<AFFIRM>TREE..17 30-Sep-81 15:48:06

```
(FILECREATED "26-Sep-81 18:15:53" <AFFIRM>TREE..17 88867

     changes to:  SchemaCaseNames

     previous date: "31-Mar-81 19:56:32" <AFFIRM>TREE..16)


(PRETTYCOMPRINT TREECOMS)

(RPAQQ TREECOMS ((COMS (FNS ActualExprAt CompressTransform GetLabel)
                       (VARS (Truenode (create Node prop#←0))
                             (Nodes <Truenode>)
                             (Theorems NIL)
                             (ProofChain NIL)
                             (RecencyCount 1)))
                 (COMS (* nodes)
                       (FNS * NodeFns))
                 (COMS (* naming of propns - the Propositions record takes prop#s to names. The
                          THEOREM property of the name gives the corresponding prop#.)
                       (FNS GetName NewNodeName SetName DecodeName Names ClearNames)
                       (VARS (Prop#sToNames)))
                 (COMS (* nodes and transformations)
                       (FNS * TransFns))
                 (COMS (* theorems)
                       (FNS * TheoremFns))
                 (COMS (* maintenance of thm:factsused and :usedby)
                       (FNS * FactFns))
                 (COMS (* proof states)
                       (FNS * StatusFns)
                       (VARS (ApproximateStatus NIL)))
                 (COMS (* top level user commands and movement)
                       (FNS * TreeUserFns))
                 (COMS (* toplevel structure commands used by theoremprover)
                       (FNS * StructFns))
                 (COMS (* maintenance of parentage (:parents, :factsused, and :leaves))
                       (FNS * ParentageFns)
                       (TEMPLATES MapChildren))
                 (COMS (* Proof Chain queries, ops; movement functions)
                       (FNS PCTopChain PCCurrNode PCMemb Descendant?)
                       (FNS Descend DescendAlongChain AscendTo AscendBy)
                       (FNS ClearProof MakeCurrent))
                 (COMS (* leaves; use proof chains)
                       (FNS * LeafFns))
                 (COMS (* annotations)
                       (FNS Annotate GetAnnotation RestoreAnnotations))
                 (COMS (* printing proof trees)
                       (FNS * PrintFns)
                       (VARS * PrintVars))
                 (COMS (* circularity tests)
                       (FNS * CircleFns))
                 (PROP GLOBALVAR CurrentPropn CurrentNode CurrentTheorem Truenode)))
(DEFINEQ
```

1

```
(ActualExprAt
  [LAMBDA (nodeId)                                    (* R.Erickson "11-Jun-80 21:35")

          (* * this kludge gives you what the normalized expression is below a node; it's needed since theorems aren't stored
          in a normalized form.)


    (PROG (result)
          [result←(TreatFreeAsGiven (EVAL (GetExpression (GetNode# nodeId]
          (if (type? Qexpression result) and result:expr=TRUE
              then result←TRUE)
          (RETURN result])
```

2

```
(CompressTransform
  [LAMBDA (t)                                         (* R.Erickson "16-Aug-79 11:49")
    t:children←(UseNumbers t:children)
    t:uses←(UseNumbers t:uses)
    t])
```

3

```
(GetLabel
  [LAMBDA (child parent)                                    (* R.Erickson " 6-Feb-80 16:34")

          (* * child memb parent:trans:children. Return the corresponding label, if any)


    (PROG (c# trans)
          (c#←(GetNode# child))
          (trans←(GetTransformation parent))
          (RETURN (if trans:labels
                      then (for c in trans:children as l in trans:labels when c=c# do (RETURN l])
)

(RPAQ Truenode (create Node prop#←0))

(RPAQ Nodes <Truenode>)

(RPAQ Theorems NIL)

(RPAQ ProofChain NIL)

(RPAQ RecencyCount 1)
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

```
(* nodes) ]

(RPAQQ NodeFns (GetNode GetNodeP GetNode# ExprToNode ExprToNode# NodeToExpr MakeNode NodeId True?))
(DEFINEQ
```

4

```
(GetNode
  [LAMBDA (nodeid)                                    (* R.Erickson "20-Aug-79 18:55")

          (* * nodeid is node record, prop #, name, or NIL (meaning current) Return the node record.)


    (if (GetNodeP nodeid)
      else (AffirmError <"node unknown" nodeid>])
```

5

```
(GetNodeP
  [LAMBDA (nodeid)                                    (* R.Erickson "20-Aug-79 18:55")

          (* * nodeid is node record, prop #, name, or NIL (meaning current) Return the node record.)


    (PROG (number)
          (RETURN (if ~nodeid
                      then (if CurrentNode
                               else (AffirmError "no current node"))
                    elseif (ATOM nodeid)
                      then number←(if (FIXP nodeid)
                                      then nodeid
                                    else (DecodeName nodeid))
                           (if (FASSOC number Nodes)
                             else (AffirmError <"number unknown:" number>))
                    else nodeid])
```

6

```
(GetNode#
  [LAMBDA (n)                                         (* R.Erickson "12-Sep-79 12:31")
    (if (NUMBERP n)
        then n
      elseif (LITATOM n) and n
        then (DecodeName n)
      else (GetNode n):prop#])
```

7

```
(ExprToNode
  [LAMBDA (x)                                         (* R.Erickson "17-Oct-79 15:29")
    (if (LISTP x)
        then                                          (* translate the expression to the corresponding node)
             [PROG (number)
                   (number←(TranslateTo# x))          (* gives <0 if new)
                   (RETURN (if (MINUSP number)
                               then number←(-number)
                                    (MakeNode number)
                             elseif (FASSOC number Nodes)
                             else (AffirmError <"translated to unknown number" x number> 'internal]
      elseif x=TRUE
        then Truenode
      else (GetNode x])
```

8

```
(ExprToNode#
  [LAMBDA (x)                                         (* R.Erickson "20-Aug-79 18:01")
    (if (NUMBERP x)
        then x
      else (ExprToNode x):prop#])
```

9

```
(NodeToExpr
```

```
  [LAMBDA (node)                                    (* R.Erickson " 4-Sep-79 17:44")
    (GetExpression (GetNode node):prop#])
```

10

```
(MakeNode
  [LAMBDA (pr#)                                     (* R.Erickson " 9-Oct-79 18:13")
    (CAR (ADDTOLIST 'Nodes (create Node
                                   prop# ← pr#])
```

11

```
(NodeId
  [LAMBDA (node)                                    (* R.Erickson " 7-Sep-79 19:50")

          (* * return name, if any, else number)


    (if (GetName node)
      elseif (FIXP node)
        then node
      else (if (NLISTP node)
               then node←(GetNode node))
           node:prop#])
```

12

```
(True?
  [LAMBDA (node#)                                   (* R.Erickson "16-Aug-79 14:51")
                                                    (* given # or node)

          (* * T if this is the node for TRUE)


    (if (LISTP node#)
        then node#←node#:prop#)
    node#=0])
)
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

```
(* naming of propns - the Propositions record takes prop # s to names. The THEOREM property of the
   name gives the corresponding prop # .)  ]

(DEFINEQ

                                                                                                    13

(GetName
  [LAMBDA (n)                                                    (* R.Erickson "11-Sep-79 18:34")
                                                                 (* arg is prop #, node, or theorem.
                                                                 Return name, if any)

    (if (LISTP n)
        then n←n:prop#)
    n:name])

                                                                                                    14

(NewNodeName
  [LAMBDA (prefix suffix)                                        (* R.Erickson "17-Nov-80 13:24")

        (* * Vaguely like GENSYM. Produces an atom like prefix1suffix, starting from 1 and skipping any names which are
        already known. Both args are optional; if both are ommited, we use prefix "L".)


    (OR prefix suffix prefix←"L")
    (for i from 1 bind cand eachtime cand←(PACK < !(MKLIST prefix)
                                                  i !(MKLIST suffix)
                                                  >)
       while cand MEMB KnownNames:Nodes do NIL finally (RETURN cand])

                                                                                                    15

(SetName
  [LAMBDA (name nodeid)                                          (* R.Erickson "27-Oct-80 15:14")

        (* * This routine associates a name with a proposition (proof node).)


    (if ~(LITATOM name)
        then (AffirmError <"Invalid name" name>)
      elseif (EndsColon name)
        then (AffirmError "Node names can't end in colon; that's reserved for arc labels."))
    nodeid←(GetNode# nodeid)
    (if ~(FIXP nodeid)
        then (AffirmError "Nodeid not numeric" 'internal))
    (if (GETPROP name 'THEOREM)
        ~MEMB <NIL nodeid>
        then (printout NIL .TAB0 0 "Resetting name" , name , "from" , (GETPROP name 'THEOREM)
                       Period)
             (/replace name of (GETPROP name 'THEOREM) with NIL))
    (pushnew KnownNames:Nodes name)                              (* not undoable)
    (/PUTPROP name 'THEOREM nodeid)
    (/replace name of nodeid with name])

                                                                                                    16

(DecodeName
  [LAMBDA (name optional)                                        (* R.Erickson "29-Oct-79 20:03")
    (if (GETPROP (AFFIRMSpellingCorrect name KnownNames:Nodes NIL optional)
                 'THEOREM)
      elseif ~optional
        then (AffirmError <"name unknown" name>])

                                                                                                    17

(Names
  [LAMBDA NIL                                                    (* R.Erickson "20-Oct-79 17:12")
    (for assoc in Prop#sToNames when assoc::1 collect assoc::1])

                                                                                                    18

(ClearNames
  [LAMBDA (names)                                                (* R.Erickson "21-Oct-80 17:13")

        (* * Delete the specified names, or all if omitted. Prop #sToNames in an ASSOC list from # to name;
```

 # is on the property list of the name atom.)

```
    (if ~names
        then                                          (* will kill all names)
             names←KnownNames:Nodes)
    (for name in names do (/PUTPROP name 'THEOREM NIL))
    (/SET 'Prop#sToNames (for al in Prop#sToNames unless al::1 MEMB names collect al))
    (/replace Nodes of KnownNames with (LDIFFERENCE KnownNames:Nodes names))
    names])
)

(RPAQ Prop#sToNames NIL)
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

(* nodes and transformations)]

```
(RPAQQ TransFns (GetChild GetChildNames GetTransformation GetOldTransformations
                          CheckTransformationOccurs FlagArcDescent))
(DEFINEQ
```

19

```
(GetChild
  [LAMBDA (label nodeid optional)                    (* R.Bates "23-Jan-80 13:40")

          (* * may be specified by arc label, position, prop# or name -
          returns the prop#)


    (PROG ((node (GetNode nodeid))
           xform index res corrected)
          (xform←node:trans)
          (if (EndsColon label)
              then (if corrected←(AFFIRMSpellingCorrect label KnownNames:Arcs optional)
                       and index←(POS corrected xform:labels)
                           ~=0
                       then res←(FNTH xform:children index):1)
            else res←(if (NUMBERP label)
                         then (FNTH xform:children label):1
                       elseif (ATOM label)
                         then (GetNode# label) MEMB xform:children))
          (if res
              then (RETURN res)
            elseif ~optional
              then (AffirmError <"child not found" (corrected or label)
                                >])
```

20

```
(GetChildNames
  [LAMBDA (nodeid)                                   (* R.Erickson "14-Aug-79 18:04")
    nodeid←(GetNode nodeid)
    (if nodeid:trans
        then nodeid:trans:labels])
```

21

```
(GetTransformation
  [LAMBDA (nodeid)                                   (* R.Erickson "14-Aug-79 18:05")
    (GetNode nodeid):trans])
```

22

```
(GetOldTransformations
  [LAMBDA (nodeid)                                   (* R.Erickson "14-Aug-79 18:04")
    (GetNode nodeid):oldtrans])
```

23

```
(CheckTransformationOccurs
  [LAMBDA (trans node)                               (* R.Erickson "20-Aug-79 16:46")
                                                     (* both are records)
    (OR trans=node:trans (trans MEMB node:oldtrans)
        (AffirmError <"transformation not found for node" trans node:prop#>])
```

24

```
(FlagArcDescent
  [LAMBDA (trans child#)                             (* R.Bates "23-Jan-80 13:40")

          (* * We are about to descend to child. If it is a child of the given transformation, and it has an arc label, print
          it, so the user knows where we are.)


    (PROG (n l)
          (if trans:labels and (0~=n←(POS child# trans:children)) and l←(FNTH trans:labels n):1
              then (printout NIL " (" l ")"])
)
```

```
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

(* theorems)]

```
(RPAQQ TheoremFns (GetTheorem GetTheoremP TheoremId NodeToThm ThmToNode Provees MostRecent
                          SortRecency GetTheorems UndertakeProof Restart))
(DEFINEQ

                                                                                    25

(GetTheorem
  [LAMBDA (node)                                          (* R.Erickson "20-Aug-79 16:37")
    (if (GetTheoremP node)
      else (AffirmError <"nonexistent theorem" node> 'internal])

                                                                                    26

(GetTheoremP
  [LAMBDA (node)                                          (* R.Erickson "19-Oct-79 15:58")

          (* * node = prop# name or a Node record. Return NIL if fail.)


    (if (LISTP node)
        then node←node:prop#
      elseif node and (LITATOM node)
        then node←(DecodeName node))
    (if ~(FIXP node)
        then (AffirmError <"not a number" node> 'internal))
    (FASSOC node Theorems])

                                                                                    27

(TheoremId
  [LAMBDA (thm)                                           (* R.Erickson "11-Sep-79 18:31")
                                                          (* thm is a prop# or record)

          (* * thm must be a theorem. We return name, if any, otherwise number)


    (if (LISTP thm)
        then thm←thm:prop#)
    (if (GetName thm)
      else thm])

                                                                                    28

(NodeToThm
  [LAMBDA (node)                                          (* R.Erickson "20-Aug-79 17:46")
    (GetTheoremP node:prop#])

                                                                                    29

(ThmToNode
  [LAMBDA (thm)                                           (* R.Erickson "20-Aug-79 17:45")
    (GetNode thm:prop#])

                                                                                    30

(Provees
  [LAMBDA (node)                                          (* R.Erickson "20-Aug-79 19:48")

          (* * list of theorems being proved by node, maybe self)


    (PROG (thm)
          (RETURN (if thm←(GetTheoremP node)
                      then <thm>
                    else (for p in node:parents collect (GetTheorem p])

                                                                                    31

(MostRecent
  [LAMBDA (thms)                                          (* R.Erickson "15-Oct-79 16:58")
```

```
        (* * Returns the one most recently attempted, or just the first)

    (PROG (sorted)
          (thms←(for t in thms collect (GetTheorem t)))
          (sorted←(SortRecency thms))
          (RETURN (if sorted:1:recency
                      then sorted:1
                    else thms:1])
```

32

```
(SortRecency
  [LAMBDA (thms)                                    (* R.Erickson "22-Oct-79 18:20")

        (* * given a list of theorems, sort most recent first)

    (SORT (if (type? Theorem thms:1)
              then thms
            else (GetTheorems thms))
          (FUNCTION (LAMBDA (t1 t2)
              (if t1:recency
                  then ~(t2 : recency) or (IGEQ t1:recency t2:recency)
                elseif t2:recency
                  then NIL
                else (ILEQ t1:prop# t2:prop#])
```

33

```
(GetTheorems
  [LAMBDA (ts)                                      (* R.Erickson "15-Oct-79 17:08")
    (for t in ts collect (GetTheorem t])
```

34

```
(UndertakeProof
  [LAMBDA (thm chain)                               (* D.Thompson " 4-Sep-80 16:49")

        (* * enter the proof of a theorem. possibly starting at some chain. thm is either NIL (meaning CurrentTheorem) or a
        thm record. If chain is given, it is a proof chain from thm, which we descend.)

    (PROG (pgm?)
          (if thm
              then (if ~(type? Theorem thm)
                       then (SHOULDNT))
                   (/SET 'CurrentTheorem thm)
                   (/replace recency of thm with RecencyCount)
                   (add RecencyCount 1))
          [if chain
              then (if chain:-1~=CurrentTheorem:prop#
                       then (AffirmError "bad undertake chain" 'internal))
                   (/replace lastchain of CurrentTheorem with chain)
            else                                    (* starting at the theorem)
                 (printout NIL .TAB0 0 (TheoremId thm)
                           .)
                 (if (GetAnnotation (ThmToNode thm)
                                    NIL 'PROGRAM)
                     then (printout NIL "is the root of a program verification.")
                          pgm?←T
                   else (printout NIL "is" , (PrettyStatus thm))
                        (if thm:lastchain::1
                            then (printout NIL , "(and may be resumed)." T)
                          else (printout NIL Period T]
          (/SET 'ProofChain (OR chain <CurrentTheorem:prop#>))
          (MakeCurrent)
          (RETURN (if pgm?
                      then CurrentPropn
                    else                            (* else we avoid printing the dummy propn verification)
                         (Normalize T])
```

35

```
(Restart
  [LAMBDA (thm)                                     (* D.Thompson " 4-Sep-80 16:42")
```

```
(* * enter the proof of thm. If lastchain at an unproven leaf, go there, else at leaf if possible)

    (if ~(type? Theorem thm)
        then (SHOULDNT))
    (if thm:lastchain and ~(True? thm:lastchain:1)
        then (printout NIL .TAB0 0 "Resuming" , (TheoremId thm)
                       Period T)
             (UndertakeProof thm thm:lastchain)
      elseif thm:leaves and thm:leaves:1:1~=thm:prop#
        then                                          (* avoid leaf which is self)
             (printout NIL .TAB0 0 "Going to a leaf of" , (TheoremId thm)
                       Period T)
             (UndertakeProof thm thm:leaves:1)
      else (UndertakeProof thm])
)
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

```
(* maintenance of thm:factsused and :usedby) ]

(RPAQQ FactFns (AddUsage DropUsage Facts FactsApproxP UsedBy addRules dropRules addUsed dropUsed))
(DEFINEQ

                                                                                    36

(AddUsage
  [LAMBDA (facts users)                                    (* R.Erickson "21-Aug-79 19:01")
                                                           (* facts = <thm #'s> users = <theorems>)

          (* * do the bookkeeping to indicate that facts are used under the proof of users)


    (if facts and users
        then (for user in users do (addRules facts user)     (* user depends upon fact))
             (for fact in facts do (addUsed (GetTheorem fact)
                                            users)           (* fact is used by users)
                                   ])

                                                                                    37

(DropUsage
  [LAMBDA (facts users)                                    (* R.Erickson "21-Aug-79 19:06")
    (if facts and users
        then (for user in users do (dropRules facts user))
             (for fact in facts do (dropUsed (GetTheorem fact)
                                             users])

                                                                                    38

(Facts
  [LAMBDA (thm approx)                                     (* R.Erickson "30-Sep-79 12:00")
                                                           (* thm a record)

          (* * return the (corrected) list of facts used by thm. approx->dont recompute)


    (if 'APPROXIMATE ~MEMB thm:factsused
        then thm:factsused
      elseif approx
        then (REMOVE 'APPROXIMATE thm:factsused)
      else                                                 (* recompute)
           (PROG (Facts trans (node (ThmToNode thm)))
                 (DECLARE: (SPECVARS Facts))
                 (if node:trans
                     then Facts←node:trans:uses)
                 [MapChildren thm:prop# (FUNCTION (LAMBDA (node thm)
                                                  (if thm
                                                      then (pushnew Facts thm:prop#)
                                                    elseif node:trans
                                                      then Facts←(UNION Facts node:trans:uses]
                 (/replace factsused of thm with Facts)
                 (RETURN Facts])

                                                                                    39

(FactsApproxP
  [LAMBDA (thm)                                            (* R.Erickson "16-Aug-79 11:28")
    ('APPROXIMATE MEMB thm:factsused])

                                                                                    40

(UsedBy
  [LAMBDA (thm approx)                                     (* R.Erickson "24-Oct-80 13:23")
                                                           (* given a theorem record)

          (* * Returns a list of theorem numbers of all those who are users of this thm. This includes both "use" by a
          transformation and children.)


    (if 'APPROXIMATE ~MEMB thm:usedby
        then thm:usedby
      elseif approx
        then (REMOVE 'APPROXIMATE thm:usedby)
```

```
    else

        (* recompute. Note: we may have been used by nodes which once were theorems, but no longer are.
        (Addition of resultant new users isn't our responsibility.))


         (/replace usedby of thm with (for use in thm:usedby when (use~='APPROXIMATE)
                                                                  and (GetTheoremP use)
                                                                  and thm:prop# MEMB
                                                                      (Facts (GetTheorem use)
                                                                             NIL)
                                                          collect          (* should the Facts be approximate ???)
                                                                  use])
```

41

```
(addRules
  [LAMBDA (rules thm)
    (CLISP: UNDOABLE)                                    (* R.Erickson "14-Aug-79 18:24")
    thm:factsused←(UNION thm:factsused rules])
```

42

```
(dropRules
  [LAMBDA (rules thm)
    (CLISP: UNDOABLE)                                    (* R.Erickson "20-Aug-79 16:59")
                                                         (* remove use of rules under thm)
                                                         (* diagnose?)

    (if rules
        then thm:factsused←(LDIFFERENCE thm:factsused rules)
             (pushnew thm:factsused ('APPROXIMATE])
```

43

```
(addUsed
  [LAMBDA (theorem users)
    (CLISP: UNDOABLE)                                    (* R.Erickson "21-Aug-79 18:53")
                                                         (* thm, list of thms)

        (* * notice that theorem has been used for the proofs of users.)


    theorem:usedby←(UNION (for u in users collect u:prop#)
                          theorem:usedby)
    NIL])
```

44

```
(dropUsed
  [LAMBDA (theorem users)
    (CLISP: UNDOABLE)                                    (* R.Erickson "30-Sep-79 12:03")
                                                         (* thm list of thms)

        (* * theorem is not longer used in the proof of users. We don't remove from :usedby, since we might be used twice.
        UsedBy willl recompute, but can only decrease, not incrrease.)


    (if users
        then (pushnew theorem:usedby ('APPROXIMATE])
)
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

(* proof states)]

```
(RPAQQ StatusFns (Assume Unassume Status PrettyStatus Proved? Finished? UpdateStatus))
(DEFINEQ
```

45

```
(Assume
  [LAMBDA (nodeid)                                    (* D.Thompson "29-Oct-80 13:58")
    (PROG (node thm)
          (node←(GetNode nodeid))
          (thm←(MakeTheorem node))
          (if node:trans
              then (DiscardTransformation node:prop# node:trans T))
          (UpdateStatus <thm> 'assumed)
          (RETURN T])
```

46

```
(Unassume
  [LAMBDA (nodeid)                                    (* R.Erickson "23-Aug-79 14:49")

          (* * undo the effect of assume: make this a provable theorem)


    (PROG ((node (GetNode nodeid))
           (thm (GetTheoremP nodeid)))
          (if thm and thm:status='assumed
              then                                    (* UpdateStatus sets status properly;)
                   (UpdateStatus <thm> 'unassume)
            else (AffirmError <"not an assumed theorem" nodeid>])
```

47

```
(Status
  [LAMBDA (thm)                                       (* R.Erickson "15-Oct-79 17:10")
                                                      (* thm a record)
                                                      (* return one of (untried tried lemmas proved assumed))

    thm:status])
```

48

```
(PrettyStatus
  [LAMBDA (thm)                                       (* R.Erickson "19-Oct-79 12:38")

          (* * Return a status comment suitable for printing)


    (SELECTQ (Status thm)
             (lemmas "awaiting lemma proof")
             (Status thm])
```

49

```
(Proved?
  [LAMBDA (thm)                                       (* R.Erickson " 8-Oct-79 19:40")
                                                      (* thm a record or theoremid)

          (* * true if proved or assumed)


    (if ~(LISTP thm)
        then thm←(GetTheorem thm))
    (thm:status MEMB '(proved assumed])
```

50

```
(Finished?
  [LAMBDA (thm)                                       (* R.Erickson "16-Oct-79 21:02")
    (if (NLISTP thm)
        then thm←(GetTheorem thm))
    (thm:status MEMB '(proved assumed lemmas])
```

```
(UpdateStatus
  [LAMBDA (thmsor#s assumer)
    (CLISP: UNDOABLE)                                   (* D.Thompson " 3-Sep-80 19:15")

          (* * Changes have been made in the proofs of thms; reset their status field. If a change occurs, it will be
          propagated up to parents. If assumer is given, it is used to change the assumed state of thms, per:
          (if assumer = 'assumed then assume it elseif assumer then unassume it.) If ApproximateStatus = T, we don't worry if
          facts used is too large a set. -
          thmsor#'s is a list of theorem records or numbers.)


     (for thm in thmsor#s bind name stat lemmas node
        do (if (NUMBERP thm)
               then thm←(GetTheorem thm))
           (node←(ThmToNode thm))
           (stat←(if assumer='assumed or ~assumer and thm:status='assumed
                     then 'assumed
                   elseif thm:leaves
                     then (if node:trans
                              then 'tried
                            else 'untried)
                   elseif lemmas←(for use in (Facts thm ApproximateStatus) unless (Proved? use)
                                    collect (NodeId use))
                     then 'lemmas
                   else 'proved))
           (if stat~=thm:status
               then thm:status←stat
                    [if stat MEMB '(proved lemmas)
                        then name←(TheoremId thm)              (* notify the user of the change in status)
                             (if (GetAnnotation (ThmToNode thm)
                                                NIL 'PROGRAM)
                                 then                          (* a program, was created by genvcs)
                                      (if stat='lemmas
                                          then (AFFIRMMAPRINT (CONCAT "Program " name
                                                                      " is awaiting the proof of "
                                                                      (Plural 'vcs lemmas))
                                                              lemmas T T)
                                        else (if ~(DISPLAYTERMFLG and GOODGUY)
                                                 then (printout NIL .TAB0 0 "Program" , name ,
                                                              "verified!"
                                                              T)
                                               else           (* highlight in inverse video for program proven by PV
                                                              member)
                                                    (printout NIL .TAB0 0 # (EnhanceHp 'C)
                                                              "!" # (EnhanceHp 'B)
                                                              , "Program" , name , "verified" #
                                                              (EnhanceHp 'C)
                                                              "!" T))
                                                    (AutoMechanism 'ProgramVerified name))
                               else                            (* a regular theorem)
                                    (if stat='proved
                                        then (printout NIL .TAB0 0 name , "proved" Period T)
                                             (AutoMechanism 'TheoremProved name)
                                      else (AFFIRMMAPRINT (CONCAT name " is awaiting the proof of "
                                                                  (Plural 'lemma lemmas))
                                                          lemmas T T]
                     (UpdateStatus node:parents)
                     (UpdateStatus (UsedBy thm])
)

(RPAQ ApproximateStatus NIL)
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

(* top level user commands and movement)]

```
(RPAQQ TreeUserFns (Arc SchemaCaseNames Try arc down name next resume retry theorem up))
(DEFINEQ
```

52

```
(Arc
  [LAMBDA (arclabel)                                    (* D.Thompson " 4-Sep-80 16:19")
    (PROG (targ)
          (if arclabel←(AFFIRMSpellingCorrect arclabel KnownNames:Arcs)
              then (if targ←(SeekArcLabelled arclabel (PCTopChain))
                       then (AscendTo targ)
                            (RETURN (Try (GetChild arclabel)))
                     else (AffirmError <"Arc not found:" arclabel>))
            else (RETURN NIL])
```

53

```
(SchemaCaseNames
  [LAMBDA (schema)                                      (* R.Erickson "23-Jul-81 19:30")

        (* * return a list of unique case labels for this induciton schema. They all end in colon)

    (if (NLISTP schema) or schema:Operator~='cases\Schema
        then                                            (* only one case)
             <'OneAndOnly: >
      else (PROG (labels oper label editComs)
                 (editComs← <<'ORR <<'F PROPOP T> <'LPQ 'F PROPOP> 2> NIL> <'LPQ 1>>)

        (* * These editor commands seek an operator to identify each case. If the case contains Prop, we find the last one
        and descend into its first argument. No matter what, we then look for the first operator. This will even work for
        atomic cases; no guarantee that the result is unique.)

                 (for case in schema:Arguments do (oper←(EDITL <case> editComs):1)
                                                    (* EDITL takes and returns an edit chain;
                                                    this is a list of expressions, the first of which is
                                                    current.)
                                                  (oper←(Shorten oper))
                                                  (label←oper)
                                                  (for i from 1 while label MEMB labels
                                                     do (* collision)
                                                        (label←(PACK* oper i)))
                                                  (labels← < !! labels label>))
                 (RETURN (for l in labels collect (PACK* l Colon])
```

54

```
(Try
  [LAMBDA (node)                                        (* R.Erickson "28-Oct-80 17:42")

        (* * This routine implements the TRY command. It takes a node name or an expression (already parsed) random access:
        put us at node, normalize, etc. -
        An AFFIRM command function.)

    (PROG (theorem parent)
          (node←(GetNode node))
          (RETURN (if theorem←(NodeToThm node)
                      then                              (* start at a theorem)
                           (UndertakeProof theorem)
                    elseif node:prop# MEMB CurrentNode:trans:children
                      then                              (* child of current node)
                           (Descend node:prop#)
                    elseif (PCMemb node:prop#)
                      then                              (* Somehwere above us)
                           (AscendTo node:prop#)
                           (Normalize T)
                    elseif CurrentTheorem:prop# MEMB node:parents
                      then                              (* try of a node under current theorem, but not a child
                                                        or ancestor of current node)
                           (UndertakeProof NIL (GetChainTo node CurrentTheorem))
                    elseif (EQLENGTH node:parents 0)
                      then                              (* try of a foreign node; must determine its parent
                                                        theorem)
```

```
                    (if node:trans
                        then                    (* since transformed, and no parents, must have been
                                                tried before.)
                             (printout NIL .TAB0 0
        "Starting proof of a disconnected subtree.
As it isn't part of the proof of a theorem"
                                                ,
                                               (NodeId node)
                                               , "is being made a theorem itself." T)
                             (if (OccursAsOperatorIn IH2OP (NodeToExpr node))
                                 then (printout NIL "Be warned that IH is undefined." T)))
                    (UndertakeProof (MakeTheorem node))
               else (if (EQLENGTH node:parents 1)
                        then parent←node:parents:1
                        else                    (* unusual. a node with >1 parent.
                                                blech.)
                             parent←[AFFIRMUSER NIL NIL <"Node" (NodeId node)
                                                         "has"
                                                         (FLENGTH node:parents)

                                                "parent theorems.  Which should be entered? "
                                                         >
                                                (MAKEKEYLST (for theorem
                                                               in (SortRecency node:parents)
                                                               collect (TheoremId theorem]
                             (if ~parent
                                 then                   (* user answered None)
                                      (ERROR!)))
                        theorem←(GetTheorem parent)
                        (UndertakeProof theorem (GetChainTo node theorem])
```

55

```
(arc
  [LAMBDA (label)                                   (* D.Thompson " 6-Nov-79 16:26")

          (* * This routine moves the user from the current position to the arc whose label is the parameter.
          -
          An AFFIRM command function.)


          (* * check label exists, is a # or ends in :)


    (Arc (EndinColon label])
```

56

```
(down
  [LAMBDA (child)                                   (* D.Thompson "22-Sep-80 13:36")

          (* * This routine moves the current proposition cursor based on the current value. It first attempts to descend to
          the child provided as the parameter; It next tries an untried child. -
          An AFFIRM command function.)


    (Try (if child
             then (GetChild child)
           elseif ~(CurrentNode:trans)
             then (AffirmError "No children!")
           elseif (for c in CurrentNode:trans:children thereis ~(True? c)
                                                               and ~(GetTransformation c))
           else

          (* * tried for the 1st untransformed now just pick one)


                (GetChild 1)))                      (* return T so auto mechanism won't fire, but the new
                                                    propn will still be printed.)
    T])
```

57

```
(name
  [LAMBDA (parms)                                   (* R.Erickson "27-Oct-80 15:10")

          (* * This routine names a node. Its parameter is a list of 2 elements, name, optional node (default current) -
```

An AFFIRM command function.)

```
(if parms:1 and ~(parms : : 2)
    then (SetName parms:1 (ExprToNode (PexecLists parms:2)))
  else (AffirmError "The proper syntax is '<name> [,<node>]:'"])
```

58

```
(next
  [LAMBDA NIL                                        (* R.Erickson "17-Mar-81 17:18")
```

(* This routine moves to the 'next' node, where the definition of next is in the AFFIRM reference manual.*
-
An AFFIRM command function.)

```
    (PROG (nx label ask)
          (nx←(Next))
          (RETURN (SELECTQ nx:1
                           (STAY (printout NIL .TAB0 0
                                           "You're at the only leaf -- there's nowhere to go."
                                           T)
                                 NIL)
                           (LEAF (printout NIL .TAB0 0 "Going to" ,)
                                                        (* can we provide a label? nx:2 is a proof chain.)
                                 (if nx:2:2 and label←(GetLabel nx:2:1 nx:2:2)
                                     then (printout NIL "leaf" , label Period T)
                                   else (printout NIL "a leaf." T))
                                 (UndertakeProof NIL nx:2))
                           (DOWN                        (* :2 is a list where :1 we use. :-1 we target)
                                 (printout NIL .TAB0 0 "Going to lemma" , nx:2:-1)
                                 (for l in (REVERSE nx:2)::1 do (printout NIL , "which is used by" ,
                                                                          l)
                                    finally (printout NIL Period T))
                                 (Restart (GetTheorem nx:2:-1)))
                           (UP (if ~(nx::2)
                                   then                 (* only one choice)
                                        (printout NIL .TAB0 0 "Going to" , nx:2 Period T)
                                        ask←'Y
                                   else                 (* show user choices. suggest first, seek confirmation)
                                        (AFFIRMMAPRINT "Options are" nx::1 NIL T)
                                        (printout NIL T "I suggest" , nx:2)
                                        ask←(AFFIRMUSER NIL 'Y "Ok?"))
                               (if ask='Y
                                   then (Restart (GetTheorem nx:2))
                                 else NIL))
                           (NIL (printout NIL .TAB0 0 "The proof of this part is finished." T)
                                (printHelper 'status '(unproved))
                                NIL)
                           (Unexpected 'NextCategory T])
```

59

```
(resume
  [LAMBDA NIL                                        (* D.Thompson " 5-Nov-79 15:13")
```

(* This routine recovers the lastchain field from the current node, a theorem -*
An AFFIRM command function.)

```
    (if ~((PCTopChain)::1)
        then (if CurrentTheorem:lastchain::1
                 then (DescendAlongChain CurrentTheorem:lastchain)
               else (AffirmError "Nothing to resume. (Do you mean 'normalize'?)"))
      else (AffirmError "You're not at a theorem for resume."])
```

60

```
(retry
  [LAMBDA NIL                                        (* R.Erickson "15-Nov-79 13:16")
```

(* This routine retries the current theorem. -*
An AFFIRM command function.)

```
    (if CurrentTheorem
        then (AscendTo CurrentTheorem:prop#)
```

```
          (Normalize T)
    else (AffirmError "There's no current theorem."])
```

61

```
(theorem
  [LAMBDA (node)                                    (* D.Thompson " 1-Nov-79 16:01")

          (* * This routine enters the theorem, but leaves the proof stack. current propn untouched. -
          An AFFIRM command function.)


    (PROG (thm)
          (CheckForCurrentType)
          (thm←(MakeTheorem node))
          (printPropn thm:prop#])
```

62

```
(up
  [LAMBDA (n)                                       (* D.Thompson "22-Sep-80 13:34")

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


    (AscendBy (OR n 1))
    (Normalize T)                                   (* return T so auto mechanism won't fire, but the new
                                                    propn will still be printed.)
    T])
)
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

(* toplevel structure commands used by theoremprover)]

```
(RPAQQ StructFns (Transform DeleteNodes DiscardTransformation MakeTheorem UnMakeTheorem
                         TransformToTrue))
(DEFINEQ
```

63

```
(Transform
  [LAMBDA (transformation node command)                    (* R.Erickson "28-Oct-80 17:23")

        (* * save the old xform, if any. Do bookkeeping for parents, leaves, status, factsused. Return the # of the 1st
        nontrivial child, if any)


    (PROG (circle provees chains thm nextchild oldnodes label)
          (node←(GetNode node))

          (* * check for permissible)


          (if (True? node)
              then (AffirmError "current node, TRUE, cannot be transformed!"))
          (if circle←(CircularTransformP transformation node:prop#)
              then (if node:prop#=transformation:children:1
                       then (if command
                                then (printout NIL .TAB0 0 (L-CASE command T)
                                                   , "had no effect." T)
                                     (RETURN NIL)
                              else (AffirmError "Transformation had no effect."))
                     else (printout NIL "Cannot transform node" , (NodeId node)
                                        , "because of circular reasoning involving" ,)
                          (AFFIRMMAPRINT NIL circle T T)
                          (AffirmError NIL)))

          (* * check format of xform.)


          (if transformation:labels
              then                                         (* insist they end in colon. This would be an
                                                           inplementation error)
                   (if (for l in transformation:labels always (EndsColon l))
                     else (AffirmError <"recoverable implementation bug: bad labels"
                                        transformation:labels "user may proceed." >
                                       'mild)
                          transformation:labels←(for l in transformation:labels
                                                   collect (EndinColon l)))
                   KnownNames:Arcs←(UNION transformation:labels KnownNames:Arcs)
                                                           (* deliberately not undoable) )
          (for use in transformation:uses do (MakeTheorem use))
          (/replace event# of transformation with CurrentEventNumber)

          (* * make the changes)


          (provees←(Provees node))                         (* provees are the records of thms under proof by
                                                           transformation -
                                                           may be self)
          (chains←(for p in provees collect (GetChainTo node p)))
                                                           (* chains are needed for rebuilding leaves)
          (if node:trans
              then                                         (* save old xform, update records)
                   (if ~(MEMBER node:trans node:oldtrans)
                       then (/replace oldtrans of node with <node:trans ! node:oldtrans>))
                   (RemoveChildren node provees)
            else                                           (* leaf)
                 (RemoveLeaf node:prop# provees))

          (* * will throw in RulesUsedToNormalize here)


          (oldnodes←(LDIFFERENCE transformation:children (if node:trans
                                                             then node:trans:children)))
          (/replace trans of node with transformation)
          (nextchild←(if (NLISTP transformation:children)
                         then NIL
                       elseif (for n in transformation:children
                                 thereis (if ~(True? n)
                                             then (FlagArcDescent transformation n)
```

```
                                                    T))
                        else transformation:children:1))      (* compute here for the sake of the oldnodes message.)

        (* * Does this transformation hit any old structure, excluding nodes which were already children of us? If so,
        remind the user.)


        (oldnodes←(for n in oldnodes when (GetNode n):trans collect n))
                                                              (* new children, have structure.
                                                              Do we want implicit??)
        [if oldnodes
            then                                              (* tell user)
                 (PRINTBELLS)                                 (* SURPRISE!)
                 (printout NIL T

"By the way, this command has generated the following children which already
existed, and which already have proof attempts.")
                 (if nextchild MEMB oldnodes
                     then (printout NIL T "Thus, you are not currently at a leaf."))
                 (for n in oldnodes
                    do (label←(for c in transformation:children as l in transformation:labels
                                 when c=n do (RETURN l)))
                       (printout NIL T .TAB0 10 ,, (if label
                                                       then label
                                                     else (NodeId n)))
                       (if label←(Provees (GetNode n))
                           then (printout NIL ,, "used in the proof of" , (Plural 'theorem label)
                                        ,)
                                (AFFIRMMAPRINT NIL (for l in label collect (TheoremId l)))
                         else (printout NIL ", an orphan."]
        (AddChildren transformation provees chains)
        (UpdateStatus provees (if (NodeToThm node):status='assumed
                                  then                        (* If assumed thm, unassume it.
                                                              -
                                                              here we depend on the fact that Provees
                                                              (thm) = <thm>)
                                       (printout NIL " (unassuming " (NodeId node)
                                                     ") ")
                                       'unassume))
        (RETURN nextchild])
```

64

(DeleteNodes

```
  [LAMBDA (node#s)                                            (* R.Erickson "21-Oct-80 18:30")

        (* * Given a list of nodes to garbage-collect. Remove from Nodes. PropStorage. Members of this set should not be
        theorems or children of nodes outside the set.)


    (if node#s
        then node#s←(MKLIST node#s))
    (for n in node#s do                                       (* one last check for consistency)
                        (if (GetTheoremP n)
                            then (AffirmError <"Attempt to GC a theorem" n> 'internal)
                          elseif (GetNode n):parents
                            then (AffirmError <"Attempt to GC a parented node" n> 'internal)))

        (* * We don't need to DiscardTransformations, since the :parents pointer is only to theorems, and, having to
        theorems, there are no leaves either.)


    (/SET 'Nodes (for n in Nodes unless n:prop# MEMB node#s collect n))
    (/replace ExprTo# of PropStorage with (for al in PropStorage:ExprTo# unless al::1 MEMB node#s
                                             collect al))
    (/replace #ToExpr of PropStorage with (for al in PropStorage:#ToExpr unless al:1 MEMB node#s
                                             collect al))
    node#s])
```

65

(DiscardTransformation

```
  [LAMBDA (nodeid trans saveold)                              (* R.Erickson " 8-Oct-79 19:43")
    (PROG (node provees chains thm)
          (node←(GetNode nodeid))
          (if trans=node:trans
              then                                            (* drop the current xform. Cut off our descendants, make
                                                              us a leaf in parents (or self in thm))
                   provees←(Provees node)
```

```
            (RemoveChildren node provees)
            (if saveold
                then                                    (* (pushnew node:oldtrans trans))
                     (if trans ~MEMB node:oldtrans
                         then (/replace oldtrans of node with <trans ! node:oldtrans>)))
            (/replace trans of node with NIL)
            chains←(for p in provees collect (GetChainTo node:prop# p))
            (AddLeaf provees chains)
            (UpdateStatus provees)
       elseif trans MEMB node:oldtrans
         then (/replace oldtrans of node with (REMOVE trans node:oldtrans))
       else (SHOULDNT])
```

66

(MakeTheorem

```
  [LAMBDA (node)                                        (* R.Erickson "24-Sep-80 14:29")
```

(* cause this to be a theorem. If untransformed, it has initial status 'open, itself as leaf.*
If it is someone's child, update them accordingly. Move bookkeeping to here.)

```
    node←(GetNode node)
    (if (GetTheoremP node:prop#)
      else (PROG ((numb (node:prop#))
                  thm parents)
                 (if (True? node)
                     then (AffirmError "Attempt to make TRUE a theorem" 'internal))
                 (thm←(create Theorem
                              prop# ← numb
                              status ←('untried)
                              leaves ←(<<numb>>)))      (* (push Theorems thm))
                 (ADDTOLIST 'Theorems thm)
                 (parents←(for p in node:parents collect (GetTheorem p)))
                 (if parents
                     then                               (* leave the nest: -leaves, + factsused)
                          (if node:trans
                              then (RemoveChildren node parents)
                            else (RemoveLeaf node parents))
                          (AddUsage <numb> parents))
                 (if node:trans
                     then (/replace leaves of thm with NIL)
                          (AddChildren node:trans <thm> <<numb>>)
                                                        (* sets leaves)
                          (UpdateStatus <thm>))
                 (RETURN thm])
```

67

(UnMakeTheorem

```
  [LAMBDA (node)                                        (* R.Erickson "26-Oct-80 17:33")
```

(* We previously made node a thm, and now wish to undo this fact. We zap ourselves, disown our children, and*
become transparent to our parents. NO EFFECT if impossible. Return T if succeed.)

(* NO guarantee this is right for theorems which have parents/users.)*

```
    (PROG ((thm (GetTheorem node:prop#))
           chains parents users)
          (if users←(UsedBy thm)
              then (printout NIL T "Can't make" , (TheoremId thm)
                             , "a nontheorem because of use as lemma by" ,)
                   (AFFIRMMAPRINT NIL (for u in users collect (TheoremId u)))
                   (RETURN))
          (DROPFROMLIST 'Theorems thm)
          (DropUsage (Facts thm)
                     <thm>)                             (* get rid of our lemmas. If had parents, they would
                                                        need to adopt.)
          (RemoveChildren node <thm>)
          (parents←(for p in node:parents collect (GetTheorem p)))
                                                        (* Can a theorem have parents? Think not, descend would
                                                        break.)
          (if parents
              then (AffirmError "Theorem has parents. Can't do discard." 'internal)
                   (DropUsage <thm:prop#> parents)
                   chains←(for parent in parents collect (GetChainTo node parent))
```

```
                (if node:trans
                    then (AddChildren node:trans parents chains)
                  else (AddLeaf parents chains))
                (UpdateStatus parents)                    (* ?))
        (RETURN T])
```

68

```
(TransformToTrue
  [LAMBDA (nodeid)                                        (* R.Erickson "15-Nov-79 13:15")
    (PROG (trans)
          (if ~(True? (GetNode# nodeid))
              then trans←(create Transformation
                                 children ← NIL)
                   (Transform trans nodeid)
                   (Annotate 'Implicit trans 'TYPE)       (* so we know not to print)])
)
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

(* maintenance of parentage (:parents, :factsused, and :leaves))]

```
(RPAQQ ParentageFns (AddChildren AddChildren1 RemoveChildren RemoveChildren1 MapChildren MapChildren1)
)
(DEFINEQ
```

69

```
(AddChildren
  [LAMBDA (transform parents chains)                    (* R.Erickson "21-Aug-79 19:14")
                                                        (* parents are records, chains are short)

          (* * Insert this transformation, and its targets, as children of parents. parents adopt descendants of
          transformation. Take account of childrens leaves (deletion of old leaf is NOT done here), include facts used.
          chains:k is an edit chain, below parents:k, from our parent)

    (AddChildren1 transform parents chains])
```

70

```
(AddChildren1
  [LAMBDA (transform parents chains)                    (* R.Erickson "28-Oct-80 17:26")

          (* * process one transform, recurse)

    (if transform:uses
        then (AddUsage transform:uses parents))
    (for child# in transform:children bind cnode newchains unless (True? child#)
       do (cnode←(GetNode child#))
          (/replace parents of cnode with (UNION cnode:parents (for t in parents collect t:prop#)))
                                                        (* adopt kids)
          (newchains←(for c in chains collect <child# ! c>))
                                                        (* extend edit chains to here)
          (if (NodeToThm cnode)
              then (AffirmError <"Implementation restriction: can't proceed while" (TheoremId child#)
                                 "is a theorem.
To proceed, discard it." >)                             (* parents are not leaves)
                   (AddUsage <child#> parents)
            elseif cnode:trans
              then (AddChildren1 cnode:trans parents newchains)
            else (AddLeaf parents newchains])
```

71

```
(RemoveChildren
  [LAMBDA (node Parents)                                (* R.Erickson "21-Aug-79 19:17")
    (DECLARE: (SPECVARS Parents))                       (* Parents are theorem records. Usually a list of one)

          (* * disown descendants of node. remove from leaves in Parents. (we DONT fill hole here) check for rules no longer
          used.)

    (if node:trans
        then (DropUsage node:trans:uses Parents))
    (MapChildren node 'RemoveChildren1])
```

72

```
(RemoveChildren1
  [LAMBDA (node thm)                                    (* R.Erickson " 9-Nov-79 17:43")

          (* * Called by MapChildren. Uses Parents, bound in RemoveChildren. Drop one node,trans)

    (if node:trans:uses
        then (DropUsage node:trans:uses Parents)        (* old rules))
    (for parent in Parents do (/replace parents of node with (/DREMOVE parent:prop# node:parents))
                                                        (* disown))
    (if thm
        then                                            (* we were a factused)
             (DropUsage <node:prop#> Parents)
      elseif ~(node:trans)
        then                                            (* else MapChildren will descend)
             (RemoveLeaf node:prop# Parents])
```

73

```
(MapChildren
  [LAMBDA (node mapfn)                                    (* R.Erickson "17-Sep-79 19:38")

          (* * apply mapfn to <node (GetTheorem node:prop # ) > at each Proper, nontrue descendant of node #. up thru, but not
          past, theorems)


    node←(GetNode node)
    (if node:trans
        then (for child in node:trans:children do (MapChildren1 child mapfn])
```

74

```
(MapChildren1
  [LAMBDA (node# mapfn)                                   (* R.Erickson "20-Aug-79 16:43")
    (if ~(True? node#)
        then (PROG ((node (GetNode node#))
                    (thm (GetTheoremP node#)))
                   (APPLY* mapfn node thm)
                   (if ~thm and node:trans
                       then (for child in node:trans:children do (MapChildren1 child mapfn])
)
(SETTEMPLATE (QUOTE MapChildren)
             (QUOTE (EVAL FUNCTION . PPE)))
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

```
(* Proof Chain queries, ops; movement functions) ]

(DEFINEQ

                                                                    75

(PCTopChain
  [LAMBDA NIL                                    (* R.Erickson "16-Oct-79 21:03")
                                                 (* return the edit chain)
    ProofChain])

                                                                    76

(PCCurrNode
  [LAMBDA NIL                                    (* R.Erickson "15-Oct-79 17:19")
                                                 (* return the nodeid which is current, according to the
                                                 proof chain)
    ProofChain:1])

                                                                    77

(PCMemb
  [LAMBDA (elem)                                 (* R.Erickson "15-Oct-79 17:21")
                                                 (* is elem in (PCTopChain) ?)
    (elem MEMB ProofChain])

                                                                    78

(Descendant?
  [LAMBDA (parent Child)                         (* R.Erickson "27-Oct-80 15:18")
                                                 (* DECLARE: (SPECVARS Child))
                                                 (* both node #'s)

        (* * T if child derives, in one or more steps of the proof tree, from parent. Does NOT follow lemma usage or theorem
        children.)

    (MapChildren parent (FUNCTION (LAMBDA (node thm?)        (* MapChildren normally returns NIL)
                    (if node:prop#=Child
                        then (RETFROM 'MapChildren T])
)
(DEFINEQ

                                                                    79

(Descend
  [LAMBDA (node#s dontNormalize)
    (CLISP: UNDOABLE)                            (* R.Erickson "16-Oct-79 21:06")

        (* node #'s = prop #' or list thereof. We move down one by one, make the last current, and return the proposition.
        NIL->return NIL. We print arc labels as we go. Unless dontNormalize, we call Normalize on the result.)

    (PROG (res)
          (if node#s
              then node#s←(MKLIST node#s)
                   (if CurrentNode and node#s:1=CurrentNode:prop#
                       then (AffirmError <"cant descent TO, since are already AT " node#s:1>))
                   (for ns on node#s bind n trans pos do (n←ns:1)
                                                         (if (GetTheoremP n)
                                                             then (AffirmError "Descend to theorem"
                                                                               'internal))
                                                         (/SET 'ProofChain <n ! ProofChain>)
                                                         (if ns::1 and trans←(GetTransformation
                                                                               n)
                                                             then
                                                         (* next element may be labelled.
                                                         Tell user where we are going)
                                                                (FlagArcDescent trans ns:1)))
                   res←(MakeCurrent (GetNode node#s:-1))
                   CurrentTheorem:lastchain←ProofChain
                   (RETURN (if dontNormalize
                               then res
                           else (Normalize res])
```

80

```
(DescendAlongChain
  [LAMBDA (chain)                                    (* R.Erickson "10-Sep-79 17:15")
                                                     (* given a short edit chain, of which we may be at the
                                                     origin. descend to tip.)

    (Descend (if chain:-1=CurrentTheorem:prop#
                 then (REVERSE chain)::1
               else (REVERSE chain])
```

81

```
(AscendTo
  [LAMBDA (nodeid)                                   (* R.Erickson "16-Oct-79 21:06")

        (* * rise in the edit chain of the current theorem, to the designated node. If T, rise all the way.)

    (if nodeid=T
        then (/SET 'ProofChain NIL)
             (/SET 'CurrentTheorem NIL)
             (MakeCurrent T)
      else (PROG ((node# (GetNode# nodeid))
                  found)
                 (found←(node# MEMB ProofChain))
                 (if found
                     then                            (* ok)
                          (/SET 'ProofChain found)   (* here we assume chainends in thm)
                          (MakeCurrent (GetNode node#))
                          (RETURN node#)
                   else (AffirmError <"not below" node#>])
```

82

```
(AscendBy
  [LAMBDA (n)                                        (* R.Bates "23-Jan-80 13:40")

        (* * rise by n nodes -- limited to within the current proof chain)

    (PROG (destination)
          (destination←(CADR (FNTH (PCTopChain)
                                   n)))
          (if destination
              then (AscendTo destination)
            else (AffirmError <"can't rise" n>])
)
(DEFINEQ
```

83

```
(ClearProof
  [LAMBDA NIL                                        (* R.Erickson "30-Oct-79 23:11")
    (/SET 'Nodes <Truenode>)
    (/SET 'Theorems NIL)
    (/SET 'ProofChain NIL)
    (MakeCurrent T)
    (/SET 'PropStorage (create PropStorage))
    (ClearNames)
    (/replace Arcs of KnownNames with NIL])
```

84

```
(MakeCurrent
  [LAMBDA (node)                                     (* R.Erickson " 2-Nov-79 18:19")

        (* * node is T (for TRUE) or a node record. Assuming the edit chains are properly set, establish currency.
        Return the proposition itself. We assume, unless node = T, that our caller has checked for any needed changes to
        CurrentTheorem.)

    (if node
        then (ZapSkolemFunctions)
             (if node=T
                 then (/SET 'CurrentNode Truenode)
                      (/SET 'CurrentTheorem NIL)
                      (/SET 'CurrentPropn TRUE)
```

```
                else (printAnnotation? (GetAnnotation node))
                     (/SET 'CurrentNode node)
                     (/SET 'CurrentPropn (GetExpression node:prop#)))
        else (if (PCCurrNode)
                 then (MakeCurrent (GetNode (PCCurrNode)))
                else (MakeCurrent T])
)
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

(* leaves; use proof chains)]

```
(RPAQQ LeafFns (AddLeaf RemoveLeaf SortLeaves Next nextLemmas nextUsers unfinishedAncestors NextLeaf
                       orderofLeaves GetChainTo seekChainTo SeekArcLabelled))
(DEFINEQ
```

85

```
(AddLeaf
  [LAMBDA (thms chains)
    (CLISP: UNDOABLE)                                    (* R.Erickson "13-Aug-79 16:14")
                                                         (* thms are records, chains short)

        (* * add a leaf under thms. We may be reached by different chains from different thms. chains:k corresponds to
        thms:k)

    (for thm in thms as chain in chains do (if ~(MEMBER chain thm:leaves)
                                               then                (* no leaves to us-- add)
                                                    (push thm:leaves chain])
```

86

```
(RemoveLeaf
  [LAMBDA (p# thms)                                      (* R.Bates "14-Dec-79 14:43")
                                                         (* thms are records, p# a prop# or node)

        (* * remove the leaf corresp. to p# under each of thms. If more than one descending path, zap all of them.
        May be reflexive)

    (if (LISTP p#)
        then p#←p#:prop#)
    (for thm in thms do ((FASSOC p# thm:leaves) or (SHOULDNT))
                        (/replace leaves of thm with (/DREMASSOC p# thm:leaves])
```

87

```
(SortLeaves
  [LAMBDA (thm)                                          (* R.Erickson "16-Aug-79 21:09")

        (* * put the leaves of thm in "nice" order: siblings should be together, and in the order of their arcs
        (non-arc children come last for each parent))

    (/replace leaves of thm with (SORT < ! thm:leaves> 'orderofLeaves])
```

88

```
(Next
  [LAMBDA (disambig)                                     (* R.Erickson " 9-Nov-79 13:18")

        (* * Computes the next place to go. If disambig = T, ASKUSER to resolve any ambiguities in upward search.
        Result is keyed according to CAR as follows -
        STAY We are at the only leaf. stay put -
        LEAF :2 is the next leaf, ~ = where we are -
        DOWN CDR is a list. in descending preference, of lower lemmas needing proof. For each entry, :1 is the lemma we use,
        and :-1 the thing to be proven. These are theorem ids. -
        UP CDR is preference as above. Each entry is a unproven ancestor -
        NIL We are done with this proof section.)

    (PROG (nxleaf lemmas nxusers)
          (if ~CurrentTheorem
              then (RETURN <NIL>)
            elseif ~(Finished? CurrentTheorem)
              then nxleaf←(NextLeaf ProofChain CurrentTheorem)
                                                         (* returns NIL if the only one is where we are)
                   (if nxleaf
                       then (RETURN <'LEAF nxleaf>)
                     else (RETURN <'STAY >))

        (* * we are finished -
        are there lemmas?)
```

```
        elseif ~(Proved? CurrentTheorem)
          then (RETURN <'DOWN !(nextLemmas CurrentTheorem)
                       >)

    (* * do any unfinished call us?)


        elseif nxusers←(unfinishedAncestors CurrentTheorem)
          then (RETURN <'UP ! nxusers>)
        else

    (* * done)


             (RETURN <NIL>])
```

89

```
(nextLemmas
  [LAMBDA (thm)                                    (* R.Erickson "29-Oct-79 13:58")

        (* * given a theorem, return a list, in preference, of unproven lemmas used. Elements are lists, :1 use by thm, :-1
        is unproven)


    (PROG (lems)
          (if ~(type? Theorem thm)
              then (SHOULDNT))
          (lems←(for l in (Facts thm) unless (Proved? l←(GetTheorem l)) collect l))
          (lems←(SortRecency lems))
          (RETURN (for lem in lems join (if (Finished? lem)
                                            then                (* unproven, so has lemmas)
                                                 (for l2 in (nextLemmas lem) collect
                                                                          <(TheoremId lem)
                                                                            ! l2>)
                                          else                  (* we are it)
                                                                (* a list for joi8n, 1 for entry)
                                               <<(TheoremId lem)
                                               >>])
```

90

```
(nextUsers
  [LAMBDA (thm disamb)                             (* R.Bates "14-Dec-79 14:43")

        (* * If thm is proven, return preferential list of unfinished lemmas someplace above thm; ei9ther a user of a
        subling. Each entry has :1 a user of thm, and :-1 unfinished (or a list of unproven parents) disamb causes ASKUSER
        if necessary, but user may opt out If thm is unproved, return a list going down to unfinished lemmas.)


    (PROG (pars)
          (if ~(type? Theorem thm)
              then (SHOULDNT))
          (if ~(Proved? thm)
              then

        (* * mark us as on path)


                (RETURN (for ls in (nextLemmas thm) collect <(TheoremId thm) ! ls>))
          else pars←(unfinishedParents thm)
               (SELECTQ (FLENGTH pars)
                        (0                                  (* none- return NIL)
                           (RETURN))
                        (1                                  (* patch)
                           (RETURN (for l in (nextLemmas pars:1) collect <(TheoremId thm) ! l>)))
                        (AffirmError <
"I'm sorry, but the case of multiple unfinished users is
not implemented. Try one of" (for t in pars collect (TheoremId t))
                                     >])
```

91

```
(unfinishedAncestors
  [LAMBDA (theorem)                                (* R.Erickson "27-Nov-79 12:47")
```

```
        (* * return first ancestor/sibling along each path which is unfinished.)

    (if ~(type? Theorem theorem)
        then (SHOULDNT))
    (REMOVEDUPLICATES (for p in (UsedBy theorem) join (p←(GetTheorem p))
                                                      (if (Proved? p)
                                                          then (unfinishedAncestors p)
                                                        elseif (Finished? p)
                                                          then (for series in (nextLemmas p)
                                                                  collect series:-1)
                                                        else <(TheoremId p)
                                                              >])
```

92

```
(NextLeaf
  [LAMBDA (shortchain thm nth)                          (* R.Erickson "19-Oct-79 15:49")

        (* * return the (nth) "next" leaf's short edit chain; if shortchain is given, that's where we are now, and we want
        someplace different. If we fail, return NIL.)


    (if thm:leaves
        then (if shortchain
                 then (PROG (augleaves prospects)
                            (augleaves←(UNION <shortchain> < ! thm:leaves>))
                                                            (* ensure present is a member)
                            (augleaves←(SORT augleaves 'orderofLeaves))
                            (prospects← < !(MEMBER shortchain augleaves)::1
                                        !(for e in augleaves until (EQUAL e shortchain)
                                            collect e)
                                        >)

        (* we want to start with those following shortchain, then we take from the beginning of the order given;
        we exclude shortchain. We don't loop around if nth is too big, just fail.)

                            (if nth
                                then (to nth-1 do (pop prospects)))
                            (if prospects
                                then (RETURN prospects:1)))
              else thm:leaves:1])
```

93

```
(orderofLeaves
  [LAMBDA (a b)                                         (* R.Erickson " 8-Oct-79 20:07")

        (* * return T if a goes before b, otherwise NIL. We are given two leaves (short edit chains) -
        we choose an order which places siblings together. Siblings are placed in the order that they are children;
        nonchildren come after children, and are ordered numerically (so nodes created later come later) -
        We expect a and b are nonNIL, numeric lists.)


    a←(REVERSE a)
    b←(REVERSE b)                                       (* expensive, but who cares? This isn't done often)
    (if a:1~=b:1
        then (ILEQ a:1 b:1)
      else                                              (* two leaves should differ somewhere)
           (for ta on a as tb on b bind node children
              do                                        (* assert ta:1 = tb:1)
                 (if ta:2~=tb:2
                     then node←(GetNode ta:1)
                          children←(if node:trans
                                       then node:trans:children)
                                                        (* SORT wants T or NIL)
                          (RETURN (AND (if tb:2 MEMB children
                                           then tb:2 MEMB (ta:2 MEMB children)
                                         elseif ta:2 MEMB children
                                         elseif ~(ta : : 1) or ~(tb : : 1)
                                         else (ILEQ ta:2 tb:2))
                                       T])
```

94

```
(GetChainTo
  [LAMBDA (node from)                                   (* R.Bates "14-Dec-79 14:42")
```

```
                                                    (* from is a theorem)

        (* * return a (short) edit chain from parent "from" to descendant "node" (We are guaranteed parentage) We give
        preferance to the current chain.)


    (PROG ((node# (GetNode# node)))
          (RETURN (if node#=from:prop#
                      then                                  (* reflexive)
                           <node#>
                     elseif from=CurrentTheorem and node# MEMB ProofChain
                     elseif (for leaf in from:leaves bind m when m←(node# MEMB leaf)
                                do (RETURN m))
                     else

        (* * there arent any leaves at node or below. We grit our teeth and descend, looking.)


                          (if (for child in (ThmToNode from):trans:children bind seek
                                 when seek←(seekChainTo child node#) do (RETURN < !(REVERSE seek)
                                                                                  from:prop#>))
                            else (AffirmError <"GetChainTo caller lied: node not below from" node
                                                 from>
                                              'internal])
```

95

```
(seekChainTo
  [LAMBDA (lower target)                                (* R.Erickson " 5-Nov-79 14:20")
                                                        (* both p#'s)

        (* * look for a chain from lower to target -
        :1 is target. :-1 is lower)


    (PROG (lowertrans)
          (RETURN (if lower=target
                      then <lower>
                     elseif lowertrans←(GetTransformation lower)
                       then (for child in lowertrans:children bind seek
                                 when seek←(seekChainTo child target) do (RETURN <lower ! seek>])
```

96

```
(SeekArcLabelled
  [LAMBDA (label shortchain)                            (* R.Erickson "15-Aug-79 13:42")

        (* * We are at the bottom of a given short edit chain. Return the # of the node, if any, which is in the chain at or
        above us and which has a child with the designated label)


    (for n in shortchain thereis                        (* start at bottom, work up)
                                 (GetChild label n T])
)
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

```
(* annotations) ]

(DEFINEQ

                                                                    97

(Annotate
  [LAMBDA (note target type)                    (* R.Erickson "15-Nov-79 13:11")

          (* attach a comment to target, which is either a node or a transformation, but a record in either case.
          If type is given, then this is a system note, which we store by ASSOC with type. Overwrite old.)


    (PROG (ann)
          (OR (type? Node target)
              (type? Transformation target)
              (AffirmError "Illegal arg to Annotate" 'internal))
          [ann←(if target:annotation
                   else (target:annotation←(create annotation]
          (if type
              then (if ann:systemnote
                       then (/PUTASSOC type note ann:systemnote)
                     else (/replace systemnote of ann with <(<type ! note>)
                                                            >))
            else (/replace usernote of ann with note])

                                                                    98

(GetAnnotation
  [LAMBDA (node transformation type)            (* R.Bates "14-Dec-79 14:43")
                                                (* transformation a record)

          (* * return the note, if any. If given type, fetch corresp. system comment)


    (PROG (target)
          (node←(GetNode node))
          (target←(if ~transformation
                      then node
                   else (CheckTransformationOccurs transformation node)
                        transformation))
          (RETURN (if target:annotation
                      then (if type
                               then (FASSOC type target:annotation:systemnote)::1
                             else target:annotation:usernote])

                                                                    99

(RestoreAnnotations
  [LAMBDA (node anns)                           (* R.Erickson " 2-Sep-79 17:57")
                                                (* anns is a copy of the annotation field at some
                                                previous time; for example, the node may have been kept
                                                & entered.)


    (/replace annotation of node with anns])
)
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

(* printing proof trees)]

```
(RPAQQ PrintFns (PrintBoth PrintProof printProofNode? printProofNode printPrefix printCommand?
                           printAnnotation? printPropn))
(DEFINEQ
```

100

```
(PrintBoth
  [LAMBDA (node listApplied)                                   (* R.Erickson " 1-Oct-79 13:29")
```

(* print tree alone, then with propns, both times with index #'s for comparison. May list exprs applied as lemmas, after first tree only.)*

```
    (if (PrintProof node listApplied NIL T) gt 10
        then                                                   (* leave some space between)
             (if (OUTPUT)=T
                 then (printout NIL .SKIP 3)
               else (printout NIL .PAGE))
      else (printout NIL .SKIP 3))
    (PrintProof node NIL T T])
```

101

```
(PrintProof
  [LAMBDA (node list? props? printline#s)                      (* D.Thompson " 7-Nov-80 16:12")
```

(* Print proof tree there and below. -*
node is a record. PrintProofNode? gets called with (node: arc label to it if any; indentation) at each traversed node; it may call PrintProofNode. We use a series of global vars to control printing. -
ProofLine# counts tree entries, PrintLine#s tells us whether to show them. -
Below any parent with labelled children, PrintingParentSequence is a list of all arc labels to the root.
Its last element is the name of the root, and PrintingParentLine# the proofline# of its most recent child; these are used to print label (parent): where useful. -
PrintPropns = T->print both. -
PrintingNamesSeen, if non-NIL, gets a list of names in the tree -
list? causes us to print, after the tree, if the root is a theorem, either lemmas used (list? = T) or named nodes (list? = 'named))

```
    (RESETVARS ((ProofLine# 1)
                (PrintLine#s printline#s)
                (PrintingParentSequence (<(NodeId node)
                                           >))
                (PrintingParentLine# 0)
                (PrintPropns props?)
                (PrintingNamesSeen (if list?='named
                                       then <NIL>)))
               [RESETFORM (GCGAG)
                          (PROG ((thm (NodeToThm node)))
                                (if thm
                                    then (printout NIL .TAB0 0 "theorem" , (TheoremId thm)
                                                   Comma , # (PrettyPrint (NodeToExpr node)
                                                                          T)
                                                   CommandTerminator T)
                                         (if (Facts thm)
                                             then (AFFIRMMAPRINT
                                                    (CONCAT (TheoremId thm)
                                                            " uses")
                                                    [for th in (Facts thm) bind info
                                                       eachtime info←(GetTheorem th)
                                                       collect (CONCAT (TheoremId info)
                                                                       (SELECTQ info:status
                                                                                (proved
                                                                                 ExclamationPoint)
                                                                                (assumed PercentSign)
                                                                                (untried QuestionMark)
                                                                                (PROGN NullString]
                                                    T T)))
                                (if node:trans
                                    then (printProofHeader)
                                         (printProofNode? node NIL 6))
                                (if list? and (list?='named or thm and (Facts thm))
                                    then                       (* print lemmas used, separated from tree by bars.)
                                         (printout NIL T Hyphens)
                                         (for f in (if list?='named
                                                       then (DREMOVE NIL PrintingNamesSeen)
```

```
                                                    else (Facts thm))
                                                do (printPropn f))
                                               (printout NIL T Hyphens T]
            (RETURN ProofLine#])
```

102

```
(printProofNode?
  [LAMBDA (node arclabel indentation)                    (* R.Erickson "30-Sep-79 12:36")

          (* * deferred)


    (printProofNode node arclabel indentation T T T PrintPropns T])
```

103

```
(printProofNode
  [LAMBDA (node arclabel indentation command? args? annotation? propn? descend?)
                                                         (* D.Thompson " 7-Nov-80 16:09")
                                                         (* node a record)

          (* * printout a node of the proof tree, according to specs. Call printProofNode? for children, as appropriate.
          Update PrintingNamesSeen.)


    (PROG ((thm (NodeToThm node))
           (trans (node:trans)))
          (if PrintingNamesSeen
              then (pushnew PrintingNamesSeen (GetName node)))
          (printPrefix node thm)
          (if arclabel
              then (printout NIL .TAB0 indentation arclabel)
                   (if ProofLine#-PrintingParentLine# gt 3
                       then (printout NIL . # (MAPRINT (REVERSE PrintingParentSequence::1)
                                                       NIL LeftCurlyBracket RightCurlyBracket ", ")))
                   (add indentation 3)                   (* label lines up with indentation supplied, following
                                                         lines are further indented))
          (add ProofLine# 1)

          (* * print out the body)


          (printout NIL .TAB0 indentation)
          (if (True? node) or trans and (GetAnnotation node trans 'TYPE)='Implicit
              then (printout NIL (if arclabel
                                     then "immediate"
                                   else "(proven!)")
                             T)
                   (RETURN)
            else (printout NIL (NodeId node)
                           .TAB
                           (indentation+4))
                 (printCommand? node trans command? args?)
                 [if annotation?
                     then (printAnnotation? (GetAnnotation node))
                          (if trans
                              then (printAnnotation? (GetAnnotation node trans]
                 (if propn?
                     then (PrettyPrint (GetExpression node:prop#)
                                       T)))

          (* * whew! now move on)


          (printout NIL T)

          (* * do children. If labelled or more than one child, indent over one. If labelled, remember parent.)


          (if descend? and trans
              then (if trans:labels
                       then                              (* set up for a(b,c) labels in children.)
                            [RESETVARS ((PrintingParentLine# ProofLine#))
                                       (for child in trans:children as label in trans:labels
                                          do (RESETVARS ((PrintingParentSequence (<label
                                                                                  !
                                                                                  PrintingParentSequence>)))
                                                        (printProofNode? (GetNode child)
```

```
                                                        label indentation+1]
            else (for child in trans:children do (printProofNode? (GetNode child)
                                                                  NIL
                                                                  (if trans:children::1
                                                                      then indentation+1
                                                                    else indentation))
                                                 (if trans:children::1
                                                     then
                                                  (* separate the straight-line sections)
                                                         (printout NIL T])
```

104

```
(printPrefix
  [LAMBDA (node thm)                                    (* R.Erickson "19-Oct-79 15:39")

          (* * print the *?!%|%%)


    (PROG ((stuff (<NIL>)))                             (* stuff is a TCONC list of what to print)
          (if node:trans and node:trans:event#          (* print U15: at the beginning of the line, to show the
              then                                      event number when this command was given)

                    (TCONC stuff node:trans:event#)
                    (TCONC stuff ":"))
          (if PrintLine#s
              then (TCONC stuff ProofLine#))
          (if thm
              then (TCONC stuff (SELECTQ thm:status
                                         (proved "!")
                                         (assumed "%%")
                                         (untried "?")
                                         "|"))
            elseif ~(node : trans) and ~(True? node)
              then (TCONC stuff "?"))
          (if node=CurrentNode
              then (TCONC stuff "->"))
          (for p in stuff:1 do (printout NIL p])
```

105

```
(printCommand?
  [LAMBDA (node trans cmd? args?)                       (* D.Thompson "18-Mar-81 12:19")
    (if cmd? and trans:command and 'Implicit ~=(GetAnnotation node trans 'TYPE)
        then (printout NIL # (printName trans:command 'ProofDisplayCommand))
             (if trans:parameters and args?
                 then (if trans:command MEMB '(use apply)
                          then                          (* for lemmas, give by name, not as entered)
                               (printout NIL , (TheoremId trans:uses:1)
                                         ,)
                        else (for params on trans:parameters bind (previousAtom thisAtom)
                                finally (printout NIL ,) eachtime thisAtom←params:1
                                do (if (SeparateWithBlanks? previousAtom thisAtom)
                                       then (printout NIL ,))
                                   (PrettyPrint thisAtom T)
                                   (previousAtom←thisAtom])
```

106

```
(printAnnotation?
  [LAMBDA (note name)                                   (* D.Thompson "23-Jul-80 16:44")

          (* * This routine pretty-prints an annotation, if there is one. The second parameter is the name of the node having
          the annotation; if it's present, then the annotation is output as a command. Otherwise, the annotation is output as
          a comment.)


    (if note
        then (if name
                 then (printout NIL .TAB0 0 "annotate" , (if name=T
                                                             then ""
                                                           else (CONCAT name Comma Blank))
                                .PARA NIL -1 (MKLIST note)
                                CommandTerminator T)
               else (printout NIL LeftCurlyBracket .PARA NIL -1 (MKLIST note)
                               RightCurlyBracket ,,])
```

```
(printPropn
  [LAMBDA (node label)                                    (* D.Thompson "22-Jul-80 11:16")

          (* * This routine prints the name and expression for node, in the form of a series of AFFIRM commands.
          If label is given, it overrides.)


    (PROG (annotation nodeName object proved theorem)
          (node←(GetNode node))
          (theorem←(GetTheoremP node))
          (if theorem
              then proved←(Proved? theorem)
                   object←'theorem
            else proved←NIL
                 object←'propn)
          (annotation←(GetAnnotation node))
          (if label
              then nodeName←label
            else nodeName←(NodeId node)
                 (if (FIXP nodeName)
                     then nodeName←NIL))

          (* * exact command names are chosen as follows: -
          -
          ... named? ... proved? ... annotated? ... command names -
          ..... (or assumed) -
          ..... no ..... no ..... no ... theorem/propn -
          ..... no ..... no ..... yes ... try; annotate -
          ..... no ..... yes ..... no ... assume -
          ..... no ..... yes ..... yes ... try; assume; annotate -
          ..... yes ..... no ..... no ... theorem -
          ..... yes ..... no ..... yes ... theorem; annotate -
          ..... yes ..... yes ..... no ... assume -
          ..... yes ..... yes ..... yes ... assume; annotate)


          (if nodeName
              then                                        (* node is named; can print using "assume" or "theorem"
                                                          command.)
                   (printout NIL .TAB0 0 T (if proved
                                               then "assume"
                                              else object)
                             , nodeName Comma , # (PrettyPrint (NodeToExpr node)
                                                               T)
                             CommandTerminator T)
                   (printAnnotation? annotation nodeName)
            else                                          (* node is NOT named; must "try", if there's an
                                                          annotation.)
                 (printout NIL .TAB0 0 T (if annotation
                                             then "try"
                                            elseif proved
                                             then "assume"
                                            else object)
                           , LeftCurlyBracket (NodeId node)
                           Comma RightCurlyBracket , # (PrettyPrint (NodeToExpr node)
                                                                    T)
                           CommandTerminator T)
                 (if proved and annotation
                     then (printout NIL .TAB0 0 "assume" CommandTerminator T))
                 (printAnnotation? annotation T])
)

(RPAQQ PrintVars [PrintingParentSequence PrintingParentLine# PrintPropns ProofLine# PrintLine#s
                                         PrintingNamesSeen (Hyphens (APPLY (QUOTE CONCAT)
                                                                           (to 20 collect "-"])

(RPAQQ PrintingParentSequence NIL)

(RPAQQ PrintingParentLine# NIL)

(RPAQQ PrintPropns NIL)

(RPAQQ ProofLine# NIL)

(RPAQQ PrintLine#s NIL)

(RPAQQ PrintingNamesSeen NIL)

(RPAQ Hyphens (APPLY (QUOTE CONCAT)
```

```
                    (to 20 collect "-")))
[DECLARE: DONTEVAL@LOAD DONTCOPY
```

(* circularity tests)]

```
(RPAQQ CircleFns (CircularTransformP CircularChildP))
(DEFINEQ
```

108

```
(CircularTransformP
  [LAMBDA (transform source)                    (* R.Erickson "17-Apr-80 17:58")
                                                (* source is a p#)

        (* * We are about to apply transformation in the proof of source. Is this circular? If so, return a list of the
        names of offending theorems or nodes, ending in source.)


    (if ~transform
        then NIL
      elseif (for use in transform:uses bind cir do (if cir←(CircularChildP use source)
                                                        then (RETURN cir)))
      else (for child in transform:children bind cir do

        (* (if cir← (CircularChildP child source) then (* get label of child, if any) clabel← (if transform:labels and
        (CAR (NTH transform:labels (POS child transform:children))) else child) (RETURN (create Circularity circulartype ←
        (<'child clabel>) circularity ← cir))))

                                                        (if cir←(CircularChildP child source)
                                                            then (RETURN <child ! cir>])
```

109

```
(CircularChildP
  [LAMBDA (child source)                        (* R.Erickson " 8-Nov-79 21:12")
                                                (* both args are prop#s)

        (* * Source is a would-be parent of child (it wants to use it for its proof) Is this illegal? Does child really
        depend on source? If so, we return a list of theorem names along the way, ending in source.)


    (PROG (chthm chnode sorthm cir)

        (* * do case analysis on which of child, source are theorems)


        (chthm←(GetTheoremP child))
        (chnode←(GetNode child))
        (sorthm←(GetTheoremP source))
        (RETURN (if sorthm
                    then (if chthm
                             then

        (* * both thms: check identity, facts used by child.)

                                                                (if child=source
                                                                    then <(TheoremId chthm) >
                                                                  else (for use in (Facts chthm T)
                                                                          until cir←(CircularChildP use source) do)
                                                                       (if cir and (FactsApproxP chthm)
                                                                           then          (* oops! before we complain about a circularity, make
                                                                                         sure it didn't arise because of our lazy bookkeeping)
                                                                                cir←NIL
                                                                                (for use in (Facts chthm)
                                                                                   until cir←(CircularChildP use source) do))
                                                                       (if cir
                                                                           then <(TheoremId chthm) ! cir>))
                                                    else

        (* * only source a thm: check child's proof tree)

                                                            (CircularTransformP chnode:trans source))
                                                elseif (if chthm
                                                           then

        (* * Child is a theorem, source is not. This happens when we use a lemma. We could use brute force, descending into
        child's tree. Instead, we observe that source:parents gives us all theorems which directly depend on source.
        Therefore, if source appears below child, so will one of source:parents; this is easy to check.)
```

```
                                (for parent in (GetNode source):parents
                                   until cir←(CircularChildP child parent) do)
                                cir
                          else

          (* * neither a thm: check identity, and child's proof tree.)


                                (if child=source
                                   then <source>
                                  else (CircularTransformP chnode:trans source])
)

(PUTPROPS CurrentPropn GLOBALVAR T)

(PUTPROPS CurrentNode GLOBALVAR T)

(PUTPROPS CurrentTheorem GLOBALVAR T)

(PUTPROPS Truenode GLOBALVAR T)
(DECLARE: DONTCOPY
  (FILEMAP (NIL (1939 3153 (ActualExprAt 1951 . 2468) (CompressTransform 2472 . 2673) (GetLabel 2677 .
3150)) (3478 6608 (GetNode 3490 . 3803) (GetNodeP 3807 . 4446) (GetNode# 4450 . 4709) (ExprToNode 4713
 . 5383) (ExprToNode# 5387 . 5582) (NodeToExpr 5586 . 5746) (MakeNode 5750 . 5928) (NodeId 5932 . 6278
) (True? 6282 . 6605)) (6796 9847 (GetName 6808 . 7114) (NewNodeName 7118 . 7700) (SetName 7704 . 8651
) (DecodeName 8655 . 8943) (Names 8947 . 9137) (ClearNames 9141 . 9844)) (10093 12370 (GetChild 10105
. 11010) (GetChildNames 11014 . 11222) (GetTransformation 11226 . 11380) (GetOldTransformations 11384
. 11545) (CheckTransformationOccurs 11549 . 11892) (FlagArcDescent 11896 . 12367)) (12588 17900 (
GetTheorem 12600 . 12819) (GetTheoremP 12823 . 13270) (TheoremId 13274 . 13675) (NodeToThm 13679 .
13826) (ThmToNode 13830 . 13972) (Provees 13976 . 14337) (MostRecent 14341 . 14772) (SortRecency 14776
 . 15277) (GetTheorems 15281 . 15453) (UndertakeProof 15457 . 17065) (Restart 17069 . 17897)) (18098
22813 (AddUsage 18110 . 18706) (DropUsage 18710 . 19024) (Facts 19028 . 20025) (FactsApproxP 20029 .
20188) (UsedBy 20192 . 21243) (addRules 21247 . 21429) (dropRules 21433 . 21874) (addUsed 21878 .
22297) (dropUsed 22301 . 22810)) (22971 27814 (Assume 22983 . 23361) (Unassume 23365 . 23893) (Status
23897 . 24227) (PrettyStatus 24231 . 24514) (Proved? 24518 . 24884) (Finished? 24888 . 25115) (
UpdateStatus 25119 . 27811)) (28034 38023 (Arc 28046 . 28504) (SchemaCaseNames 28508 . 29956) (Try
29960 . 32645) (arc 32649 . 33024) (down 33028 . 33922) (name 33926 . 34391) (next 34395 . 36293) (
resume 36297 . 36841) (retry 36845 . 37217) (theorem 37221 . 37611) (up 37615 . 38020)) (38246 49487 (
Transform 38258 . 43580) (DeleteNodes 43584 . 44873) (DiscardTransformation 44877 . 45964) (
MakeTheorem 45968 . 47372) (UnMakeTheorem 47376 . 49097) (TransformToTrue 49101 . 49484)) (49712 53727
 (AddChildren 49724 . 50307) (AddChildren1 50311 . 51505) (RemoveChildren 51509 . 52024) (
RemoveChildren1 52028 . 52885) (MapChildren 52889 . 53308) (MapChildren1 53312 . 53724)) (53895 55291
(PCTopChain 53907 . 54129) (PCCurrNode 54133 . 54417) (PCMemb 54421 . 54658) (Descendant? 54662 .
55288)) (55293 58206 (Descend 55305 . 56633) (DescendAlongChain 56637 . 57042) (AscendTo 57046 . 57790
) (AscendBy 57794 . 58203)) (58208 59449 (ClearProof 58220 . 58571) (MakeCurrent 58575 . 59446)) (
59693 70541 (AddLeaf 59705 . 60288) (RemoveLeaf 60292 . 60859) (SortLeaves 60863 . 61241) (Next 61245
. 62816) (nextLemmas 62820 . 63795) (nextUsers 63799 . 65047) (unfinishedAncestors 65051 . 65701) (
NextLeaf 65705 . 66851) (orderofLeaves 66855 . 68283) (GetChainTo 68287 . 69466) (seekChainTo 69470 .
70074) (SeekArcLabelled 70078 . 70538)) (70604 72605 (Annotate 70616 . 71493) (GetAnnotation 71497 .
72202) (RestoreAnnotations 72206 . 72602)) (72817 84599 (PrintBoth 72829 . 73455) (PrintProof 73459 .
76056) (printProofNode? 76060 . 76287) (printProofNode 76291 . 79333) (printPrefix 79337 . 80357) (
printCommand? 80361 . 81247) (printAnnotation? 81251 . 82005) (printPropn 82009 . 84596)) (85206 88674
 (CircularTransformP 85218 . 86292) (CircularChildP 86296 . 88671)))))
STOP
```

```
(FILECREATED "11-Aug-80 21:34:35" <AFFIRM>TYPEPARAMETER..13 2186

     changes to:  TYPEPARAMETER)


(PRETTYCOMPRINT TYPEPARAMETERCOMS)

(RPAQQ TYPEPARAMETERCOMS ((P (CheckLoad (QUOTE TYPE)
                                        (QUOTE (110 . <AFFIRM>BASE-AFFIRM.EXE.54))
                                        (QUOTE TypeParameter)))
                          (FNS * TypeParameterFNS)
                          (FNS * TypeParameter\InterfaceFNS)
                          (VARS * TypeParameterConstants)
                          (VARS * TypeParameter\InterfaceConstants)
                          (IFPROP ALL * TypeParameterConstants)
                          (IFPROP (PrimaryLHSides EqualOp EQOP)
                                  * TypeParameterFNS)
                          (IFPROP (PrimaryLHSides EqualOp EQOP)
                                  * TypeParameter\InterfaceFNS)
                          (P (InitializeLoad TYPE TypeParameter 110 ((NoteInterfaces
                                                                      TypeParameter\InterfaceFNS)
                                                                     (InitInfix (QUOTE TypeParameter))
                                                                     (InitNeeds (QUOTE TypeParameter))
                                                                     (NoteDeclarations (QUOTE TypeParameter))
                                                                     (NoteLeftHandSides TypeParameterFNS))
(CheckLoad (QUOTE TYPE)
           (QUOTE (110 . <AFFIRM>BASE-AFFIRM.EXE.54))
           (QUOTE TypeParameter))

(RPAQQ TypeParameterFNS (Equal\TypeParameter))
(DEFINEQ

(Equal\TypeParameter
  (LAMBDA (x y)
    (if <'Equal\TypeParameter x y>))
)

(RPAQQ TypeParameter\InterfaceFNS NIL)
(DEFINEQ
)

(RPAQQ TypeParameterConstants (TypeParameter))

(RPAQQ TypeParameter TypeParameter)

(RPAQQ TypeParameter\InterfaceConstants NIL)
  (PUTPROPS TypeParameter IsConstant T
                          DeclaredType TypeParameter
                          LocalDeclarations NIL
                          Infix NIL
                          Needs NIL
                          EqualOp Equal\TypeParameter)

(RPAQQ TypeParameterFNS (Equal\TypeParameter))

(PUTPROPS Equal\TypeParameter EqualOp Equal\Boolean)

(PUTPROPS Equal\TypeParameter EQOP T)

(RPAQQ TypeParameter\InterfaceFNS NIL)
(InitializeLoad TYPE TypeParameter 110 ((NoteInterfaces TypeParameter\InterfaceFNS)
                (InitInfix (QUOTE TypeParameter))
                (InitNeeds (QUOTE TypeParameter))
                (NoteDeclarations (QUOTE TypeParameter))
                (NoteLeftHandSides TypeParameterFNS)))
(DECLARE: DONTCOPY
  (FILEMAP (NIL (1109 1211 (Equal\TypeParameter 1121 . 1208)) (1261 1272))))
STOP
```