NCPath

By: David Newman (Newman.pasa@Xerox.COM)

NCPathParse, NCPathUse

INTRODUCTION

This file is a module designed to allow a programmer to specify a path through a notecards network and a root card, and obtain from these a data structure describing all the paths through the network from the root card that match the path specification. The function NCPath.FSM.PathCollect is the top level function. In conjunction with NCPathParse (which translates a simple path language into the path descriptor that NCPath.FSM.PathCollect requires), it returns a collection of paths for the programmer to manipulate. A typical call to the NCPath functions should look like the following.

```
(NCPath.FSM.PathCollect (NCPathParse <Path-description> ) <RootCard>)
```

This package and its functions are implemented on a bastardized finite state machine model. The key difference is that in order to allow the imposition of limits on loops in the finite state network (limits on the number of times a loop can be traversed), the number of times each loop is traversed must be recorded in the finite state machine. In addition, this system will not collect any paths which attempt to traverse the same link of a notefile in the same direction a second time.

The data structures of this system include the InterLisp datatypes NCPathFSM, NCPathFSMNode, and NCPathPathStep, and two list structures. A collection is a list of paths. A path is a list of pathsteps with the root card at the end. Paths contain the steps taken in reverse order, and the various paths in a collection share cons cells in order to conserve memory. Please note also the descriptions of the three records listed below.

RECORDS

NCPathFSM [Datatype]

The fields of an NCPathFSM are InitialState, CurrentState, Path, and LoopLimitAList. Each is a Lisp POINTER. InitialState and CurrentState contain objects of type NCPathFSMNode, and are the current and initial states of the finite state machine respectively. The Path field contains the intermediate

results of a traversal through the NoteCards network. Finally, LoopLimitAList is an a-list that keeps track of the number of times any loop in the finite state transition network has been traversed.

NCPathFSMNode [Datatype]
The fields of an NCPathFSMNode are Predicate, Card/Link, Direction, NextNodes, and LoopLimit. The first and fourth are POINTERs, the second and third are FLAGs, and the last is an INTEGER. The Predicate field points to a function or LAMBDA expression, which when applied to a notecard or link returns T or NIL; this indicates whether the item in question will satisfy the next step in the path description. The Card/Link flag indicates whether the Predicate is to be applied to links or cards, (T indicates links and is the default, while NIL indicates cards), and the Direction flag indicates whether a link is to be followed in a forward direction or a backward direction (T indicates forward, and is the default setting). NextNodes is a list of the next nodes in the finite state transition network, and hence is essentially a list of arcs away from the current node. Lastly, LoopLimit indicates the number of times that a loop beginning with the current node may be traversed; its default value is one.

NCPathPathStep [Datatype]
An NCPathPathStep has only two fields, both of which are POINTERS. the Link field contains a link in the notecards network, and the Direction field indicates the direction that this link was followed.

FUNCTIONS

NCPath.FSM.PathCollect *PathSpec RootCard*) [Function]
This function collects a list of complete paths starting at RootCard as specified by PathSpec The paths are really a network as 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.

(NCPath.FSM.RealPathCollect *FSMInstance RootCard*) [Function]
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)

(NCPath.FSM.FirstStep *RootCard FSMInstance*) [Function]
This function takes care of the case where the first NCPathFSMNode has a list as its NextState.

(NCPath.FSM.RealFirstStep *RootCard FSMInstance*) [Function]
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.

(NCPath.FSM.ListFirstSteps *RootCard FSMInstance*) [Function]
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.

(NCPath.FSM.AddPotentialSteps *FSMInstance*) [Function]
Add all potential steps to the Path in *FSMInstance* without checking them against any specification. Returns a list of NCPathFSMs

(NCPath.FSM.ListMultiplePaths *FSMInstance*) [Function]
This function is intended to help out with the problem of true FSMs. It takes a NCPathFSM that has a list of FSMNodes as its CurrentState, and returns a list of NCPathFSMs each of which has one of the FSMNodes as its CurrentState.

(NCPath.FSM.ComputeMultilpeCollections *FSMInstance*) [Function]
This function handles the case where the NextNodes field of the NCPathFSMNode in the CurrentState field of *FSMInstance* is a list rather than an individual FSMNode.

(NCPath.FSM.AddNextSteps *FSMInstance*) [Function]
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.

(NCPath.FSM.IncrementUseCount *FSMInstance*) [Function]
Increment the count kept in the AList on the FSM indicating how many times the CurrentState has been used in this path.

(NCPath.FSM.AddStep Step *FSMInstance*) [Function]
Given an old FSMInstance and a new step, this function adds the step to the path in the FSMInstance. This function also increments the CurrentState to the NextState

(NCPath.FSM.NextState *FSMInstance*) [Function]
This function is returns the next state in *FSMInstance* which defines a path specification. If the CurrentState of *FSMInstance* is NIL, we report the error and return NIL.

(NCPath.FSM.LoopLimitExceededP *FSMInstance*) [Function]
This predicate determines whether or not the FSMNodeInstance stored in the CurrentState of *FSMInstance* has been used too many time to get to the current place in the path. This function determines if a limited loop has been followed too many times.

(NCPath.FSMState.ComputeCollection *FSMInstance*) [Function]
This function takes an NCPathFSM as its single argument. 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.

(NCPath.FSMState.ListNextSteps *PathStepORCard FSMNodeInstance*) [Function]

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.

(NCPath.FSMState.SpecifiesCardP *FSMState*) [Function]
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.

(NCPath.FSMState.SpecifiesLinkP *FSMNodeInstance*) [Function]
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.

(NCPath.FSMState.TerminalP *FSMNode*) [Function]
This function determines whether or not a NCPathFSM node is a terminal node. That is, whether or not there is a transition to another node from this one.

(NCPath.Collection.ComputeNextCollection *PathCollection*) [Function]
This function computes a new list of FSMs from an old one. The new FSMs are created by taking an old FSM and adding one step to its path. This function also distinguishes the case where the next step is specified as a card predicate rather than a link predicate.

(NCPath.Collection.CollectMultiplePaths *Collection*) [Function]
This function takes a collection of NCPathFSMs and expands those that have multiple FSMNodes as their CurrentState into a list of NCPathFSMs each having one FSMNode as its current state.

(NCPath.Collection.ListRemovablePaths *Collection*) [Function]
List those paths of Collection which are complete, or loopy, or which have traversed the same loop of the FSM too many times.

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

(NCPath.Path.Create *Root FirstStepLink FSMInstance*) [Function]
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.

(NCPath.Path.End *Path*) [Function]
This function returns the end card in a path by taking the last step of the path and using the Direction field to determine whether to return the source or destination of the link in the Link field.

(NCPath.Path.AddStep *Step Path*) [Function]
Given Path and a new step, this function adds the step to the Path

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

(NCPath.Path.StepInPathP *TestStep P*ath) [Function]
This predicate determines *if TestS*tep is *in P*ath or not.
(NCPath.Path.EQU*AL Path1 P*ath2) [Function]
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.
(NCPath.Path.Loop*sP P*ath) [Function]
This predicate returns T iff the last step in Path makes Path circular.
(NCPath.PathStep.E*nd PathS*tep) [Function]
This function returns the card at the appropriate end of PathStep, as indicated by the Direction field of the PathStep.
(NCPath.PathStep.PotentialSte*ps PathStep Direct*ion) [Function]
This function computes the possible next links from the link in PathStep.
(NCPath.PathStep.MeetsFSMCardSpecificatio*nP PathStep FSMN*ode) [Function]
This function determines whether or not the card specified *by PathS*tep meets the specification *of FSMN*ode.
(NCPath.App*ly FSMNodeInstance I*tem) [Function]
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.
(Copy.NCPathF*SM FSMInsta*nce) [Function]
This function copies an NCPathFSM much faster than COPYALL does. It should save time in NCPath.FSM.IncrementCurrentState.
(Copy.NCPathFSMNo*de FSMNodeInsta*nce) [Function]
This function duplicates an NCPathFSMNode.
(NCPath.Link.ListPotentialSte*ps PreviousLink PreviousDirection Direct*ion) [Function]
This function takes a link, the direction it was followed, and another direction flag, and returns a list of all potential links that may be the next step in the path from the appropriate end of the Link in the. The second Direction flag determines which way the found links are followed.
(NCPath.NoteCard.ListPotentialSte*ps Card Direct*ion) [Function]
This function returns the potential links that might be of use from Card.
(NCPath.Link.GetC*ard PreviousLink Direct*ion) [Function]
Given a link and a flag, this function decides which end of the link (source card or destination card) to retu