*File created:* 28-Mar-89 11:14:41 **{NB:PARC:XEROX}<NOTECARDS>1.3M>LIBRARY>NCPATH.;1**

*previous date:* 4-Nov-86 11:41:58 {QV}<NOTECARDS>1.3L>LIBRARY>NCPATH>NCPATH.;1

*Read Table:* INTERLISP

*Package:* INTERLISP

*Format:* XCCS

;;


```
(RPAQQ NCPATHCOMS
       ((* * This package is intended to implement a path-description language for NoteCards. Note that Paths
          and Path&FSMs are sometimes confused.)
        (* * Path specifications are FSMs or Finite State Machines. They are implemented as lists of predicates
          to be applied to cards or links at present. FSMs and FSM-Nodes are also sometimes confused. Paths are
          implmented as lists of links. They represent a path through the NoteCards network of cards and links.
          A pointer to the appropriate node of an NCPathFSM is cons-ed onto the front of a path to make a
          PATH&FSM. Thus, each step of a path is a link. The Paths are stored as a tree structure, and each
          path shares cons cells with other paths. The paths are in reverse order and the root of the tree
          specifying all the paths is a notecard ID which was the starting point for the search.)
        (* * This list of functions implement the FSM. Note that there is alot of stuff in the other code that
          knows about FSMs and acts accordingly. In other words, this is not a true implementation of FSMs.)
        (FNS NCPath.FSM.PathCollect NCPath.FSM.RealPathCollect NCPath.FSM.FirstStep NCPath.FSM.RealFirstStep
             NCPath.FSM.ListFirstSteps NCPath.FSM.AddPotentialSteps NCPath.FSM.ListMultiplePaths
             NCPath.FSM.AddNextSteps NCPath.FSM.IncrementUseCount NCPath.FSM.AddStep NCPath.FSM.NextState
             NCPath.FSM.LoopLimitExceededP NCPath.FSM.AbsoluteDepthLimitExceededP)
        (FNS NCPath.FSMState.ComputeCollection NCPath.FSMState.ListNextSteps NCPath.FSMState.SpecifiesCardP
             NCPath.FSMState.SpecifiesLinkP NCPath.FSMState.TerminalP)
        (* * The second group of functions implements the collection data item. A collection is a list of paths
          or PATH&FSMs that share cons cells.)
        (FNS NCPath.Collection.ComputeNextCollection NCPath.FSM.ComputeMultilpeCollections
             NCPath.Collection.CollectMultiplePaths NCPath.Collection.ListRemovablePaths
             NCPath.Collection.ListFinishedPaths)
        (* * These functions implement the Path data structure.)
        (FNS NCPath.Path.Create NCPath.Path.End NCPath.Path.AddStep NCPath.Path.LastStep NCPath.Path.StepInPathP
             NCPath.Path.EQUAL NCPath.Path.LoopsP)
        (FNS NCPath.PathStep.End NCPath.PathStep.PotentialSteps NCPath.PathStep.MeetsFSMCardSpecificationP)
        (* * The last functions in this file are basically utilities that depend on implmentation details.)
        (FNS NCPath.Apply Copy.NCPathFSM Copy.NCPathFSMNode NCPath.Link.ListPotentialSteps
             NCPath.NoteCard.ListPotentialSteps NCPath.Link.GetCard)
        (* * Data types)
        (RECORDS NCPathFSM NCPathFSMNode NCPathPathStep)))
```

(* * This package is intended to implement a path-description language for NoteCards.
Note that Paths and Path&FSMs are sometimes confused.)

(* * Path specifications are FSMs or Finite State Machines. They are implemented as lists of predicates to be applied to cards or links at present. FSMs and FSM-Nodes are also sometimes confused.
Paths are implmented as lists of links. They represent a path through the NoteCards network of cards and links.
A pointer to the appropriate node of an NCPathFSM is cons-ed onto the front of a path to make a PATH&FSM.
Thus, each step of a path is a link. The Paths are stored as a tree structure, and each path shares cons cells with other paths. The paths are in reverse order and the root of the tree specifying all the paths is a notecard ID which was the starting point for the search.)

(* * This list of functions implement the FSM. Note that there is alot of stuff in the other code that knows about FSMs and acts accordingly. In other words, this is not a true implementation of FSMs.)

```
(DEFINEQ
```

## (NCPath.FSM.PathCollect

```
  [LAMBDA (PathSpec RootCard)                                    (* Newman " 4-Nov-86 10:08")
```

(* * This function collects a list of complete paths starting at RootCard as specified by PathSpec The paths are really a network, they share CONS cells, and are actually reversed. The end of each is a pointer to the ID for RootCard The first item in each path is actually the NCPathFSM representing the remaining parts of the path to be collected.
When the paths are complete, this is a NIL.)

```
    (COND
       ((NOT (NCP.ValidCard RootCard))
        (NCP.ReportError RootCard " is not an appropriate notecard.  Check the card and the notefile."))
       ((EQUAL (TYPENAME PathSpec)
               'NCPathFSM)
        (NCPath.FSM.RealPathCollect PathSpec RootCard))
       (T (NCP.ReportError " Illegal Argument to NCPath.FSM.PathCollect: " PathSpec " or " RootCard)
          NIL])
```

## (NCPath.FSM.RealPathCollect

```
  [LAMBDA (FSMInstance RootCard)                                 (* Newman " 4-Nov-86 10:17")
```

(* * This function does the real work of NCPath.FSM.PathCollect after that function has done the error checking.
(FinishedPaths has an extra NIL at the front, so the CDR at the end removes it))

```
    (bind (RemovablePaths _ NIL)
          (FinishedPaths _ (CONS))
          (UnFinishedPaths _ (NCPath.Collection.CollectMultiplePaths (NCPath.FSM.FirstStep RootCard FSMInstance)
                                                                     ))
       repeatwhile UnFinishedPaths do (SETQ RemovablePaths (NCPath.Collection.ListRemovablePaths UnFinishedPaths))
                                      (NCONC FinishedPaths (NCPath.Collection.ListFinishedPaths RemovablePaths))
                                      [SETQ UnFinishedPaths (NCPath.Collection.CollectMultiplePaths
                                                               (NCPath.Collection.ComputeNextCollection (LDIFFERENCE

                                                                                                          UnFinishedPaths

                                                                                                           RemovablePaths
                                                                                                              ]
       finally (RETURN (CDR FinishedPaths])
```

## (NCPath.FSM.FirstStep

```
  [LAMBDA (RootCard FSMInstance)                                    (* Newman "18-Mar-86 08:06")
```

(* * This function takes care of the case where the first NCPathFSMNode has a list as its NextState.)

```
    (replace (NCPathFSM LoopLimitAList) of FSMInstance with (LIST (CONS (fetch (NCPathFSM CurrentState)
                                                                           of FSMInstance)
                                                                    1)))
    (if (LISTP (NCPath.FSM.NextState FSMInstance))
        then (for NextState in (NCPath.FSM.NextState FSMInstance) bind (TempFSM _ (Copy.NCPathFSM FSMInstance))
                                                                      (TempFSMNode _ (Copy.NCPathFSMNode
                                                                                       (fetch (NCPathFSM
                                                                                                CurrentState)
                                                                                         of FSMInstance)))
              first (replace (NCPathFSM CurrentState) of TempFSM with TempFSMNode)
              eachtime (replace (NCPathFSMNode NextNodes) of TempFSMNode with NextState)
              join (NCPath.FSM.RealFirstStep RootCard TempFSM))
      else (NCPath.FSM.RealFirstStep RootCard FSMInstance])
```

## (NCPath.FSM.RealFirstStep

```
  [LAMBDA (RootCard FSMInstance)                                    (* Newman "18-Mar-86 07:43")
```

(* * This function is specially intended to get the first set of links from a path specification
(the NCPathFSM) and its root card. It constructs the first level or two of the tree of working paths.
The function handles the special case where the first two FSMNodes in NCPathFSM include card predicates.)

```
    (if (AND (NCPath.FSMState.SpecifiesCardP (NCPath.FSM.NextState FSMInstance))
             (NCPath.FSMState.SpecifiesCardP (fetch (NCPathFSM CurrentState) of FSMInstance)))
        then (for FSM in (NCPath.FSM.ListFirstSteps RootCard FSMInstance) join (NCPath.FSM.AddPotentialSteps FSM))
      else (NCPath.FSM.ListFirstSteps RootCard FSMInstance])
```

## (NCPath.FSM.ListFirstSteps

```
  [LAMBDA (RootCard FSMInstance)                                    (* Newman "21-Mar-86 15:59")
```

(* * This function gets the first set of links from a path specification and a root card.
End is bound specially so that all the paths created will share their last cons cells.)

```
    (bind (End _ (LIST RootCard)) for Link in (NCPath.FSMState.ListNextSteps RootCard (fetch (NCPathFSM
                                                                                               CurrentState)
                                                                                         of FSMInstance))
       collect (create NCPathFSM
                       InitialState _ (fetch (NCPathFSM InitialState) of FSMInstance)
                       CurrentState _ (NCPath.FSM.NextState FSMInstance)
                       Path _ (NCPath.Path.Create End Link FSMInstance)
                       LoopLimitAList _ (COPY (fetch (NCPathFSM LoopLimitAList) of FSMInstance))
                       AbsoluteDepthLimit _ (fetch (NCPathFSM AbsoluteDepthLimit) of FSMInstance])
```

## (NCPath.FSM.AddPotentialSteps

```
  [LAMBDA (FSMInstance)                                             (* Newman "19-Mar-86 14:08")
```

(* * Add all potential steps to Path&FSM without checking them against any specification.)

(* * Note that the expression for the Direction argument of the call to NCPath.Link.ListPotentialSteps is perhaps strange.
I think it should be (QUOTE BOTH) Randy and Frank think that this is the correct way to do things.)

```
    (for Link in (NCPath.Link.ListPotentialSteps (fetch (NCPathPathStep Link) of (NCPath.Path.LastStep
                                                                                    (fetch (NCPathFSM Path)
                                                                                      of FSMInstance)))
                                                 (fetch (NCPathPathStep Direction) of (NCPath.Path.LastStep (fetch (NCPathFSM Path)
                                                                                                                  of FSMInstance)))
                                                 (fetch (NCPathFSMNode Direction) of (fetch (NCPathFSM CurrentState) of FSMInstance)))
       collect (create NCPathFSM
                       InitialState _ (fetch (NCPathFSM InitialState) of FSMInstance)
                       CurrentState _ (fetch (NCPathFSM CurrentState) of FSMInstance)
                       Path _ (NCPath.Path.AddStep (create NCPathPathStep
                                                           Link _ Link
                                                           Direction _ (fetch (NCPathFSMNode Direction)
```

```
                                                                    of (fetch (NCPathFSM CurrentState)
                                                                       of FSMInstance)))
                                                  (fetch (NCPathFSM Path) of FSMInstance))
                                    LoopLimitAList _ (COPY (fetch (NCPathFSM LoopLimitAList) of FSMInstance))
                                    AbsoluteDepthLimit _ (fetch (NCPathFSM AbsoluteDepthLimit) of FSMInstance])
```

## (NCPath.FSM.ListMultiplePaths

```
[LAMBDA (FSMInstance)                                                    (* Newman "19-Mar-86 14:09")
```

(* * This function is intended to help out with the problem of true FSMs.
It takes a Path&FSM that has a list of FSMNodes as the CurrentState of the NCPathFSM and returns a list of Path&FSMs each of which has one of the list as its CurrentState.)

```
  (if (LISTP (fetch (NCPathFSM CurrentState) of FSMInstance))
      then (for Node in (fetch (NCPathFSM CurrentState) of FSMInstance)
              collect (create NCPathFSM
                              InitialState _ (fetch (NCPathFSM InitialState) of FSMInstance)
                              CurrentState _ Node
                              Path _ (fetch (NCPathFSM Path) of FSMInstance)
                              LoopLimitAList _ (COPY (fetch (NCPathFSM LoopLimitAList) of FSMInstance))
                              AbsoluteDepthLimit _ (fetch (NCPathFSM AbsoluteDepthLimit) of FSMInstance)))
    else
```

(* * Important Note%: This function always returns a list -
never just a single Path&FSM.)

```
      (LIST FSMInstance])
```

## (NCPath.FSM.AddNextSteps

```
[LAMBDA (FSMInstance)                                                    (* Newman "18-Mar-86 07:44")
```

(* * This function adds the next set of steps to a particular path.
That is, it takes a path, gets the NCPathFSM representing the PathSpec and the last link of the path, and adds each of the appropriate next steps to that path -
returning a list of several paths with common CONS cells. It also puts the next state of the NCPathFSM on the front of the Path for the next time around.)

```
  (for Link in (NCPath.FSMState.ListNextSteps (NCPath.Path.LastStep (fetch (NCPathFSM Path) of FSMInstance))
                                              (fetch (NCPathFSM CurrentState) of FSMInstance))
     collect (NCPath.FSM.AddStep Link FSMInstance])
```

## (NCPath.FSM.IncrementUseCount

```
[LAMBDA (FSMInstance)                                                    (* Newman "18-Mar-86 08:11")
```

(* * Incrment the count kept in the AList on the FSM indicating how many times the CurrentState has been used in this path.)

```
  (PUTASSOC (fetch (NCPathFSM CurrentState) of FSMInstance)
            (ADD1 (OR (CDR (ASSOC (fetch (NCPathFSM CurrentState) of FSMInstance)
                                  (fetch (NCPathFSM LoopLimitAList) of FSMInstance)))
                      0))
            (fetch (NCPathFSM LoopLimitAList) of FSMInstance])
```

## (NCPath.FSM.AddStep

```
[LAMBDA (Step FSMInstance)                                               (* Newman "19-Mar-86 14:10")
```

(* * Given an old FSMInstance and a new step, this function adds the step to the FSMInstance -
when NCPath.PathAddStep returns NIL. This function also increments the CurrentState to the NextState)

```
  (create NCPathFSM
          InitialState _ (fetch (NCPathFSM InitialState) of FSMInstance)
          CurrentState _ (NCPath.FSM.NextState FSMInstance)
          Path _ (NCPath.Path.AddStep (create NCPathPathStep
                                              Link _ Step
                                              Direction _ (fetch (NCPathFSMNode Direction)
                                                                 of (fetch (NCPathFSM CurrentState) of FSMInstance)))
                                      (fetch (NCPathFSM Path) of FSMInstance))
          LoopLimitAList _ (COPY (fetch (NCPathFSM LoopLimitAList) of FSMInstance))
          AbsoluteDepthLimit _ (fetch (NCPathFSM AbsoluteDepthLimit) of FSMInstance])
```

## (NCPath.FSM.NextState

```
[LAMBDA (FSMInstance)                                                    (* Newman " 4-Mar-86 13:30")
```

(* * This function is supposed to return the next state in NCPathFSM which defines a path specification.
If the CurrentState of NCPathFSM is NIL, we report the error and return NIL.)

```
  (if (NULL (fetch (NCPathFSM CurrentState) of FSMInstance))
      then (NCP.ReportError " NIL CurrentState of FSM in NCPath.FSM.NextState ")
    else (fetch (NCPathFSMNode NextNodes) of (fetch (NCPathFSM CurrentState) of FSMInstance])
```

## (NCPath.FSM.LoopLimitExceededP

```
[LAMBDA (FSMInstance)                                                    (* Newman "19-Mar-86 16:27")
```

(* * This predicate determines whether or not FSMNodeInstance has been used too many time to get to the current place in the path.)

```
    (AND [NOT (EQUAL 0 (fetch (NCPathFSMNode LoopLimit) of (fetch (NCPathFSM CurrentState) of FSMInstance]
         (ASSOC (fetch (NCPathFSM CurrentState) of FSMInstance)
                (fetch (NCPathFSM LoopLimitAList) of FSMInstance))
         (GEQ (CDR (ASSOC (fetch (NCPathFSM CurrentState) of FSMInstance)
                          (fetch (NCPathFSM LoopLimitAList) of FSMInstance)))
              (fetch (NCPathFSMNode LoopLimit) of (fetch (NCPathFSM CurrentState) of FSMInstance])
```

## (NCPath.FSM.AbsoluteDepthLimitExceededP

```
  [LAMBDA (FSMInstance)                                    (* Newman "19-Mar-86 15:45")
```

(* * This function checks to see if the absolute depth limit of the FSM has been exceeded by this path.)

```
    (AND (NOT (EQUAL 0 (fetch (NCPathFSM AbsoluteDepthLimit) of FSMInstance)))
         (GEQ (SUB1 (LENGTH (fetch (NCPathFSM Path) of FSMInstance)))
              (fetch (NCPathFSM AbsoluteDepthLimit) of FSMInstance])

)

(DEFINEQ
```

## (NCPath.FSMState.ComputeCollection

```
  [LAMBDA (FSMInstance)                                    (* Newman "19-Mar-86 14:13")
```

(* * This function takes a Path&FSM. The CurrentState of the NCPathFSM specifies a card rather than a link.
If the last step of the path meets specification, and the next node in the NCPathFSM specifies a link, the path is returned, If the last step of the path meets the specification, and the next node in the NCPathFSM specifies a card, the intervening links are added, and the list of new paths is returned.)

```
    (if (NCPath.PathStep.MeetsFSMCardSpecificationP (NCPath.Path.LastStep (fetch (NCPathFSM Path) of FSMInstance))
                (fetch (NCPathFSM CurrentState) of FSMInstance))
        then (if [AND (NCPath.FSMState.SpecifiesCardP (NCPath.FSM.NextState FSMInstance))
                      (NOT (NCPath.FSMState.TerminalP (NCPath.FSM.NextState FSMInstance]
                 then (NCPath.FSM.AddPotentialSteps FSMInstance)
                 else (LIST (create NCPathFSM
                                    CurrentState _ (NCPath.FSM.NextState FSMInstance)
                                    InitialState _ (fetch (NCPathFSM InitialState) of FSMInstance)
                                    Path _ (fetch (NCPathFSM Path) of FSMInstance)
                                    LoopLimitAList _ (COPY (fetch (NCPathFSM LoopLimitAList) of FSMInstance))
                                    AbsoluteDepthLimit _ (fetch (NCPathFSM AbsoluteDepthLimit) of FSMInstance])
```

## (NCPath.FSMState.ListNextSteps

```
  [LAMBDA (PathStepORCard FSMNodeInstance)                 (* Newman " 4-Nov-86 10:17")
```

(* * This function finds the next steps that can be taken from a particular point in a path.
It accepts either a card or a link as the indicator of the current path position and it returns a list of links.)

```
    (if FSMNodeInstance
        then (for Link in (COND
                            ((NCP.ValidCard PathStepORCard)
                             (NCPath.NoteCard.ListPotentialSteps PathStepORCard (fetch (NCPathFSMNode Direction)
                                                                                   of FSMNodeInstance)))
                            ((NCP.ValidLink (fetch (NCPathPathStep Link) of PathStepORCard))
                             (NCPath.PathStep.PotentialSteps PathStepORCard (fetch (NCPathFSMNode Direction)
                                                                             of FSMNodeInstance)))
                            (T (NCP.ReportError " PathStepORCard  not a link or card in NCPath.ListNextSteps ")
                               (SHOULDNT " Illegal argument in NCPath.ListNextSteps: PathStepORCard not a Card or
                                         Link ")))
                  when (NCPath.Apply FSMNodeInstance Link) collect Link)
      else (NCP.ReportError " NIL FSMNodeInstance in NCPath.ListNextSteps ")
           (SHOULDNT " Illegal argument in NCPath.ListNextSteps: NIL "])
```

## (NCPath.FSMState.SpecifiesCardP

```
  [LAMBDA (FSMState)                                       (* Newman " 4-Mar-86 12:43")
```

(* * This function returns T iff the FSM-State is not NIL and it's CARD/LINK flag indicates that the predicate is to be applied to a CARD.)

```
    (AND FSMState (NULL (fetch (NCPathFSMNode Card/Link) of FSMState])
```

## (NCPath.FSMState.SpecifiesLinkP

```
  [LAMBDA (FSMNodeInstance)                                (* Newman " 4-Mar-86 13:26")
```

(* * This function returns T iff the FSM-State is not NIL and the LINK/CARD flag indicates that the predicate is to be applied to a LINK.)

```
    (AND FSMNodeInstance (fetch (NCPathFSMNode Card/Link) of FSMNodeInstance])
```

## (NCPath.FSMState.TerminalP

```
  [LAMBDA (FSMNode)                                          (* Newman " 4-Mar-86 13:33")

          (* * This function determines whether or not a NCPathFSM node is a terminal node.
          That is, is there a transition to another node from this one.)

    (NULL FSMNode])

)


          (* * The second group of functions implements the collection data item.
          A collection is a list of paths or PATH&FSMs that share cons cells.)

(DEFINEQ
```

## (NCPath.Collection.ComputeNextCollection

```
  [LAMBDA (PathCollection)                                   (* Newman "18-Mar-86 08:08")

          (* * This function computes a new list of paths from an old one.
          The new paths are created by taking an old path and adding one step to it.
          This function also distinguishes the case where the next step is a card predicate rather than a link predicate.)

    (for FSMInstance in PathCollection eachtime (NCPath.FSM.IncrementUseCount FSMInstance)
       join (if (NCPath.FSMState.SpecifiesLinkP (fetch (NCPathFSM CurrentState) of FSMInstance))
                then (NCPath.FSM.AddNextSteps FSMInstance)
              elseif (NCPath.FSMState.SpecifiesCardP (fetch (NCPathFSM CurrentState) of FSMInstance))
                then (NCPath.FSM.ComputeMultilpeCollections FSMInstance)
              else (NCP.ReportError " FSMInstance   does not specify a Card or a Link in NCPath.ComputeCollection
")
                   (SHOULDNT " PathCollection not a list of FSMs "])
```

## (NCPath.FSM.ComputeMultilpeCollections

```
  [LAMBDA (FSMInstance)                                      (* Newman "18-Mar-86 07:47")

          (* * This function handles the case where the NextNodes of the CurrentState is a list rather than an individual FSMNode.)

          (* * This is analogous to the situation in NCPath.FSM.FirstStep)

    (if (LISTP (NCPath.FSM.NextState FSMInstance))
        then (for NextState in (NCPath.FSM.NextState FSMInstance) bind (TempFSMNode _ (Copy.NCPathFSMNode
                                                                                       (fetch (NCPathFSM
                                                                                                CurrentState)
                                                                                         of FSMInstance)))
                                                                  (TempFSMInstance _ (Copy.NCPathFSM
                                                                                       FSMInstance))
                  first (replace (NCPathFSM CurrentState) of TempFSMInstance with TempFSMNode)
                  eachtime (replace (NCPathFSMNode NextNodes) of TempFSMNode with NextState) join (
                                                                                    NCPath.FSMState.ComputeCollection
                                                                                      TempFSMInstance))
      else (NCPath.FSMState.ComputeCollection FSMInstance])
```

## (NCPath.Collection.CollectMultiplePaths

```
  [LAMBDA (Collection)                                       (* Newman "19-Feb-86 17:14")

          (* * This function takes a collection of Path&FSMs and expands those that have multiple FSMNodes as their CurrentState.)

    (for Path&FSM in Collection join (NCPath.FSM.ListMultiplePaths Path&FSM])
```

## (NCPath.Collection.ListRemovablePaths

```
  [LAMBDA (Collection)                                       (* Newman "19-Mar-86 14:03")

          (* * List those paths of Collection which are complete or loopy.)

    (for FSMInstance in Collection when (OR (NCPath.FSMState.TerminalP (fetch (NCPathFSM CurrentState)
                                                                         of FSMInstance))
                                            (NCPath.Path.LoopsP (fetch (NCPathFSM Path) of FSMInstance))
                                            (NCPath.FSM.LoopLimitExceededP FSMInstance)
                                            (NCPath.FSM.AbsoluteDepthLimitExceededP FSMInstance))
       collect FSMInstance])
```

## (NCPath.Collection.ListFinishedPaths

```
  [LAMBDA (Collection)                                       (* Newman "18-Mar-86 08:10")

          (* * List those paths in Collection that are complete as specified.
          These are the paths that the user wants to see.)

    (for FSMInstance in Collection when (NCPath.FSMState.TerminalP (fetch (NCPathFSM CurrentState) of FSMInstance)
                                        )
       collect (fetch (NCPathFSM Path) of FSMInstance])

)
```

```
(* * These functions implement the Path data structure.)

(DEFINEQ
```

## (NCPath.Path.Create

```
  [LAMBDA (Root FirstStepLink FSMInstance)                    (* Newman " 4-Mar-86 13:30")

          (* * This function creates a new path. Since a path is a list of NCPathPathStep in reverse order with a root card ID at the
          end, we create the first step and the root and cons them together.)

    (if (AND Root FirstStepLink)
        then (CONS (create NCPathPathStep
                           Link _ FirstStepLink
                           Direction _ (fetch (NCPathFSMNode Direction) of (fetch (NCPathFSM CurrentState)
                                                                                  of FSMInstance)))
                   Root)
      else (NCP.ReportError " NIL Root or FirstStepLink in NCPath.Path.Create. "])
```

## (NCPath.Path.End

```
  [LAMBDA (Path)                                              (* Newman "18-Mar-86 08:32")

          (* * This function returns the end car in a path.)

    (NCPath.PathStep.End (NCPath.Path.LastStep Path])
```

## (NCPath.Path.AddStep

```
  [LAMBDA (Step Path)                                         (* Newman "15-Mar-86 14:53")

          (* * Given Path and a new step, this function adds the step to the Path)

    (CONS Step Path])
```

## (NCPath.Path.LastStep

```
  [LAMBDA (Path)                                              (* Newman "21-Jan-86 13:01")

          (* * Since a Path is a reversed list, we just take the CAR to get the last step.)

    (CAR Path])
```

## (NCPath.Path.StepInPathP

```
  [LAMBDA (TestStep Path)                                     (* Newman " 4-Nov-86 11:30")

          (* * This predicate determines if TestStep is in Path or not.)

    (for PathStep in Path thereis (AND (NOT (NCP.ValidCard PathStep))
                                       (EQUAL (fetch (NCPathPathStep Direction) of TestStep)
                                              (fetch (NCPathPathStep Direction) of PathStep))
                                       (NC.SameLinkP (fetch (NCPathPathStep Link) of TestStep)
                                              (fetch (NCPathPathStep Link) of PathStep])
```

## (NCPath.Path.EQUAL

```
  [LAMBDA (Path1 Path2)                                       (* Newman " 4-Nov-86 11:30")

          (* * This function checks for equality of two paths that are still reversed, and have the root card at the end.
          It is significantly faster than EQUALALL, but uses a number of CONS cells.)

    (if [AND (EQUAL (LENGTH Path1)
                    (LENGTH Path2))
             (EQUAL (CAR (LAST Path1))
                    (CAR (LAST Path2]
        then (for Path1Step in (CDR (REVERSE Path1)) as Path2Step in (CDR (REVERSE Path2))
                always (AND (NC.SameLinkP (fetch (NCPathPathStep Link) of Path1Step)
                                          (fetch (NCPathPathStep Link) of Path2Step))
                            (EQUAL (fetch (NCPathPathStep Direction) of Path1Step)
                                   (fetch (NCPathPathStep Direction) of Path2Step])
```

## (NCPath.Path.LoopsP

```
  [LAMBDA (Path)                                              (* Newman "15-Mar-86 14:55")

          (* * This predicate returns T iff the last step in Path makes Path circular.)

    (NCPath.Path.StepInPathP (NCPath.Path.LastStep Path)
           (CDR Path])

)

(DEFINEQ
```

## (NCPath.PathStep.End

```
  [LAMBDA (PathStep)                                          (* Newman "18-Mar-86 08:33")
```

(* * This function returns the card at the appropriate end of PathStep, as indicated by the Direction field of the PathStep.)

```
(if (fetch (NCPathPathStep Direction) of PathStep)
    then (NCP.GetLinkDestination (fetch (NCPathPathStep Link) of PathStep))
  else (NCP.GetLinkSource (fetch (NCPathPathStep Link) of PathStep])
```

## (NCPath.PathStep.PotentialSteps

```
[LAMBDA (PathStep Direction)                                    (* Newman "18-Mar-86 07:46")
```

(* * This function computes the possible next links from the link in PathStep.)

```
(NCPath.Link.ListPotentialSteps (fetch (NCPathPathStep Link) of PathStep)
       (fetch (NCPathPathStep Direction) of PathStep)
       Direction])
```

## (NCPath.PathStep.MeetsFSMCardSpecificationP

```
[LAMBDA (PathStep FSMNode)                                      (* Newman " 4-Mar-86 13:37")
```

(* * This function determines whether or not the card specified by NCPathPathStep meets the specification of CardSpecification. Note that the specification is presently expected to be a Lisp predicate and that the card could be contained in either the Destination or the Source field of the link passed as PathStep.)

```
(if (NCPath.FSMState.SpecifiesCardP FSMNode)
    then (NCPath.Apply FSMNode (fetch (NCPathPathStep Link) of PathStep))
  else (NCP.ReportError " FSM-State does not specify a card in NCPath.PathStep.MeetsFSMCardSpecificationP "])

)
```

(* * The last functions in this file are basically utilities that depend on implmentation details.)

```
(DEFINEQ
```

## (NCPath.Apply

```
[LAMBDA (FSMNodeInstance Item)                                  (* Newman " 4-Nov-86 10:16")
```

(* * This function is anticipated to make the general path language easier to implement.
It will look at the NCPathFSMNode and it will use APPLY appropriately, else it will perform the right parsing and whatnot and then use apply.)

```
(SELECTQ (fetch (NCPathFSMNode Predicate) of FSMNodeInstance)
     (NIL (NCP.ReportError " NIL Predicate in SPECIAL-APPLY "))
     (* (TRUE))
     (APPLY (fetch (NCPathFSMNode Predicate) of FSMNodeInstance)
            (COND
               ((fetch (NCPathFSMNode Card/Link) of FSMNodeInstance)
                (LIST Item))
               (T
```

(* The cond following is different from that in NCPath.NoteCard.ListPotentialSteps because we are looking at the link from the other end.)

```
                  (LIST (COND
                           ((fetch (NCPathFSMNode Direction) of FSMNodeInstance)
                            (NCP.GetLinkDestination Item))
                           (T (NCP.GetLinkSource Item])
```

## (Copy.NCPathFSM

```
[LAMBDA (FSMInstance)                                           (* Newman "19-Mar-86 14:11")
```

(* * This function copies an NCPathFSM much faster than COPYALL does.
It should save time in NCPath.FSM.IncrementCurrentState.)

```
(create NCPathFSM
        CurrentState _ (fetch (NCPathFSM CurrentState) of FSMInstance)
        InitialState _ (fetch (NCPathFSM InitialState) of FSMInstance)
        Path _ (fetch (NCPathFSM Path) of FSMInstance)
        LoopLimitAList _ (COPY (fetch (NCPathFSM LoopLimitAList) of FSMInstance))
        AbsoluteDepthLimit _ (fetch (NCPathFSM AbsoluteDepthLimit) of FSMInstance])
```

## (Copy.NCPathFSMNode

```
[LAMBDA (FSMNodeInstance)                                       (* Newman "17-Mar-86 08:41")
```

(* * This function duplicats an NCPathFSMNode.)

```
(create NCPathFSMNode
        Predicate _ (fetch (NCPathFSMNode Predicate) of FSMNodeInstance)
        Card/Link _ (fetch (NCPathFSMNode Card/Link) of FSMNodeInstance)
        Direction _ (fetch (NCPathFSMNode Direction) of FSMNodeInstance)
        NextNodes _ (fetch (NCPathFSMNode NextNodes) of FSMNodeInstance)
        LoopLimit _ (fetch (NCPathFSMNode LoopLimit) of FSMNodeInstance])
```

## (NCPath.Link.ListPotentialSteps

```
  [LAMBDA (PreviousLink PreviousDirection Direction)            (* Newman "18-Mar-86 08:18")

          (* * This function takes a link and returns a list of all potential links that may be the next step in the path from that link.)

    (NCPath.NoteCard.ListPotentialSteps (NCPath.Link.GetCard PreviousLink PreviousDirection)
           Direction])
```

## (NCPath.NoteCard.ListPotentialSteps

```
  [LAMBDA (Card Direction)                                      (* Newman "17-Feb-86 15:05")

          (* * This function returns the potential links that might be of use from Card.)

    (SELECTQ Direction
         (BOTH (APPEND (NCP.GetLinks Card)
                       (NCP.GetLinks NIL Card)))
         (T (NCP.GetLinks Card))
         (NIL (NCP.GetLinks NIL Card))
         (NCP.ReportError " Direction neither T, NIL, or BOTH in NCPath.PotentialStepsFromCard. "])
```

## (NCPath.Link.GetCard

```
  [LAMBDA (PreviousLink Direction)                              (* Newman "12-Feb-86 15:05")

          (* * Given a link and a flag, this function decides which end of the link to return.)

    (COND
       (Direction (NCP.GetLinkDestination PreviousLink))
       (T (NCP.GetLinkSource PreviousLink])

)


          (* * Data types)

(DECLARE%: EVAL@COMPILE

(DATATYPE NCPathFSM ((InitialState POINTER)
                     (CurrentState POINTER)
                     (Path POINTER)
                     (LoopLimitAList POINTER)
                     (AbsoluteDepthLimit INTEGER))
       AbsoluteDepthLimit _ 0)

(DATATYPE NCPathFSMNode ((Predicate POINTER)
                         (Card/Link FLAG)
                         (Direction FLAG)
                         (NextNodes POINTER)
                         (LoopLimit INTEGER))
       Predicate _ (FUNCTION NILL)
       Card/Link _ T Direction _ T NextNodes _ NIL LoopLimit _ 1)

(DATATYPE NCPathPathStep ((Link POINTER)
                          (Direction FLAG)))
)

(/DECLAREDATATYPE 'NCPathFSM '(POINTER POINTER POINTER POINTER FIXP)
       ;; ---field descriptor list elided by lister---
       '10)

(/DECLAREDATATYPE 'NCPathFSMNode '(POINTER FLAG FLAG POINTER FIXP)
       ;; ---field descriptor list elided by lister---
       '6)

(/DECLAREDATATYPE 'NCPathPathStep '(POINTER FLAG)
       ;; ---field descriptor list elided by lister---
       '2)

(PUTPROPS NCPATH COPYRIGHT ("Xerox Corporation" 1986 1989))
```

## FUNCTION INDEX



## RECORD INDEX



## VARIABLE INDEX