NCPathParse

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

NCPath, NCPathUse

INTRODUCTION

This package parses expressions of a simple path description language into NCPathFSM data structures for use by NCPath.FSM.PathCollect. The language is described by a BNF grammar in the file NCPathLanguage.TEdit. A simpler description of the language follows.

PATH DESCRIPTION LANGUAGE

The basic units of the language are path step descriptors. Each descriptor is either a literal descriptor or a functional descriptor. The literal descriptors consist of link types and card types currently existing in the active notefile. Functional descriptors are the names of lisp predicates that can be applied to a link or card, and which return T or NIL. A proposed new step in a path through a notefile is accepted when it is of the type specified by a literal descriptor or when application of the function specified by a functional descriptor returns T. Each path step descriptor has an optional prefix of several characters. The character @ indicates that the descriptor specifies the card at the end of the path step. The character # prefaces all functional descriptors. The character _ indicates that links followed in obtaining this step should be followed backwards. Path descriptors can be combined into other descriptors by the use of AND, OR, and NOT subject to the restriction that all descriptors in a combination must either apply to cards or links. Lastly, repetitive path structures may be specified using a number or Kleene star notation. See the examples below.

EXAMPLES

FOO is a function name, GOODP is a function name, and BAR is a variable with value @#GOODP.

```
(NCPathParse (QUOTE #FOO))          (NCPathParse (QUOTE @#FOO))
(NCPathParse (QUOTE _@#FOO))        (NCPathParse (QUOTE BAR))
(NCPathParse (QUOTE @Text))         (NCPathParse (QUOTE SubBox))
```

```
(NCPathParse (QUOTE (SubBox)))      (NCPathParse (QUOTE (@#FOO)))
(NCPathParse (QUOTE (SubBox FiledCard)))
(NCPathParse (QUOTE (@#FOO @#FOO))) (NCPathParse (QUOTE (AND @#FOO BAR)))
(NCPathParse (QUOTE (OR @#FOO (AND @Text BAR))))
(NCPathParse (QUOTE (#FOO *3)))     (NCPathParse (QUOTE ((#FOO #FOO) *3)))
```

FUNCTIONS

NCPathParse *Expression*) [Function]
This function is intended to be the top-level function that parses an expression into an FSM that NCPath.FSM.PathCollect can deal with.

(NCPathParse.FSMFromPathSpec *Expression*) [Function]
This function is intended to be the top-level function that parses an expression into an FSM that NCPath.FSM.PathCollect can deal with.

(NCPathParse.Path *Expression*) [Function]
This function parses an individual path as a list of steps or repeater expressions.

(NCPathParse.PathStep *Expression*) [Function]
This function parses an individual step of a path. A step is either a path step descriptor or some combination of descriptors.

(NCPathParse.PathStepDescriptor *Expression*) [Function]
Parses a path step descriptor. A descriptor is a literal descriptor, a functional descriptor, a "don’t care" expression, or a variable pointing to a descriptor of one of the other types. The variable is checked for immediate circularity, but not for indirect circularity; either of these conditions could cause an infinite loop.

(NCPathParse.LiteralPathDescription *Expression*) [Function]
This function parses path descriptors that literally describe what kind of object qualifies as a step in the path sought through the notecards network.

(NCPathParse.FunctionalPathDescription *Expression*) [Function]
This function parses path descriptors that give function names which act as predicates to determine whether a link or card matches this step in the path descriptor.

(NCPathParse.CreateFSMNode *Item Link/CardFlag DirectionFlag FunctionFlag*) [Function]
This function creates the NCPathFSMNodes that will become part of the NCPathFSM being created by NCPathParse. It contains a call to the code which creates the LAMBDA expressions to be used as predicates if necessary.

(NCPathParse.CreatePredicateForm *Type Link/CardFLAG*) [Function]
This function creates a LAMBDA expression that will serve as a predicate. See NCPathParse.CombinePredicates.

(NCPathParse.RepeaterExpression *RepeaterExpression LoopSteps*) [Function]
This function parses repeater expressions.

(NCPathParse.RepeaterToken *RepeaterExpression LoopSteps*) [Function]
Creates a repeater loop with no limit, or with infinite limit

(NCPathParse.LimitedRepeaterToken *RepeaterExpression LoopSteps*) [Function]
This function parses repeater tokens that have an integral limit.

(NCPathParse.LoopDecision *Expression LoopSteps MinimumRepeat Symbol*) [Function]
This function decides what kind of loop is to be created (requiring one or zero pases through the loop) and then creates a description of that loop.

(NCPathParse.CreateLoop *LoopSteps MinTimes MaxTimes*) [Function]
Here we create an expression describing a loop that NCPathParse.CoalesceStates can transform into an actual loop.

(NCPathParse.PathStepOperation *Expression*) [Function]
Here we parse combinations of pathsteps. Combinations are created using AND, OR, and NOT, to combine other pathsteps.

(NCPathParse.FunctionP *Function*) [Function]
This function returns T if its argument names an appropriate function for use as a predicate in an NCPathFSMNode.

(NCPathParse.CheckAndComputeFlag *StepList FlagName*) [Function]
This function computes the Direction and Card/Link flags when parsing a combination FSMNode. It also determines if the flags involved match, and notifies the user if they do not. It returns NIL if the flags do not match.

(NCPathParse.CombinePredicates *StepList Operation*) [Function]
This function builds the LAMBDA expression that will be the predicate in a combination FSMNode.

(NCPathParse.CombinationExpressions *StepList Operation*) [Function]
This function combines pathstep specifications using AND, OR, or NOT into a single NCPathFSMNode.

(NCPathParse.CoalesceStates *NodeList*) [Function]
This function takes a lisp-style list of NCPathFSMNodes and turns them into a linked list. The linked list will include circularities where repeater expressions exist.

(NCPathParse.CoalesceRepeaterStates *LoopList Next* Limit) [Function]
This function creates the circular linked list structure that repeater expressions need.

(LOGEQUAL . *U*) [Function]
This function is a logical operator. It determines if the arbitrary number of arguments are logically equal or not.

(NAND . *Args*) [Function]
This function is a logical operator similar to AND or OR, except that it performs the logical operation of `(NOT (AND Args))`.