppose 36940 . 37952) (swapCommand 37956 . 40806) (use 40810 . 42870)
))))
STOP
```