

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

::
:: Copyright (c) 1986, 1989 by Xerox Corporation. All rights reserved.

(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
implemented 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 implementation 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 implemented 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 NCPPathFSM 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

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

(NCPPath.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 NCPPath.FSM.FirstStep)

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

(NCPPath.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 (NCPPath.FSM.ListMultiplePaths Path&FSM))

(NCPPath.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 (NCPPath.FSMState.TerminalP (fetch (NCPPathFSM CurrentState)
        of FSMInstance))
        (NCPPath.Path.LoopsP (fetch (NCPPathFSM Path) of FSMInstance))
        (NCPPath.FSM.LoopLimitExceededP FSMInstance)
        (NCPPath.FSM.AbsoluteDepthLimitExceededP FSMInstance))
        collect FSMInstance))

(NCPPath.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 (NCPPath.FSMState.TerminalP (fetch (NCPPathFSM CurrentState) of FSMInstance)
        )
        collect (fetch (NCPPathFSM Path) of FSMInstance))

)

```

(* 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 (NCPPathPathStep Direction) of PathStep)
  then (NCP.GetLinkDestination (fetch (NCPPathPathStep Link) of PathStep))
  else (NCP.GetLinkSource (fetch (NCPPathPathStep Link) of PathStep)))
```

(NCPPath.PathStep.PotentialSteps

[LAMBDA (PathStep Direction)

(* Newman "18-Mar-86 07:46")

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

```
(NCPPath.Link.ListPotentialSteps (fetch (NCPPathPathStep Link) of PathStep)
 (fetch (NCPPathPathStep Direction) of PathStep)
 Direction])
```

(NCPPath.PathStep.MeetsFSMCardSpecificationP

[LAMBDA (PathStep FSMNode)

(* Newman " 4-Mar-86 13:37")

(* This function determines whether or not the card specified by NCPPathPathStep 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 (NCPPath.FSMState.SpecifiesCardP FSMNode)
  then (NCPPath.Apply FSMNode (fetch (NCPPathPathStep Link) of PathStep))
  else (NCP.ReportError " FSM-State does not specify a card in NCPPath.PathStep.MeetsFSMCardSpecificationP ")))
```

)

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

(DEFINEQ

(NCPPath.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 NCPPathFSMNode and it will use APPLY appropriately, else it will perform the right parsing and whatnot and then use apply.)

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

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

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

(Copy.NCPPathFSM

[LAMBDA (FSMInstance)

(* Newman "19-Mar-86 14:11")

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

```
(create NCPPathFSM
 CurrentState _ (fetch (NCPPathFSM CurrentState) of FSMInstance)
 InitialState _ (fetch (NCPPathFSM InitialState) of FSMInstance)
 Path _ (fetch (NCPPathFSM Path) of FSMInstance)
 LoopLimitList _ (COPY (fetch (NCPPathFSM LoopLimitList) of FSMInstance))
 AbsoluteDepthLimit _ (fetch (NCPPathFSM AbsoluteDepthLimit) of FSMInstance))
```

(Copy.NCPPathFSMNode

[LAMBDA (FSMNodeInstance)

(* Newman "17-Mar-86 08:41")

(* This function duplicates an NCPPathFSMNode.)

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

(NCPPath.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.)

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

(NCPPath.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 NCPPath.PotentialStepsFromCard. "])
```

(NCPPath.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 NCPPathFSM ((InitialState POINTER)
  (CurrentState POINTER)
  (Path POINTER)
  (LoopLimitAList POINTER)
  (AbsoluteDepthLimit INTEGER))
  AbsoluteDepthLimit _ 0)
```

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

```
(DATATYPE NCPPathPathStep ((Link POINTER)
  (Direction FLAG)))
```

)

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

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

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

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

FUNCTION INDEX

Copy.NCPathFSM	7	NCPath.FSM.RealPathCollect	1
Copy.NCPathFSMNode	7	NCPath.FSMState.ComputeCollection	4
NCPath.Apply	7	NCPath.FSMState.ListNextSteps	4
NCPath.Collection.CollectMultiplePaths	5	NCPath.FSMState.SpecifiesCardP	4
NCPath.Collection.ComputeNextCollection	5	NCPath.FSMState.SpecifiesLinkP	4
NCPath.Collection.ListFinishedPaths	5	NCPath.FSMState.TerminalP	4
NCPath.Collection.ListRemovablePaths	5	NCPath.Link.GetCard	8
NCPath.FSM.AbsoluteDepthLimitExceededP	4	NCPath.Link.ListPotentialSteps	8
NCPath.FSM.AddNextSteps	3	NCPath.NoteCard.ListPotentialSteps	8
NCPath.FSM.AddPotentialSteps	2	NCPath.Path.AddStep	6
NCPath.FSM.AddStep	3	NCPath.Path.Create	6
NCPath.FSM.ComputeMultilpeCollections	5	NCPath.Path.End	6
NCPath.FSM.FirstStep	2	NCPath.Path.EQUAL	6
NCPath.FSM.IncrementUseCount	3	NCPath.Path.LastStep	6
NCPath.FSM.ListFirstSteps	2	NCPath.Path.LoopsP	6
NCPath.FSM.ListMultiplePaths	3	NCPath.Path.StepInPathP	6
NCPath.FSM.LoopLimitExceededP	3	NCPath.PathStep.End	6
NCPath.FSM.NextState	3	NCPath.PathStep.MeetsFSMCardSpecificationP	7
NCPath.FSM.PathCollect	1	NCPath.PathStep.PotentialSteps	7
NCPath.FSM.RealFirstStep	2		

RECORD INDEX

NCPathFSM	8	NCPathFSMNode	8	NCPathPathStep	8
-----------------	---	---------------------	---	----------------------	---

VARIABLE INDEX

NCPATHCOMS	1
------------------	---
