1988

A Visual aide for designing regular expression parsers

Norman C. Crowfoot

A Visual Aide for Designing Regular Expression Parsers

by

Norman C. Crowfoot

Rochester Institute of Technology

A Visual Aide for

Designing

Regular Expression Parsers

by

Norman C. Crowfoot

Rochester Institute of Technology

Abstract

This paper describes a thesis project in which a visually-oriented design utility is constructed in Interlisp-D for the Xerox 1108 Artificial Intelligence Workstation. This utility aids in the design of Regular Expression Parsers by visually simulating the operation of a parser. A textual program, suitable for utilization in the construction of a compiler scanner or other similar processor may be produced by the utility.

Approvals: Peter A. Lutz

_____

_____

Table of Contents

1. Introduction and Background

A software productivity crisis clearly exists in the computer science field and the computer industry as a whole. Computer software simply costs far too much. It costs too much to create, too much to maintain and too much to document.

The roots of this productivity problem lie deep within the nature of software itself, and are largely due to software's inherent complexity. The many causes and potential cures for this problem have been explored and explained by Fred Brooks [Brooks-75, Brooks-87].

Brooks explains that significant progress has been made through the invention of high-level languages. However recently, far too many expectations have been placed in seeking radical, almost magical technological solutions. He concludes that there simply are no magic potions, or "silver bullets", left in the programming arsenal.

> "Not only are there no silver bullets now in view, but there very nature of software makes it unlikely that there will ever be any--no inventions are likely to do for software productivity, reliability, and simplicity what electronics, transistors, and large-scale integration did for computer hardware. We cannot expect ever to see twofold gains every two years."
>
> -- [Brooks-87]

The goal of this thesis is to design, produce and demonstrate a graphic design tool for the design of regular expression scanners to demonstrate the productivity improvements possible with a suitably integrated design tool. The contention is that graphic design and debugging is more productive since it is more intuitive to the designer.

While this is only a limited problem domain, it is hoped that sufficient improvement can be demonstrated by this work to justify further work in allied problem areas.

1.1. Problem Statement

The design of a compiler is a non-trivial task, especially for the beginning student. While many tools are available to assist in many stages of the process, the design and checkout of a lexical scanner may be difficult.

Figure 1 details the process of designing the lexical scanner. Given the Backus Naur Form (BNF) statements describing the language, the designer converts these into a

series of regular expressions. These regular expressions are then input to a scanner generator program, such as LEX, which produces the scanner code itself. This code is then compiled and linked to form the runnable scanner. [Lesk-75].

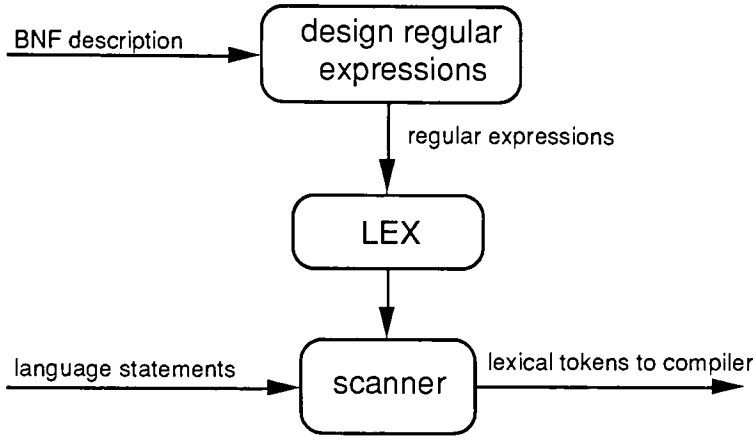

Figure 1: Lexical Parser Design Process

While the LEX processor simplifies much of the work, there still remains a problem for the student to understand how the abstract regular expressions function. The Visual Design Aide assists the student designer to visualize and understand the interactions between a set of regular expressions.

Lexical scanner designers translate BNF into a set of regular expressions. These regular expressions are then input to a scanner generator program. The problem is that the regular expressions are simply character strings; a left-to-right, top-to-bottom string of characters without structure.

Analyzing defects in regular expression strings and understanding their operation is made difficult by several factors.

(1) Scanner generator programs, such as LEX, produce relatively few error and diagnostic messages. And the messages that are produced are generally concerned with only minor syntactical defects.

(2) These scanner generators attempt to successfully produce a scanner, regardless of ambiguities in the input grammar [Aho-77]. While this approach frequently produces a running scanner, it may fail to implement the designer's intentions.

(3) Debugging scanners produced by generators is difficult due to the processing steps involved. These steps tend to remove the designer from the actual operation of the scanner. For instance, the generator may create new states, merge others and delete still others. Thus it may be difficult for the novice designer to determine what happened, or even where it happened.

This debugging problem may be further complicated by use of different source languages between implementation of the compiler itself and the program produced by the scanner generator.

1.2. Theoretical and Conceptual Development

1.2.1. Regular Expressions

Given a finite set of tokens, or an *alphabet* Σ, the *regular expressions over* Σ and the regular sets they *denote* are defined recursively, in the following manner:

(1) $\varnothing$ (empty set) is a regular expression denoting the regular set $\varnothing$.

(2) ε is a regular expression denoting the regular set $\{ \varepsilon \}$. This expression signifies an "empty transition", or identity expression.

(3) a in Σ is a regular expression denoting the regular set $\{a\}$.

(4) If p and q are regular expressions denoting the regular sets P and Q respectively, then

(a) $(p + q)$ is a regular expression denoting $P \cup Q$, denoting *union*.

(b) (pq) is a regular expression denoting PQ, denoting *concatenation*.

(c) $(p)^*$ is a regular expression denoting P^*, denoting *closure*[1].

(5) nothing else is a regular expression

In other words, regular expressions are composed of recursive application of union, concatenation and closure over a finite alphabet of input symbols.

---

1 The *closure* of a set L, denoted by L^*, is the set $L^* = \{\varepsilon\} \cup L \cup LL \cup LLL...$

Therefore

ALPHA	denotes the set {ALPHA}
(0+1)*011	denotes the regular set of all character strings composed of ones and zeros, ending in 011
AB(C+D)	denotes the regular set {ABC, ABD}.

Clearly the cardinality of regular expressions, as well as their generated sets, is infinite. Further, for each regular set, at least one regular expression may be written. And each regular expression denotes one and only one regular set [Aho-72].

1.2.2. Character Classes

A shorthand notation, a character class, may be specified between the metacharacters "[" and "]". Thus

$$[ABCD] = (A + B + C + D)$$

Further large classes of contiguous characters may be indicated by specifying the initial and final characters in the class, separated by a dash, as

$$[A\text{-}D] = [ABCD] = (A + B + C + D)$$

and

$$[\,A - Z\,] = (A + B + C + \ldots + X + Y + Z)$$

provided that the ordinal value of A is less then the value of Z.[1]

This notation is commonly used in specifying large alphabets, such as:

$$[A\text{-}Za\text{-}z] = ([A\text{-}Z] + [a\text{-}z])$$

---

[1] This notation assumes alphabetic characters are contiguous in the character set, as is the case in ASCII. The EBCDIC character code assigns the additional codes X'CA':X'D0' and X'DA':X'E1' as alphabetic characters, since they are between 'A' and 'Z'. However, since these additional codes are reserved for "national" characters and have no graphic characters assigned to these codes, little harm is done by allowing this generality [IBM-64].

1.2.3. Finite State Automata

In the classic sense, a Finite State Automaton (FSA) is a recognizer of regular expressions. The formal definition of an FSA is a 5-tuple:

(1) Q, a finite set of *states*;

(2) Σ, a finite set of permissible input symbols, or the *alphabet*

(3) a mapping of Q crossed with the alphabet generating a set of new FSA states. This is the *state transition function*. If this transition function is everywhere single-valued, the FSA is said to be *deterministic*, otherwise the FSA is said to be *non-deterministic*;

(4) a distinguished *starting state*;

(5) a set of one or more *final* or *accepting states*.

The FSA functions by making *moves* from the starting state to an accepting state, moving through the input symbols and applying the state transition function. When the FSA enters an accepting state and the input string has been exhausted, the FSA is said to have *accepted* or *recognized* the input symbols as a regular expression [Aho-72].

1.2.4. Nondeterminism Versus Determinism

Further the FSA is said to be *deterministic* if the state transition function has a single move, wherever it is defined. That is, for all states and input symbols, the state transition function generates no more than one successor state.

If instead of a state transition function, the FSA has a state transition *relation* then the FSA is said to be *nondeterministic*. This should not be confused with randomness, as the FSA *does not* choose a successor state, according to some probability. Rather a nondeterministic FSA logically goes to *all* successor states. This can be thought of as the logical replication of the FSA into multiple *instantiations*, one for each possible successor state [Aho-72].

FSA instantiations die if a valid transition from a state is not possible. A regular expression is recognized when any FSA instantiation enters an accepting state and all input symbols are exhausted.

1.2.5. Graphic Representation of Finite State Automata

There is a direct translation of an FSA to a graphic representation. Each state or node of the FSA is drawn as a closed circle. Moves defined by the state transition function are drawn as directed arcs from one state to state.

Special nodes are indicated with additional circles. Starting nodes are indicated by a small inner circle; accepting states are shown with a larger inner circle. (It is possible for a single node to be both a starting and an ending node.) Nodes may also be optionally labeled with a name.

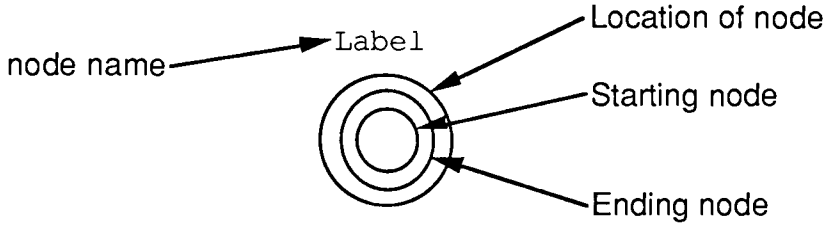

Figure 2: Graphic Representation of a Node

Transition arcs may be labeled with the character or character classes that when present allow the transition to occur. Arcs may be unlabeled, indicating no input character is required for the transition to be taken.

In Figure 3, a transition from node A to node B is possible on any alphabetic character and from node A to node C on any numeric character. Since no symbol is depicted on the transition arc to node D, a transition from node A to node D may occur at any time, without consuming an input symbol.

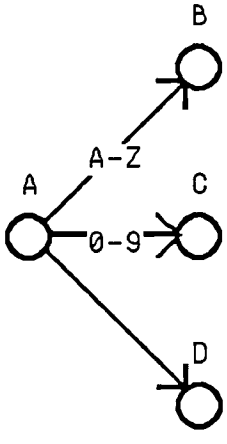

Figure 3: Graphic Representation of an FSA

1.2.6. Representation Example

The FSA described in Figure 4 is simply the set of states to recognize the keyword ALPHA. Only a single transition arc is shown from each state, and there is a single accepting state. The regular expression recognized by Figure 4 is the five characters comprising the identifier, as

ALPHA

which may be graphically described as the FSA named ALPHA:

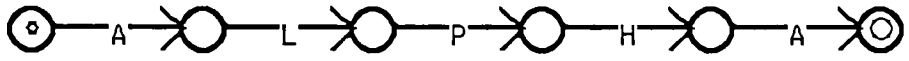

Figure 4: Finite State Automaton for ALPHA

Figure 5 is a more complicated FSA, recognizing valid Pascal language identifiers [Wirth-71]. The regular expression recognized by Figure 5 is simply

[A-Z][A-Z0-9]*

and the graphic representation is the FSA named ID:

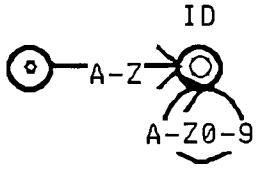

Figure 5: Finite State Automaton for ID

Note that the FSA in Figure 5 accepts a superset of the string accepted by the the FSA in Figure 4, since the keyword ALPHA is a valid identifier in Pascal. Also note that the FSA contains an ambiguous transition, specifically that a string of input symbols ABC is recognized as A, AB, ABC, B, BC and C[1].

Within an FSA, ambiguous transitions result from allowing multiple paths from the initial state to one or more final, accepting state. The proposed utility reports *each* satisfied accepting state as soon as it is entered.

---

1 This presents two additional problems that a scanner has that a FSA does not: (1) a scanner must report *where* in the input stream to stop, and (2) a scanner must also report *which* token it saw.

In general, ambiguous transitions in the accepting state are resolved by arranging for the recognizer to return the *longest* token accepted. This involves reading one additional character to determine the end of the token. This "lookahead" character is then reread on the next call to the scanner. If multiple tokens of the same size are recognized then the first FSA is used to break the tie.

The following regular expression for classic decimal constants

[0-9][0-9]*

is represented by the following FSA, named NUMBER:

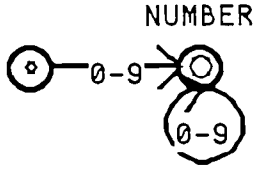

Figure 6: Finite State Automaton for NUMBER

The following regular expression for binary constants

[0+1][0+1]*B

is represented by the BIN FSA:

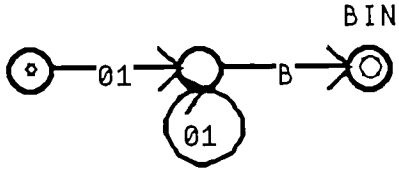

Figure 7: Finite State Automaton for BIN

The following regular expression for classic hexadecimal constants

[0-9][0-9A-F]*H

may be represented by the FSA below named HEX:

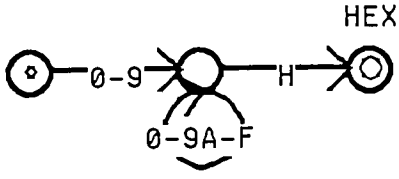

Figure 8: Finite State Automaton for HEX

Note there are several ambiguities described in the regular expressions that prevent the scanner from determining the exact form of a number until the last character is scanned. For instance, the NUMBER FSA cannot determine the end of a number until one addition character is scanned. This one additional, pseudo character is known as the End of File (EOF) character.

Figure 9 is a combination of the previous FSA's for ALPHA, ID, NUMBER, HEX and BIN. This combined FSA recognizes all phrases the constituent FSA's recognize and no more. The individual FSA's are combined by prepending the individual FSA definitions with a common start state and joining it to the existing FSA's with unlabeled (empty) transition arcs.

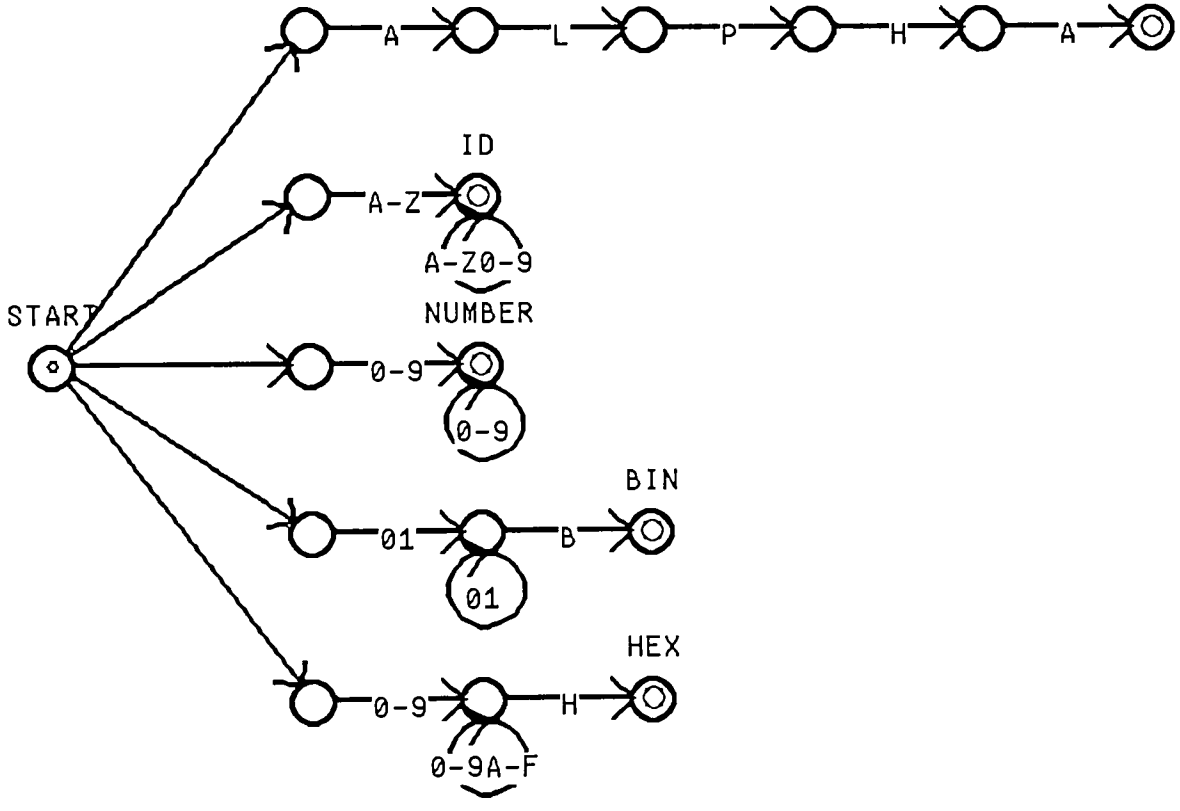

Figure 9: Combined Finite State Automaton

This combined FSA is nondeterministic since there are multiple paths from the starting node. The nondeterminisms are caused by the empty transitions. These problems may be removed in a mechanical manner, however the resulting FSA is more confusing.

The combined FSA scanner is ambiguous since multiple accepting nodes may accept the same input string. For instance the string ALPHA is accepted as both as ALPHA and also as an ID.

1.2.7. Graphic Debugging

Debugging a graphic FSA is easy since the entire design can be directly inspected and altered in an intuitive fashion. The FSA may be run on a string at any time and the results observed.

With the classic approach, the designer translates the state transition diagram into a series of regular expressions which are then submitted to a scanner generator.

The scanner generator re-generates the FSA and then attempts to alter it by:

- removing unreachable states;
- combining equivalent accepting states;
- combining equivalent intermediate states;
- detecting and arbitrating ambiguous expressions states by attempting to always return the *longest* recognized string of input token. Failing that, the *first* recognized regular expression is returned [Aho-72, Lesk-75].

The graphic programming approach allows and encourages additional flexibility at this point. This additional flexibility is due to the graphic representation utilized. The designer is able to:

(1) sketch a proposed FSA directly on the screen;

(2) propose a string of input characters to be used as an input test string to the parser;

(3) simulate the execution of the FSA in a clear and straight-forward manner, and

(4) observe the graphic representation of each "instantiation" of the FSA as the various nondeterministic transitions are detected.

Since the graphic screen allows the direct two-dimensional representation of each FSA instantiation in parallel, the designer has a clear view of each of the parallel portions of his combined FSA.

1.2.8. Debugging Example

Simulation allows the designer to step a FSA through a specified input string and observe the effect on the changing states of parallel FSA's. This is demonstrated below.

Figure 10 shows two interacting windows from an actual parse. The top window, DEMO, contains the current design being considered. The lower window, STRING, displays the input string and the cursor position within the string. As shown, the cursor is positioned just before the final character of the string ALPHA1.

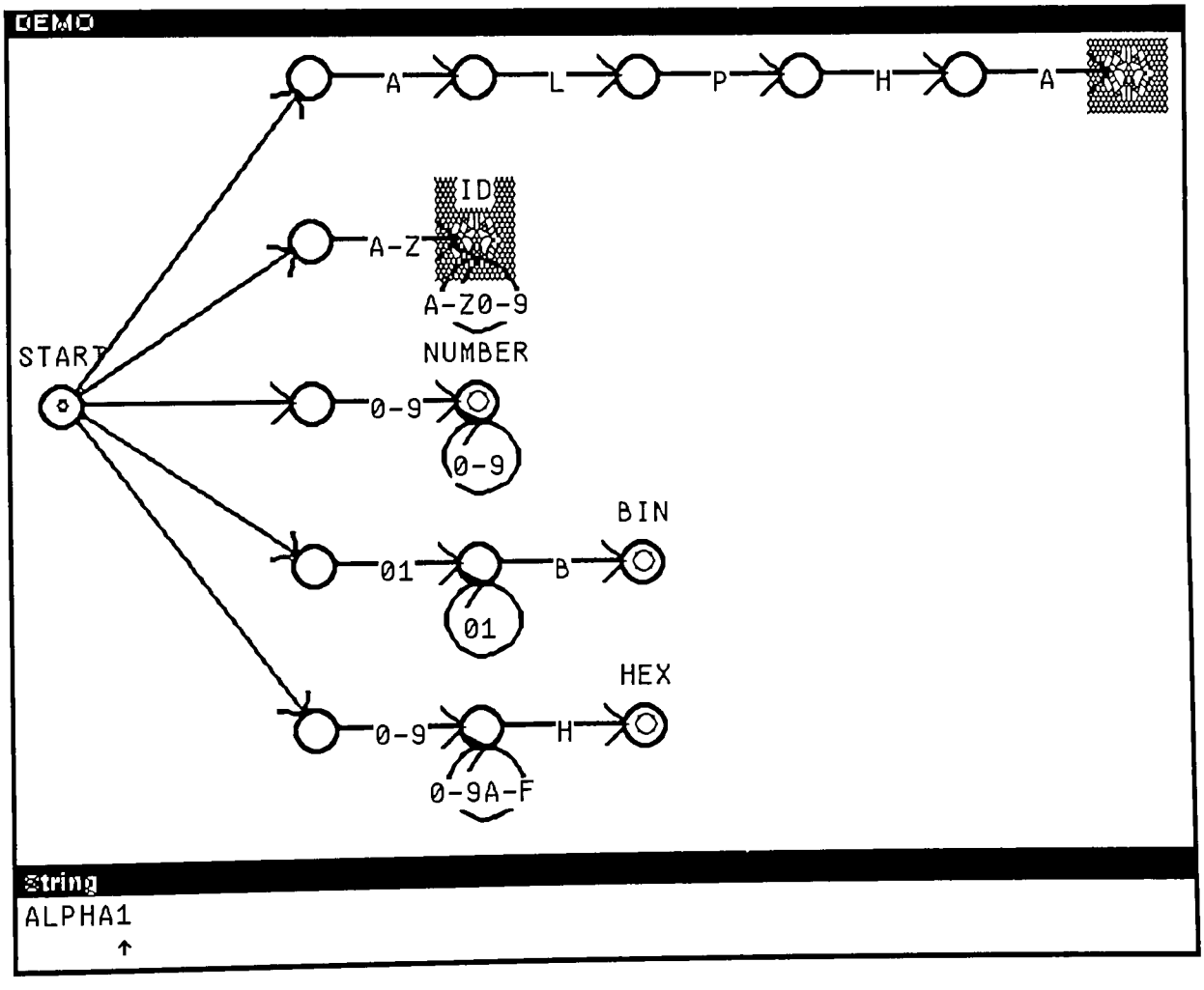

Figure 10: Representation of Parse Simulation

The rectangular gray patterns over two nodes on the FSA diagram are "instantiation" symbols. These indicate that one or more FSA's are occupying, or "instantiated" on, that node. These nodes are said to be "alive". New icons are created when two or more transition arcs exit an occupied node, labeled with the next symbol present in the input string.

In Figure 10, the user has progressed through the input string ALPHA1 to all but the last token in the string, the 1. Two FSA's are active and both are in an accepting state. As indicated in the GOAL window, this demonstrates that the string ALPHA is accepted as both a keyword and as an identifier.

The next step of the FSA parser is to consume the 1 character. The only FSA that will exist after this step is the ID FSA. The ALPHA FSA fails at this step since there is no transition out of its accepting state.

The last and final step of the parse is to advance the input cursor over the final input character, 1, placing the cursor at the end of the string. This step also has the effect of killing the last FSA, the ID, since there is no transition arc out of its accepting state labeled with the end of input. (In fact there can never be such a label placed on an arc, since there can never be a transition arc out of a state label with the end of the input string. All FSA's fail at this point, since there is no way for the designer to specify a transition arcs for the special end of string symbol.)

Since the ID recognizer was the last existing recognizer, it is displayed on the top row of the GOAL window. Therefore the string ALPHA1 is accepted as an identifier. This is because it is the longest match recognized when the last existing FSA perished.

1.3. Previous Work

While attention has been focused on automation of various stages of the compilation process [McKeeman-70, Hohmeyer-71], comparably little effort has been directed toward automation of the front end. As a result, the LEX generator is the dominant utility in the area of automated generators [Lesk-75].

While there is much activity in applying graphic methods to existing development problems [Brown-85, Moriconi-85], there is comparably little work in applying these methods to automated parser program generation.

Jacob's paper, on the visual representation of transition diagrams, [Jacob-85], is interesting since it formalizes a notation for state transition diagram. This notation is a direct graphic representation of a transition network. Labeled edges indicate changes in state based on input tokens and indicate:

(1) the input token required for the transition arc to be taken;

(2) the optional output token to be generated if the transition arc is taken;

(3) the optional name of a separate diagram to be traversed when the transition arc is taken;

(4) the optional name of a semantic level function activated if the transition arc is taken, and

(5) any optional conditions that need to be true for the transition arc to be taken.

Additional details of actions and conditions are deferred to numbered footnotes to reduce screen clutter on the diagram proper.

While much of Jacob's work is applicable to the Visual Design Aide, there are several important omissions. Most important is the limited support for executing nondeterministic programs. Nondeterminism is allowed in the input tokens, but once an output token has been generated, the interpreter is committed and cannot back up and revoke the output token. Jacob explains:

> "The interpreter itself is a deterministic machine that executes a nondeterministic program by arbitrarily trying one of the possible paths. If it reaches a dead end, it backtracks and tries another. The language permits arbitrary nondeterminisms (simulated by backtracking), but the programmer must avoid backtracking over output."
>
> -- [Jacob-85]

The approach utilized in the Visual Design Aide allows an unlimited amount of nondeterminism where such programmer restrictions as backtracking are not needed.

Other work in automatic code production from graphic forms involves Ada [Cherry-86] and the USE.IT system [HOS-85]. The latter is an application of the High Order Soft-

ware (HOS) design principles and assists in the speedy construction of prototype parsers.

Clearly, additional work is needed to in order to automatically produce parsers of production quality. However, the Visual Design Aide demonstrates some of he gains that are possible with the rapid prototyping approach offered by the graphic approach.

2. Operation of the Visual Design Aide

A window is a designated area on the Interlisp-D screen. Every rectangular box on the screen is generally a window. Borders of windows are normally highlighted, to distinguish them from the background and the other windows on the screen.

The Visual Design Aide uses many window features from the Interlisp-D environment. This programming environment is available on a variety of computers, including the Xerox 1108[1] and 1186 workstations. These features, common to the Interlisp-D environment, allow the designer to move and reshape windows any time .

2.1. Windows as Drawing Surfaces

The user of the Visual Design Aide works with Finite State Automata, or FSA's, as described in Section 1.3. A single FSA is generally located in a separate screen window. Each FSA is therefore complete and isolated from all other FSAs.

Since each design window is separated from any others on the display screen, the designer can arbitrarily alter a feature of one FSA, without worrying about the state of the other FSA's on his screen.

The designer causes a new FSA design window to be created with the function call

```
(CREATE.MAIN.WINDOW name)
```

where "name" is the name to be associated with the new window. At this point, a prompting "ghost" box appears on the screen and the designer is prompted to "drag out" a region that is to be the new window. The designer moves the mouse around while depressing the left mouse button, noting that it sweeps out a rectangle. When the rectangle reflects the location and size of the desired new window, the mouse button is released.

---

1 In the Xerox product nomenclature, the 1108 is a LISP configuration of the standard 8010 word processing workstation. Additional memory and microcode are installed to support Interlisp-D operation. Similarly the 1186 is the LISP version of the 6085 workstation.

The Xerox 1108 is also know as the "Dandelion", while the 1186 is called a "Dove". These names apparently refer to the internal project names under which the machines were originally developed.

Immediately, a main design window appears, with a starting node in the middle. This starting node is supplied to merely start the design; it may be easily deleted.

Such a new design window is show in Figure 11, immediately creation. The name of the window is "DEMO". Notice that the design window has a permanently command window attached at the upper right corner. Also a starting node (START), has been automatically provided the designer in the center of the window.

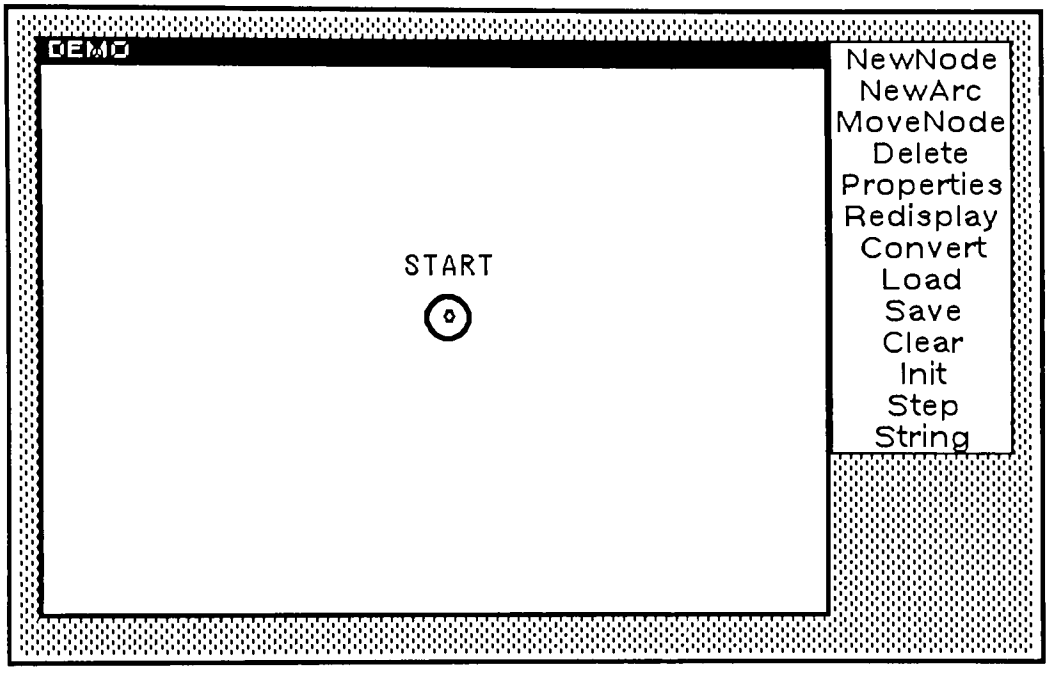

Figure 11: Initial Contents of a Design Window

2.2. Manipulating Objects in the Design Window

Objects displayed within the main design window are manipulated by first selecting the object or objects to be manipulated and then selecting the operation from the attached menu.

Objects in the main design window are selected by moving the mouse until the on-screen cursor is near the desired object and pressing the left button on the mouse. This is selecting an object. Visual feedback is given to the designer by highlighting the object. In general, only a single object may be selected at one time, though two nodes can be selected at the same time. Selecting a new object causes the previously selected objects to be deselected.

All currently selected objects can be deselected by pressing the left mouse button when the cursor is positioned away from any objects in the main design window.

After one or more objects in the main window are selected, the designer selects one of the items on the attached menu. Pressing the left mouse button selects the action specified by the menu item.

If the designer wishes to delete an object he would select the object, note that the object is highlighted and would then select "Delete" on the menu. The object is removed from the design.

One mode of operation is different. To create a node, the designer *first* selects the "NewNode" menu item, causing the cursor to change from the normal arrow into a crosshair. The designer positions the crosshair to the desired location, pushing the left mouse button to create the new node. After the node is created the cursor changes back to an arrow.

If the designer changes his mind and decides *not* to create the new node, after selecting the "NewNode" mode, he could cancel this mode by returning to the attached menu and selecting some *other* menu option. "Redisplay" is an excellent choice, as it causes the main design view to be cleared and redrawn.

2.2.1. The Main Design Window

The main design window always contains the designer's current view of the FSA being constructed. This main design window is continually updated to reflect changes in both the design and the stages of the parse simulation process.

This design window is the primary point of interaction with the FSA designer. The FSA designs are drawn on this window with the use of the cursor. Nodes are positioned, arcs are added and properties of objects are changed as the design is refined.

The main design window is a view of the design. If the design grows larger than the window, scroll bars automatically become active at the bottom and/or left side of the window. The extent of the window may be "stretched" by positioning objects at the extremes of the window. The Design Aide automatically increases the extent of the window in that direction.

2.2.2. The Attached Menu

A small menu window is attached to the upper right corner of the main design window. This smaller menu window contains the menu items that can be selected by the designer during the course of the design session. Entries in the menu are:

Menu Item	Function performed
`NewNode`	add a node
`NewArc`	add a new arc
`MoveNode`	move selected node to a new location
`Delete`	remove an arc or node
`Properties`	inspect and/or alter element properties
`Redisplay`	repaint main design window
`Convert`	convert FSA design to a regular expression
`Load`	load FSA design from a file
`Save`	save FSA design to a file
`Properties`	inspect and/or edit element properties
`Clear`	clear display of parse simulator tokens
`Init`	initialize the parse simulator
`Step`	advance the parse simulator
`String`	change the parse string

2.2.3. The String Window

During simulation mode, an additional window appears attached to the bottom of the main design window. This window shows both the string being parsed along with a moving arrow indicator indicating current location.

The parser simulation mode is initiated, via the "Init" menu command and disappears when the main window is cleared of the simulation icons with the "Clear" menu command.

2.3. Nodes And Arcs

The FSA parse network is drawn by the designer in the main design window, using only two types of objects: nodes and arcs. Nodes are shown as circle icons, while arcs are displayed as directed arrows from one node to another.

2.3.1. Nodes as Places

As discussed in 1.2.4, a node appears in the main design window as a general icon. The general form of the icon is of a circle--which indicates that this is a "node" or "place" for the parser to rest between input symbols. The node is optionally named with a string above the outer circle. "Label" is the name of this node. The small inner circle indicates this is a starting state. The intermediate circle indicates this node is a valid ending point for the parser.

2.3.2. Arcs as Transitions

An arc is a transition from one node to another. One node is the source node; the other is the destination. Normally arcs are created from one node to a different node. However one node may be both the source and the destination of an arc. Since arcs have a source and destination, they are said to be *directed* arcs. The destination is indicated by an arrow head. Arcs may be labeled with the characters that allow the parser to follow this arc.

These attributes are demonstrated in the following segment of an FSA:

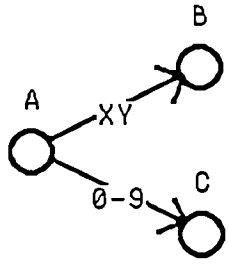

Figure 12: Segment of a FSA

If the parser enters node A, it will only pass to node B when the character X or Y appears in the input stream. Any number character, indicated by the 0-9, will cause the parser to enter node C following node A.

2.4. Manipulation of Objects

In general, objects are manipulated by first selecting them with the cursor. Then the desired operation is selected on the attached menu window. An exception to this generality is the initial creation of a node.

2.4.1. Creating Nodes

Nodes are created by setting the cursor to the crosshair mode by selecting the "NewNode" menu item. The normal arrow cursor is replaced with a crosshair cursor, which the user positions to the directed location of the new node. The left mouse button is pressed, creating a new node at the selected location. The cursor changes back to the normal arrow mode.

If the designer wishes to change his mind about creating a new node after the crosshair mode is entered, selecting any other menu item will cancel the crosshair cursor mode.

2.4.2. Moving Nodes

A selected node and its associated arcs are moved with the "MoveNode" menu entry. First the node is selected and then the "MoveNode" button is moused. A crosshair cursor appears. After a new location is selected by the user, the node and all connecting arcs are erased from the window and new ones are created at the new position.

An additional feature is that a "snap grid" is applied to the newly selected position of the node. This allows a designer to line up nodes in a design window by merely moving nodes. This "snap grid" feature is only active on "MoveNode" menu entry.

2.4.3. Changing Node Properties

Nodes are displayed with a circle, indicating that they are a "place" where the parser can live between input symbols. In addition nodes have the following attributes that can be altered on a property sheet:

Attribute	Meaning
NODENAME	Name of the node
IFSTART	set if FSA starting node
IFEND	set if FSA ending node
IFALIVE	set if node currently alive, that is, one or more FSA instantiations are current within this node

These attributes are altered by selecting the node in question and selecting the "Properties" menu command. A small property window appears, indicating the present

state of each of these attributes for the node selected. The designer may change these attribute settings with the cursor and keyboard. The new attribute setting is applied to the node when the property window is closed. The right button Interlisp-D menu is used to close the property window.

2.4.4. Deleting Nodes

Nodes may be deleted by selecting them and then selecting the "Delete" menu entry. The node is removed from the main design window. If the node has any arcs drawn from or to it, these arcs are also removed.

2.4.5. Creating Arcs

Arcs are created by selecting first the source node and then the destination node. Both nodes will be highlighted[1]. The "New Arc" menu item is selected. A new arc is created from the source to the destination node.

Most arcs are straight, directed lines between different nodes. However a single node may be connected to itself with a looping arc. Such a looping arc is created by selecting the same node twice, thereby indicating that is both the source and destination. The connecting arc is then drawn as a closed circle, hanging down from the node.

2.4.6. Changing Arc Properties

The arc transition character string is written on top of the image of the arc. This property can be manipulated:

Attribute	Meaning
ARCSTRING	Arc transition string

As in the case of changing node properties, the cursor is used to select the arc to be modified and the "Properties" menu item is selected. A small property window is shown on the screen, containing the series of characters that are to be associated with the arc. This property window must be closed before any changes become effective.

---

1 This is the only time that multiple items can be selected and highlighted at the same time.

The syntax of this string is much the same as those in regular expressions, as discussed in Section 1.2.2. Since only a single character is being described by this string, a pair of square brackets ("[" and "]") logically surround any string specified. For example, the string "A-Z" indicates that all uppercase alphabetic may cause the arc to transition.

Specifying an empty string for an arc transition string indicates that this arc is an empty transition.

2.4.7. Deleting Arcs

Arcs are deleted by first selecting them and then selecting the "Delete" menu entry. The selected arc is removed from the window.

2.5. Convert an FSA to a Regular Expression

Selecting the "Convert" menu item causes the FSA currently to be converted to an equivalent regular expression. The resulting regular expression is displayed in the Prompt Window, commonly found at the top of the Interlisp screen. Then the user is prompted to enter a file name to store the regular expression, if desired. Answering this prompt with a carriage return prevents the file from being written.

This feature is active at all times and can provide feedback to the actions of the scanner being designed.

2.6. Loading and Saving FSA Designs

FSA designs are written to and read from names files by selecting, respectively, the "Save" or "Load" menu items. The external storage format is specific to the Visual Design Aide, though it utilizes LISP S-expressions. Loading a design completely replaces any design in the window before the load, so the designer should only use the "Load" item if the current design may be overwritten.

2.7. Running the Parser

When a designer wishes to observe how his FSA design reacts to parsing a string, the simulator portion of the Visual Design Aide is activated. This simulator visually demonstrates the step-by-step process of accomplishing the parse. This is done by detailing each parse step taken by the FSA design.

2.5.1. Initializing the Parser

The parser simulator is brought into operation by selecting the "Init" menu item on the attached menu. This causes three actions:

1. The parse string is requested from the designer if one was not previously specified.

2. The initial state of the parse is entered. Active or alive nodes are indicated by special highlighting pattern being displayed over the node.[1]

3. A parse process window is opened and attached to the bottom of the main design window. This parse process progress window indicates the current string being parsed with an arrow pointing to the next character to be presented to the FSA.

2.5.2. Setting the String to be Parsed

If the designer wishes to display or alter the string to be processed, he may select the "String" menu item from the attached menu. This causes a small modal window to appear, with the request to supply the the new string to be parsed by the FSA.

This new parse string will remain in effect until another request is made to change it.

2.5.3. Stepping the Parser

After the initial setup of the parse simulator is complete, the user causes each state of the parser to be entered, one step at a time. The "Step" menu item on the right-hand attached menu window causes this stepping action to occur.

The result of each parse step is fed back to the display screen by updating the list of nodes that are alive by manipulating the gray overprint patterns. Also the parse progress window at the bottom of the main design window is updated to indicate the parse simulator's progress.

When a string is successfully parsed, the screen is flashed at the operator by inverting all the screen pixels several times and sounding the bell character. A successful

---

1 A light gray overprint is the current instantiation pattern.

parse is detected when the string has been completely consumed and there are one or more ending nodes alive.

2.5.4. Clearing the Display Window

If the designer decides that he would like to further manipulate the FSA design in the main window and *not* complete the current parse simulation, he would select the "Clear" menu item. This action removes the parse progress window from the bottom of the screen and removes the gray overprint patterns from the nodes. This change is only cosmetic in nature, in that it allows a clear view of the FSA design; no changes are made the the underlying FSA structure.

Clearing the display window differs from "Redisplay" in that the artifacts of the parse simulator are removed from the display. The "Redisplay" button merely causes the main design window to be initialized to its background white color, and the window objects to be redrawn. In normal operation, the "Redisplay" button should not be needed.

3. Structure of the Visual Design Aide

The Visual Design Aide is designed around and utilizes many of the features, particularly windows, of the Interlisp-D programming environment. This approach encourages implementation of sophisticated programs with a minimum of explicit interconnection. Further, such designs may grow and evolve easily as requirements become more clear.

3.1. Object Oriented Concept

Central to the design is the object-oriented programming concept . In this paradigm, data and procedures are combined into "objects". These objects are constructed of Interlisp-D data structures that intermix traditional program data, as well as the programs themselves.

Messages, in the form of events, are sent to the objects to cause them to perform actions appropriate to the event. Typical events in the Visual Design Aide are mouse button events in various areas of the display screen. While there is, of course, only a single type of "left mouse button" event, the Design Aide software synthesizes a far finer level of event, based on when and where the mouse button was pressed.

The central object in the Visual Design Aide is the main design window. This window is created by the parser designer. All data and program functions are linked to this window. Thus the main design window, with its property lists, is the primary abstract "object" in the design.

From the viewpoint of the Interlisp-D executive, the only direct user command action is `CREATE.MAIN.WINDOW` to create the main design window. All subsequent actions are invoked by sending messages to the window.

3.2. Windows as Objects

Interlisp-D is more than a programming environment; it is also a window management environment. Each window in the system, visible or invisible, is maintained on a completed internal structure. The `WINDOWPROP` function allows one to augment the system list of parameters with additional LISP data structures.

In addition, standard Interlisp-D attributes are defined for each design window. These include:

Property	Meaning and Value
RESHAPEFN	function called when the main design window is reshaped; currently RESHAPE.MAIN.WINDOW
REPAINTFN	function called when the main design window is repainted; currently REPAINT.MAIN.WINDOW
BUTTONEVENTFN	function to be called when a mouse button is pressed within the bounds of the main design window; currently BUTTON.MAIN.WINDOW

3.3. Interlisp-D Window Properties

The properties defined in the main design window for the Visual Design Aide include:

Property	Meaning and Value
MODE	current cursor mode, either SELECT or NEWNODE
OBJECTLIST	current list of object in the window, both NODEs and ARCs
SELECTLIST	current list of selected and highlighted objects in the window
STRING	current parse string to use as a test string

The particulars of each of these items are described in sections below.

3.3.1. MODE Property

This field reflects the current mode of the cursor, either SELECT or NEWNODE.

The SELECT mode, which is indicated by displaying the "normal" northwestward-pointing arrow cursor, indicating that the window is in "normal" cursor--that is, the cursor is a selecting cursor.

Pressing the left mouse button in the vicinity of a graphic icon has the effect of selecting the icon. The normal effect of selecting a graphic object within the main design window, is to highlight the selected object and add a descriptor for the highlighted object on the selected list of the window.

The NEWNODE mode of the cursor is indicated by a "crosshair" cursor, and indicates that the main design window is adding a new node. When the left mouse button is pressed within the window, a new node is placed at that location. After the new node is created, the cursor immediately returns to its normal "select" mode of operation.

3.3.2. OBJECTLIST Property

The OBJECTLIST window property of the main design window is a list containing one record for each object currently defined with the window. Currently, only ARCs and NODEs are defined.

Both ARCs and NODEs are defined by separate Interlisp-D RECORD structures. These structures are described below.

3.3.3. SELECTLIST Property

The SELECTLIST property is a list of objects that are currently selected. These objects are a subset of those on the OBJECTLIST. Each object in the SELECTLIST property list is highlighted on the screen to provide visual feedback to the designer.[1] No more than two objects are allowed on this list. If two objects are on this list, they are constrained to be nodes.

Objects are added to the SELECTLIST by selecting the object within its bound region. Mouse down events from the left mouse button are passed to the DO.SELECT function, which manages the SELECTLIST property.

In general, only a single object at a time is ever placed on the SELECTLIST. This object can either be an arc or a node. However two *nodes* can be selected in the case that the designer wishes to add a new arc to the FSA.

Once objects are referenced in a command, they are removed from the SELECTLIST and their bound boxes are unhighlighted. Objects may also be removed from the SELECTLIST if the left mouse button is pressed outside of the bounding box of *any*

---

1 The current scheme of highlighting objects selected on the screen within the main design window is to invert rectangular boxes at the four corners of the object's bounding box. Thus, a highlighted object will be outlined by four small boxes, generally black, at the extremes of the object's area of influence.

screen object. This causes all selected objects to be deselected and SELECTLIST to become NIL.

In the event that one arc is selected, selecting any other object causes the first arc to be removed from SELECTLIST.

3.3.4. STRING Property

The simple STRING window property contains the current parse string to be passed through the FSA during simulation. If this property is NIL when the simulation is started, a string is requested from the designer. This string remains in effect until the user indicates that he wishes another string. This action is accomplished by selecting the "String" item on the attached menu window.

3.4. Contents of the Object List

Two kinds of object descriptor records are threaded off the OBJECTLIST property list. These two kinds of objects, ARCs and NODEs, share a common portion of their record structure. This common portion contains the exact same fields, much as in the case of Pascal's variant record structures.

As show in Figure 13, common portions of the record are placed at the start of the record, with the object-specific fields placed at the end of the record.

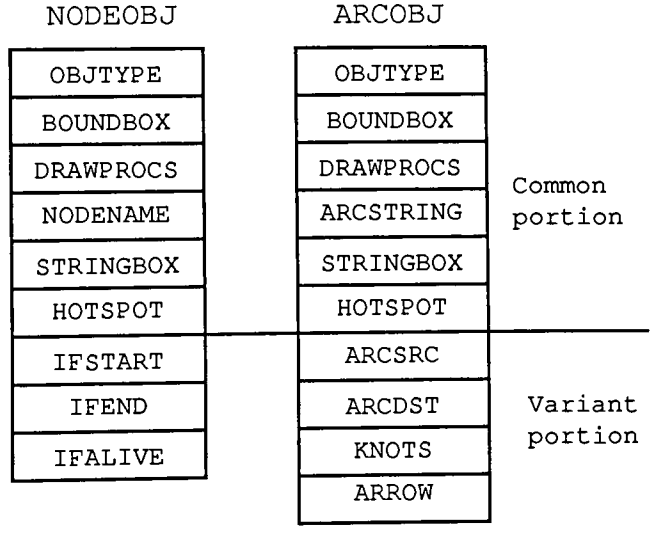

Figure 13: Variant Structure of Object Record

Use of this common record structure allows many of the object handling functions within the system to handle either type of object without regard to what the particular object actually is. For instance, the DO.SELECT function that builds and main the previously-mentioned SELECTLIST can rapidly build a list of objects that are "hit" by a particular mouse event.

```
(SETQ SELECT/NOW (for OBJECT in OBJECTLIST when (INSIDEP
   (fetch BOUNDBOX of OBJECT) X Y) collect OBJECT))
```

In short, this code checks all objects on the OBJECTLIST property for intersecting the X and Y position of the mouse event, binding the list of these intersecting objects to the atom SELECT/NOW. FETCH is a function returning contents of a named field within a record. INSIDEP is a predicate function returning T if an X and Y position intersect with a bounding rectangle.

The fields common to each type of object record are:

Field	Contents
OBJTYPE	type of object, either NODE or ARC
BOUNDBOX	a rectangular region[1], in window relative coordinates, enclosing all aspects of the object. Used to determine mouse "hits" and when to redraw the object.
DRAWPROCS	a list of functions to draw and select this type of object, see record below.
NODENAME	identifying name of the object, if any. For arcs, this is known as ARCSTRING. This field is the name for a node or the list of characters accepted by an arc.
STRINGBOX	a rectangular region enclosing the string (NODENAME or ARCSTRING) on the screen. Used to position labels. BOUNDBOX contains STRINGBOX.

1 Defined by Interlisp-D as a REGION record, with the fields LEFT, BOTTOM, WIDTH and HEIGHT.

HOTSPOT	this is a POSITION[1] relative to the window that is the *logical* center of the object. For nodes the HOTSPOT is the center of the circle, for arcs it is generally the center of the arc, it is relocated in certain cases. The HOTSPOT field is used by DO.SELECT to decide which object to select when more than one object's BOUNDBOX intersects the current mouse position. The object closest to the mouse is chosen.

3.4.1. NODEOBJ Object

A NODEOBJ record represents one node of the FSA. The fields specific to the NODEOBJ, other than the common fields described above, are:

Field	Contents
IFSTART	boolean, true if node is a starting location for the parser
IFEND	boolean, true if node is a ending location for the parser
IFALIVE	boolean, true if node is currently "alive" in the parser simulation. That is, at least one instantiation of the FSA is present within this node. This field is manipulated by the parser simulator, though it may be altered by the designer with the node property sheet.

3.4.1.1. ARCOBJ Object

A ARCOBJ record represents an arc of the FSA. The fields specific to the ARCOBJ, other than the common fields described above, are:

Field	Contents
ARCSRC	NODEOBJ record where this arc originates

---

1 Defined by the Interlisp-D POSITION record, with the fields XCOORD and YCOORD. This record is unique in the system, as it is actually implemented as a CONS'ed dotted pair.

ARCDST	NODEOBJ record where this arc terminates
KNOTS	a list of POSITION records describing how this arc is drawn. This can be as simple as a pair of POSITIONs, for a straight line, to three POSITIONs for a bent line to an 18-element list of POSITIONs approximating a circle. This last knot type is used only when the ARCSRC and ARCDST fields contain the same NODEOBJ
ARROW	a list of three POSITION records describing how the arrow head is drawn on the end of the KNOTS list. Given as the ending position of the arc, followed by each of the two arrowhead tips.[1]

3.4.1.2. DRAWPROCS

As mentioned above, the DRAWPROCS record describes the DRAWPROCS field in both the ARCOBJ and NODEOBJ records. This record isolates the window and object handling functions from the specifics of a particular object type. An object can be drawn, highlighted or unhighlighted without knowing anything about the particular object type.

The field of this simple record are:

Field	Contents
DRAWPROC	function to draw this object, either PAINT.ARC or PAINT.NODE
SELECTPROC	function to select and deselect this object, the function SELECT.OBJECT is used to both select and deselect nodes

[1] The KNOTS and ARROW fields are stored with ARCOBJ record to allows the arc to be rapidly redrawn on the screen. These points are only recalculated when the arc is changed or the ARCOBJ record altered, thus removing the need to perform the trigonometry functions on each repaint of the arc. An earlier implementation used this approach and the overhead of using the Interlisp-D circle drawing functions was noticeable.

and arcs, since both are described by a rectangular bounding box.

3.5. Source Program Layout

Source programs for the Visual Display Aide are arranged into three files. These files are listed in their entirety in Appendix C. Written descriptions of the functions are given in the following sections.

These files are:

File	Description
DRAW	all code concerned with drawing objects on the screen
FSA	all code concerned with running the FSA simulator
WINDOW	all code concerned with drawing windows and handling mouse events

3.5.1. Drawing Functions

This section describes the object painting and drawing functions.

Routine	Description
BEND.ARC	bends an existing arc into three line segments, in the case that two arcs overlap
BIAS.POS	biases a dotted pair by another dotted pair
CALC.ARC.ARROW	calculates an arrowhead for a line segment
CALC.ARC.BOUND	calculates bounding box for an arc, updating the window extent
CALC.ARC.DYNAMIC	calculates dynamic properties of an arc, bending and unbending the arc as necessary
CALC.ARC.HOTSPOT	calculates the hotspot of an arc

CALC.ARC.KNOTS	calculate the knot list to display a loop
CALC.ARC.STRINGBOX	calculate the bounding box for the descriptive text of an arc
CALC.NODE.BOUND	calculate the bounding box of a node, updating the window extent
CENTROID	returns the centroid of a list of positions
CREATE.ARC	create a new arc in response to a menu command
CREATE.NODE	create a new node in response to a menu command
DISTANCE	return the "manhattan" distance between two positions
DO.DELETE	deletes a selected object, in response to a menu command
DO.DELETE.ARC	deletes an arc
DO.DELETE.NODE	deleted a node
DO.MENU.ITEM	handles mouse event within menu window
DO.MOVE.NODE	prompts for new node locations and moves selected nodes and associated arcs, redrawing all affected window areas
DO.NEWARC	adds new arc between two selected nodes
DO.PROPERTY	opens property sheet for a selected object
DO.PROPERTY.CLOSE	handles property sheet close, using free menu package for an object. Actually updates the properties for the objects.
DO.PROPERTY.SETUP	setups a property sheet, using free menu package, along with current properties of an object.

DO.SELECT	processes mouse down events in the design window to determine which object(s) should be selected or deselected, updates SELECTLIST accordingly.
DO.SNAP.GRID	"snaps" a window-relative XY positions to the nearest multiple of GRID
EXISTING.ARC	predicate function for determine if an arc exists between two nodes
INIT.DRAW	setups initial draw objects
INIT.MENU	setup up the menu window, attaching it to the main window
INIT.MENU.TEXT	creates new window text for menu window
LINE.OFFSET	returns a position on a line, given two end points, and a distance from one of the end points
PAINT.ARC	paints an arc from one node to another, using pre-calculated knots, arrowhead and stringbounding box parameters.
PAINT.NODE	paint a node, using pre-calculated positions
PAINT.SELECT	paint select markers around all selected objects
SELECT.OBJECT	draws or undraws select markers around an object
UNBEND.ARC	unbends an existing three-segment arc back into a single segment. Used in the case that a previous arc intersected with this arc.

3.5.2. Finite State Automaton Functions

This section describes the FSA simulation functions.

Routine	Description
ALIVELIST	given a list of nodes, returns list of alive nodes

ALIVEP	predicate to determine if node is currently alive
ARCP	predicate to determine if object is an arc
BIRTH	function to make a given node alive, handling all ε arc transitions
DEADP	predicate to determine if node is dead (same as not alive)
DO.CONVERT	converts a FSA graph into a regular expression, optionally writing the resulting expression to a file
DO.FORMAT	formats a pattern into a regular expression
DO.GETARCS	returns list of arcs leaving a node
DO.GETEXIT	returns a list of paths from a root to an ending node, avoiding specified nodes
DO.GETPATH	returns a list of paths from a root to a specified node, avoiding specified nodes
DO.GETSTR	returns the pattern string associated with an arc
DO.NODUP	removes all duplicates from a list
DO.PREFIX	return the maximum length common prefix from a list of strings cons'ed with the remainder of the strings
DO.SIMULATE.CLEAR	remove all live icons from the display, in response to a menu command
DO.SIMULATE.INIT	initialize the simulator, in response to a menu command
DO.SIMULATE.STEP	move the simulator one step forward
DO.SIMULATE.STRING.DRAW	

	draws string in a small window attached to the main design window, complete with an arrow, indicating the current position in the parse string
DO.SIMULATE.STRING	prompt user for the test parse string to use, in response either to a menu command or to starting the simulator without a string
ENDP	predicate to determine if node is an ending node
GETCHAR	return the next character from the parse string
KILLALL	kills all nodes on a given list
MIN-MATCH	returns length of the maximum length prefix common to a list of strings
NODEP	predicate to determine if object is a node
NUMBER-MATCH	returns the number of common leading characters prefixing two strings, taking into account balancing square bracket characters
PARSE.STEP	controlling function for parser, in response to a menu command
REGULAR.MATCH	determine if character matches a given regular expression
SETALIVE	sets all nodes on a list to be alive
SETDEAD	sets all nodes on a list to be dead
STARTP	predicate to determine if node is a starting node

3.5.3. Window Functions

This section describes the window handing functions.

Routine	Description
BUTTON.LEFT WINDOW	handles left button closures on the mouse
BUTTON.MAIN WINDOW	handles all button closures on the mouse
CREATE.MAIN.WINDOW	create main design window and initializes all window structures and lists
DO.FILE.OPS	performs load and save operations on an FSA
DOPROPS	handles property alteration of all selected objects in response to menu command
PROMPT/YES/NO	prompts user for simple yes or no response, given a prompting string
REPAINT.MAIN.WINDOW	repaints selected portion or all of main design window, in response to changes in contents of window, scrolling or uncovering of window portions
RESHAPE.MAIN.WINDOW	reshapes main design window in the event that the window is reframed. This is caused by the user selecting this item from the system menu.

3.6. Design Tradeoffs

During the development of the Visual Design Aide many design decisions were made and sometimes remade. Several of these decisions are discussed here as they may be interesting to the reader.

3.6.1. The Arrowhead Problem

The process of generating arrowheads for arcs was thought to be a simple problem. However, it proved to be difficult and time consuming.

Initially, trigonometric functions were used to determine the coordinates of two points offset from the end of the line by 45°. These functions proved unworkable for two reasons. First they were too slow to generate while the arcs were being drawn. Sec-

ondly, these functions sometimes calculated singular points when line segments were vertical.

The solution was to pre-calculate these arrowhead coordinates when the arc is initially created, rather than each time it is drawn. The coordinates are stored in an additional field in the ARCOBJ record. The second problem was solved by using matrix multiplication of ratios rather than trigonometric functions.

3.6.2. The Arc to Self Problem

In an FSA it is perfectly legal to have a node transition to itself. In fact, designs without this feature are rare. This forces the line drawing functions to draw some sort of looping arc. Initial implementations drew such a transition as a section of a circle. However this proved to be far too slow when rapid repainting of the screen was needed.

The answer, as before, was to pre-calculate the arc as a series of straight line segments (currently 18) when the arc was defined. Then when it is time to paint the arc, it is merely passed to the Interlisp-D runtime as a series of straight line segments, which paint rapidly.

3.6.3. The Overlapping Arc Problem

A similar problem arose when it was determined that an adjoining pair of nodes could easily have mutual transitions. As shown on the left of Figure 14, overlapping arcs lead to a mess, since the lines cannot be distinguished and the descriptive character strings appear at the same position.

A second problem with such a design is that the user cannot select a particular one of the line pairs, as they are drawn precisely on top of each other.

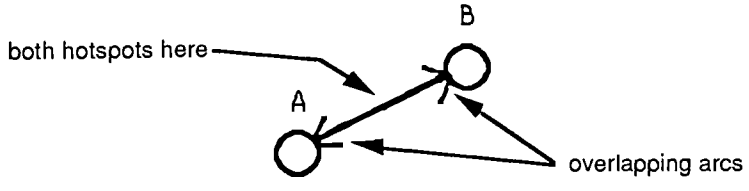

Figure 14: Overlapping Arcs

The solution, shown in Figure 15, makes use of the segmentation codes developed to solve the arc to self problem. The line segments are simply bent into two segments, following the corners of the bounding box. Also the "hotspots" are redefined to the be

the inflection points of the line segments, making the designer able to select either arc without difficulty. Remember that "hotspots" are utilized by the selection logic to determine *which* object is being pointed to by the mouse.

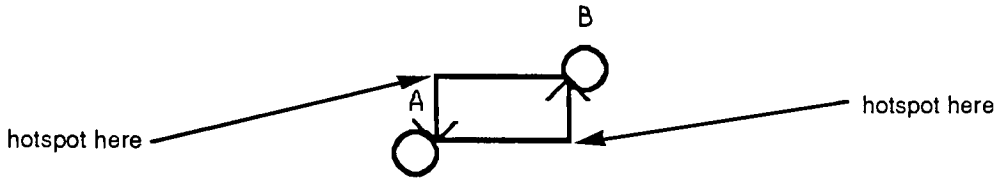

Figure 15: Overlapping Arcs Bent into Segments

This solution does not work correctly for joining two nodes close to the same horizontal or vertical position; sufficient vertical and horizontal displacement is necessary to correctly bend the arcs. The designer should simply move one of the joined nodes until the drawing is pleasing.

4. Sample Graphic Design Session

In this section a complete session with the Visual Design Aide is given. The output shown in this section is of actual Interlisp-D screens that have been captured during an illustrative session.

Initially the user logs into the Interlisp-D system and requests creation of a design window. This is shown in Figure 16. In response to the call on line 29 to create a window

```
(CREATE.MAIN.WINDOW DEMO)
```

the software prompts the user to describe a window location and size to use as the main design window.

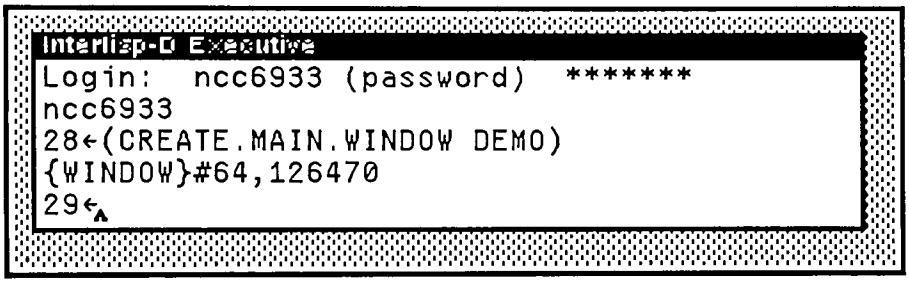

Figure 16: Creating a Main Design Window

The new design window is show in Figure 17, immediately after it is created. Notice that the name of the window is the "DEMO", the same name used on line 11 to create the window. Note also that the window has a permanently attached command window associated with it. This menu window is shown at the upper right corner.

As an start on the design, the program has automatically provided the designer with a starting node, aptly named "START" in the center of the window.

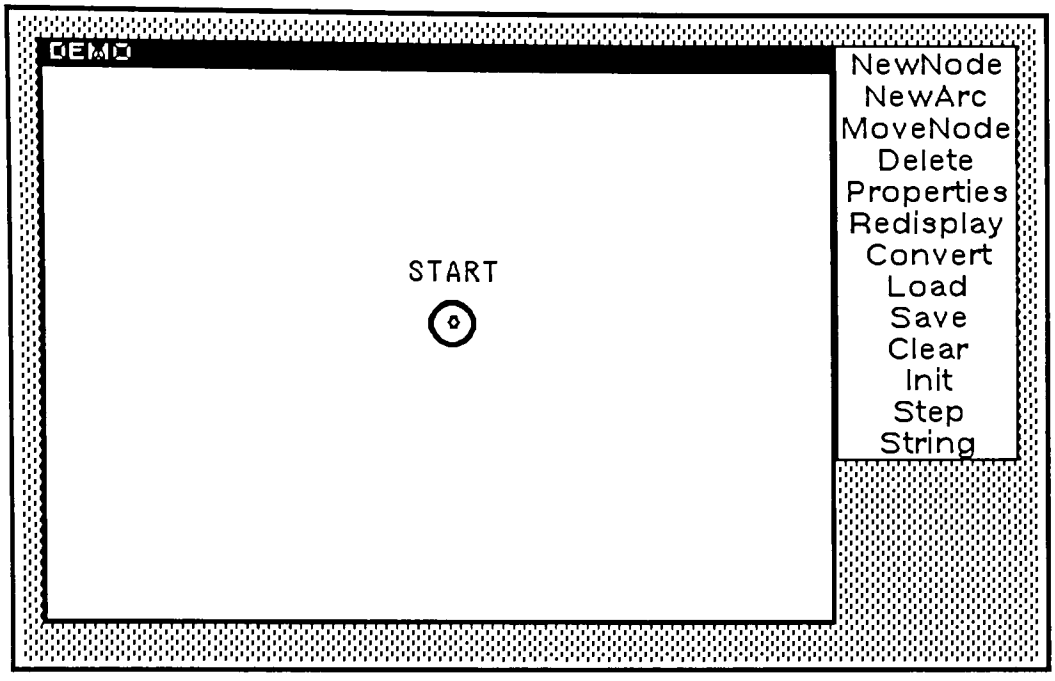

Figure 17: Initial Contents of a Design Window

Next the designer decides to add a second node to the window. This is done by selecting the "NewNode" menu item in the attached menu window. The cursor changes to a cross hair symbol until the mouse cursor button is again pushed. After placing the second node, the window appears as in Figure 18.

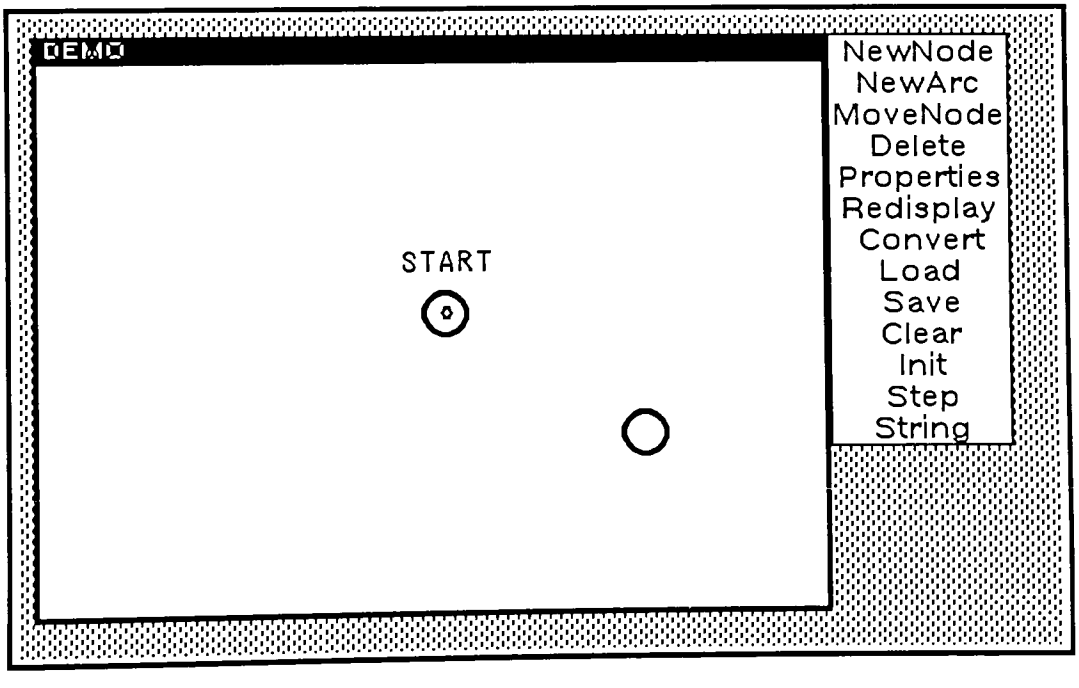

Figure 18: Creating a Second Node

Now the designer wishes to make the new node an ending node. This is done by altering the properties of the new, as yet unnamed node. The node is selected by positioning the cursor near the node and pressing the left mouse button. This causes the appearance of the display to change to that shown in Figure 19. Note the the inverted video bounding the selected node.

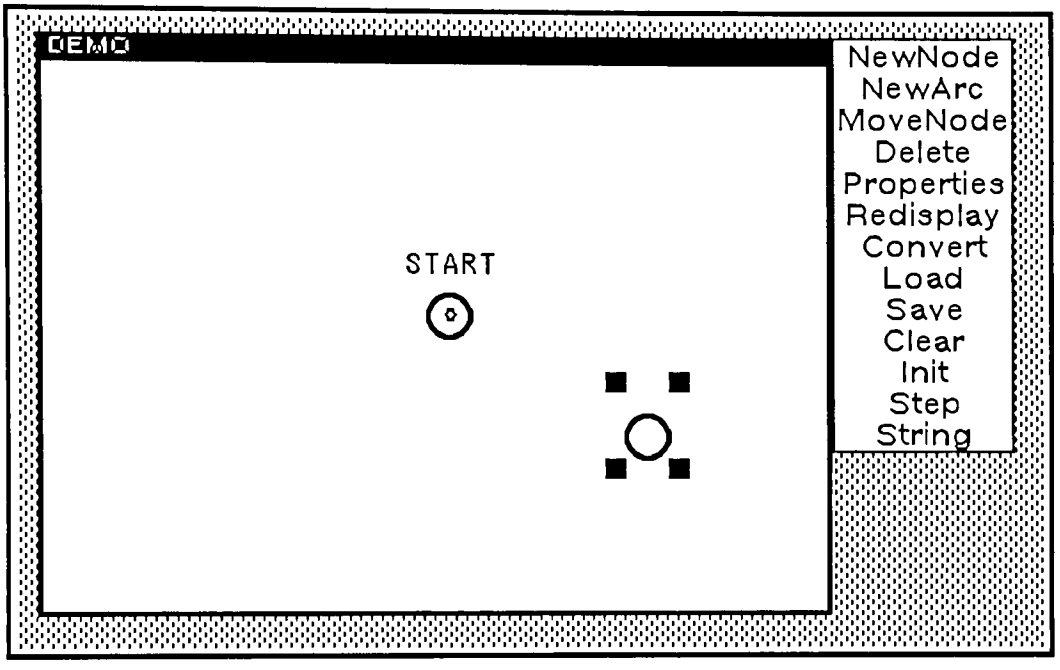

Figure 19: Selecting a Node

Now the designer selects the "Properties" menu item. Since a node was selected, the node property sheet is brought into view. This sheet is altered to make it an ending node, as shown in Figure 20.

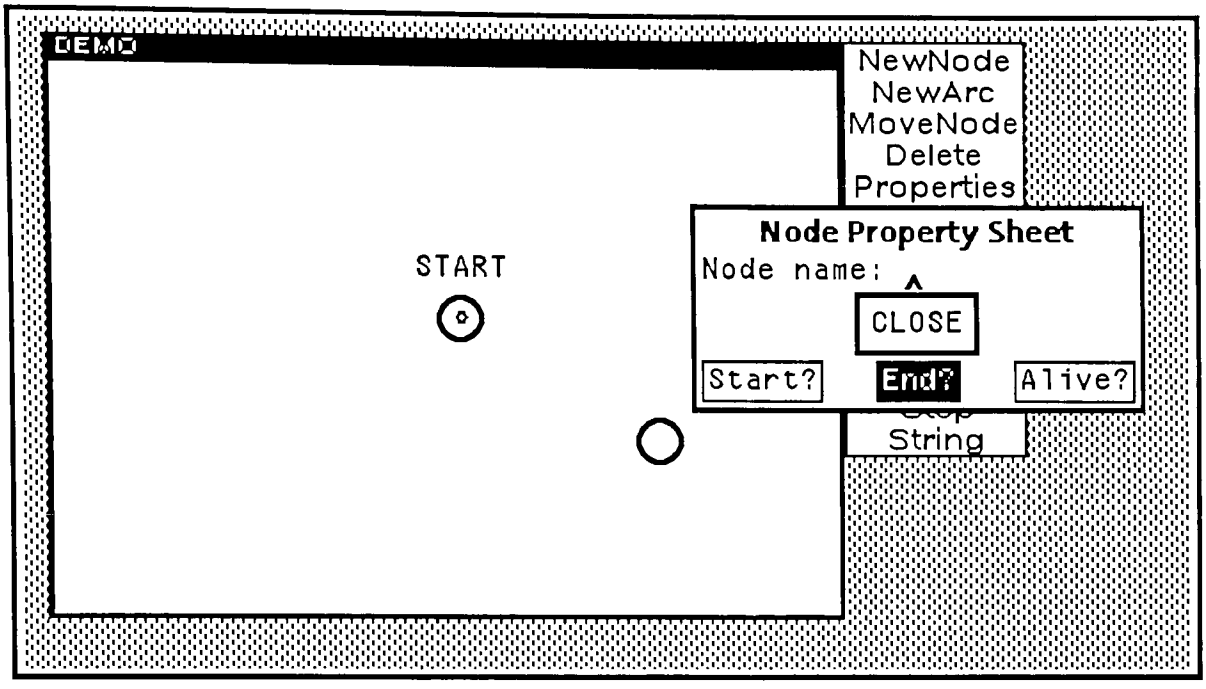

Figure 20: Changing the Properties of a Node

Figure 21 shows the effect on the main design window when the node property sheet window is closed. Node that the new node has been marked with an inner circle. This indicates that the node is an ending node.

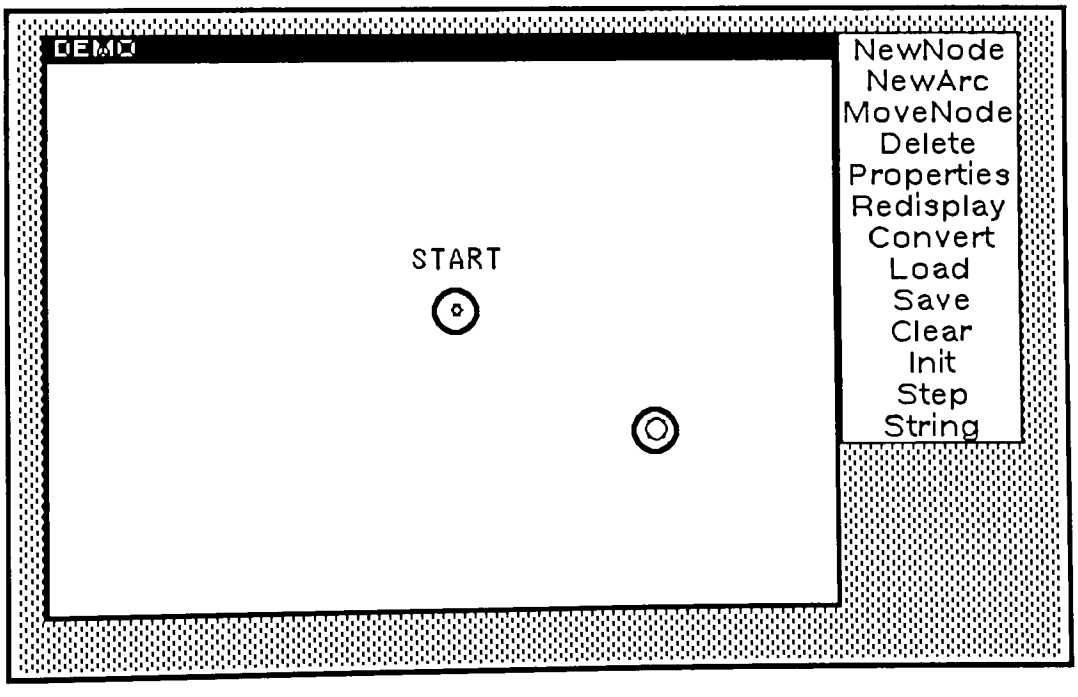

Figure 21: The Node after Alteration

Figure 22 shows the display after two nodes have been selected. The first node will be the source of a new arc, and the second node will be the destination.

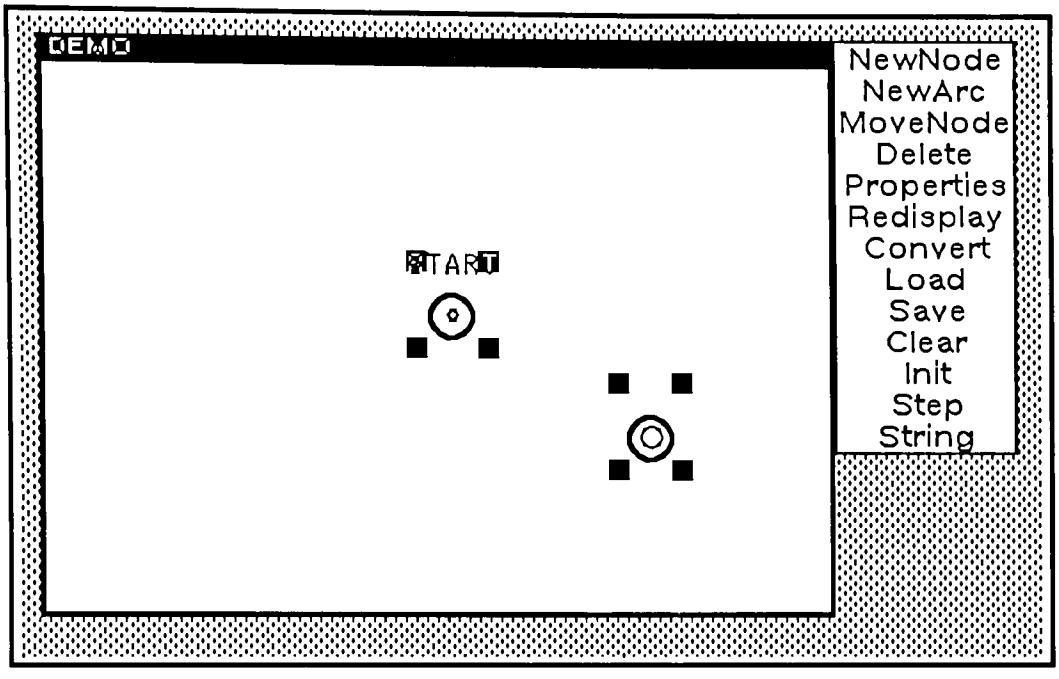

Figure 22: Selecting Two Nodes

Figure 23 shows the newly created arc. This arc was created by selecting a source and a destination node and then selecting the "NewArc" command menu item.

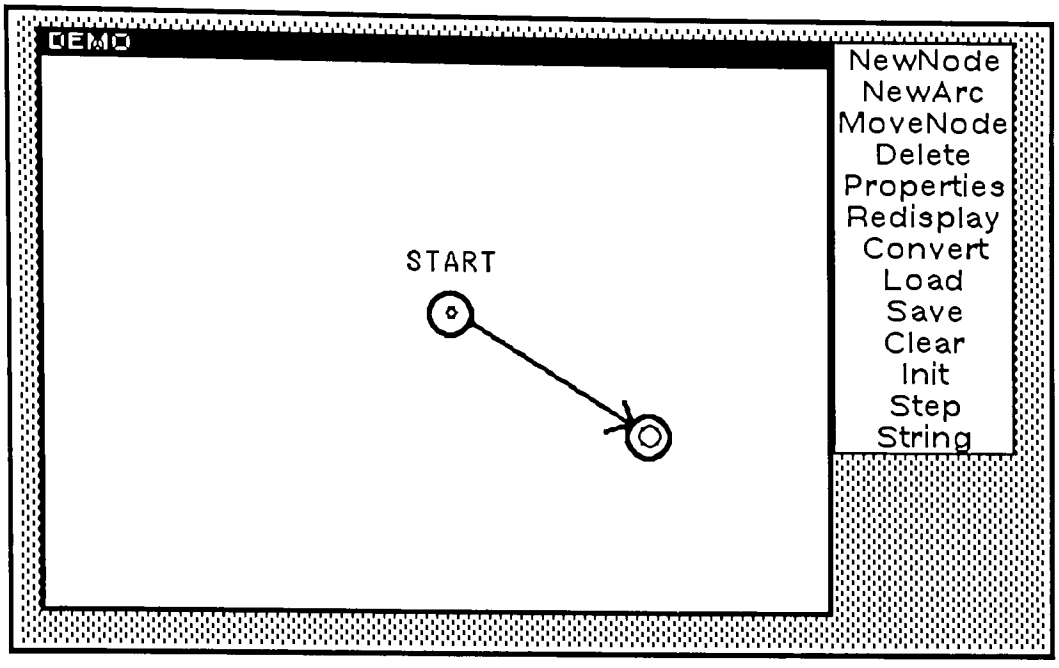

Figure 23: Creating an Arc Between Nodes

As with the case of the new node, the new arc also has a property sheet that is altered in Figure 24. The designer this arc to allow transitions for numerals. Therefor he indicates the string "0-9" in the Arc Property Sheet under Arc Name. As discussed in Section 2.4.6, this indicates a regular express for the character class of numerals.

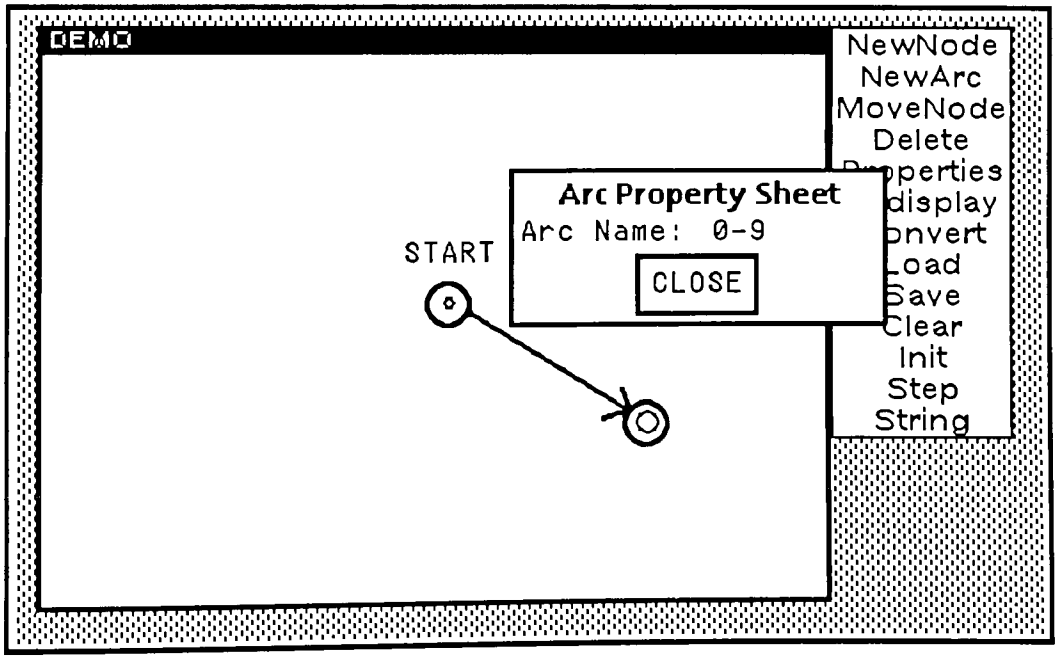

Figure 24: Altering an Arc's Properties

Figure 25 shows the altered arc, correctly labeled with the set of natural counting numbers that are allowed to follow this transition arc.

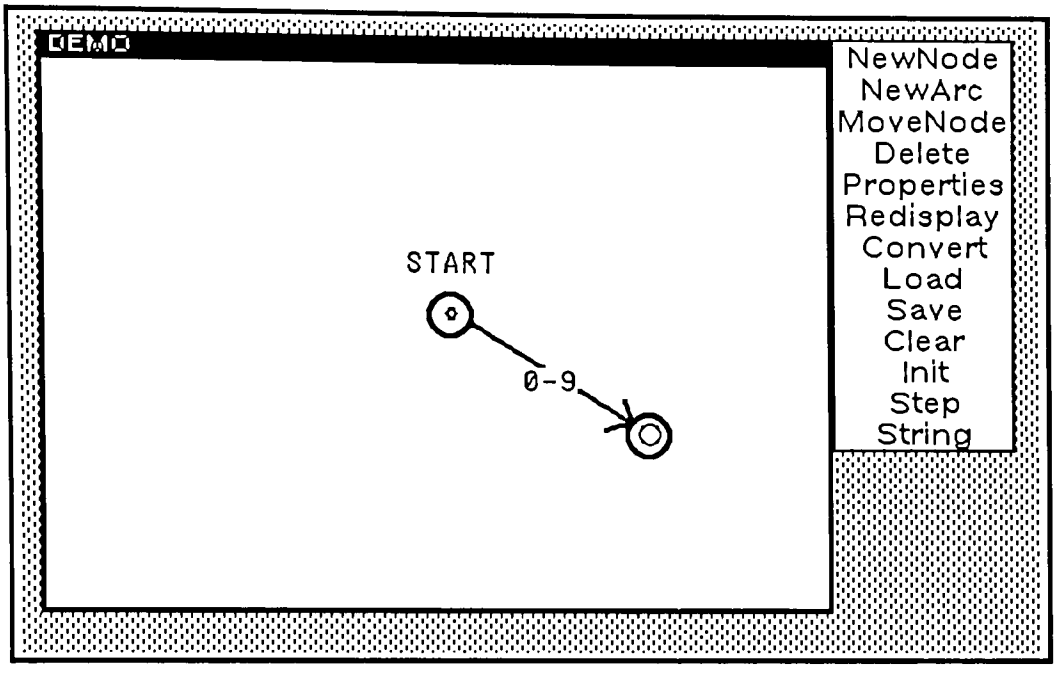

Figure 25: View of Altered Arc

Next, the designer wishes to test the small FSA already created. He does this by selecting "Init", which initializes the parser simulator. During this initialization process, the simulator prompts for a test string to be entered. The status of the parser simulator is show in Figure 26. Note that a second window has been attached to the main design window. This "String" window shows the string to be parsed, along with an arrow indicating the next character in that string to be presented to the parser. Also we see that the START node of the FSA has been highlighted to indicate that it is "alive".

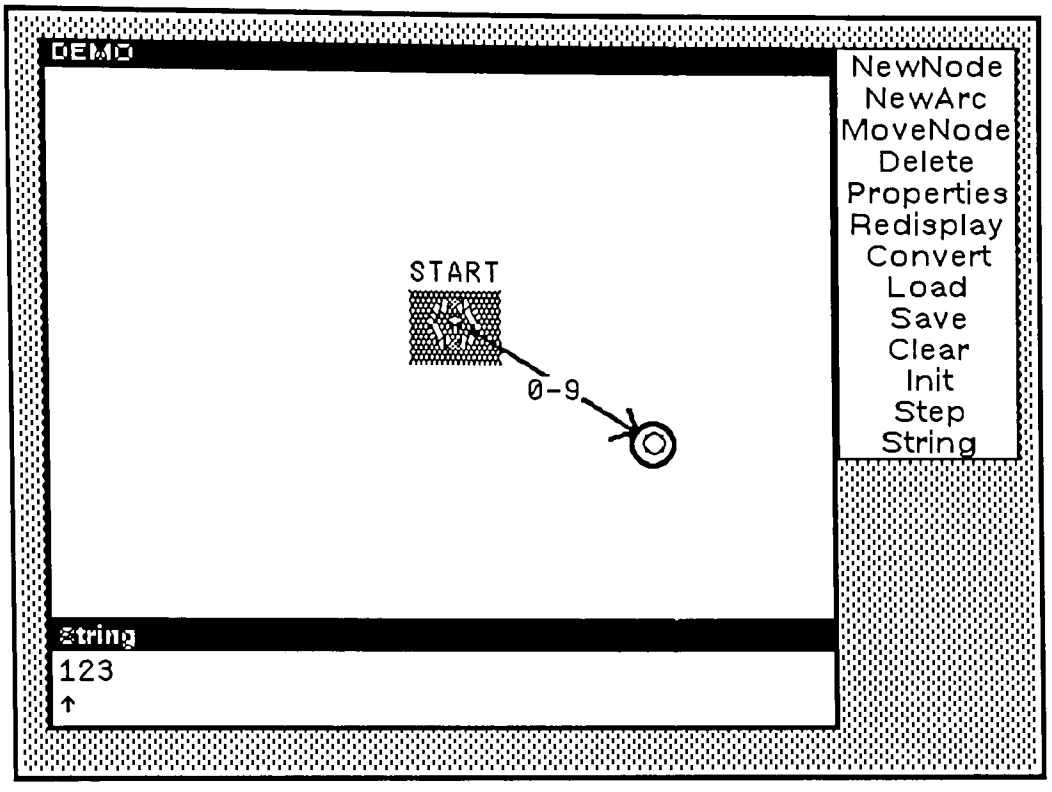

Figure 26: Starting a Test Parse

The designer causes the parser to step along, from input character to input character, by selecting the "Step" menu item. Figure 27 shows the appearance after the first step. Notice that the arrow in the string window has advanced one character. And also notice that the "alive" indicator has progressed from the starting node to the ending node, since the first character of the string was a numeral.

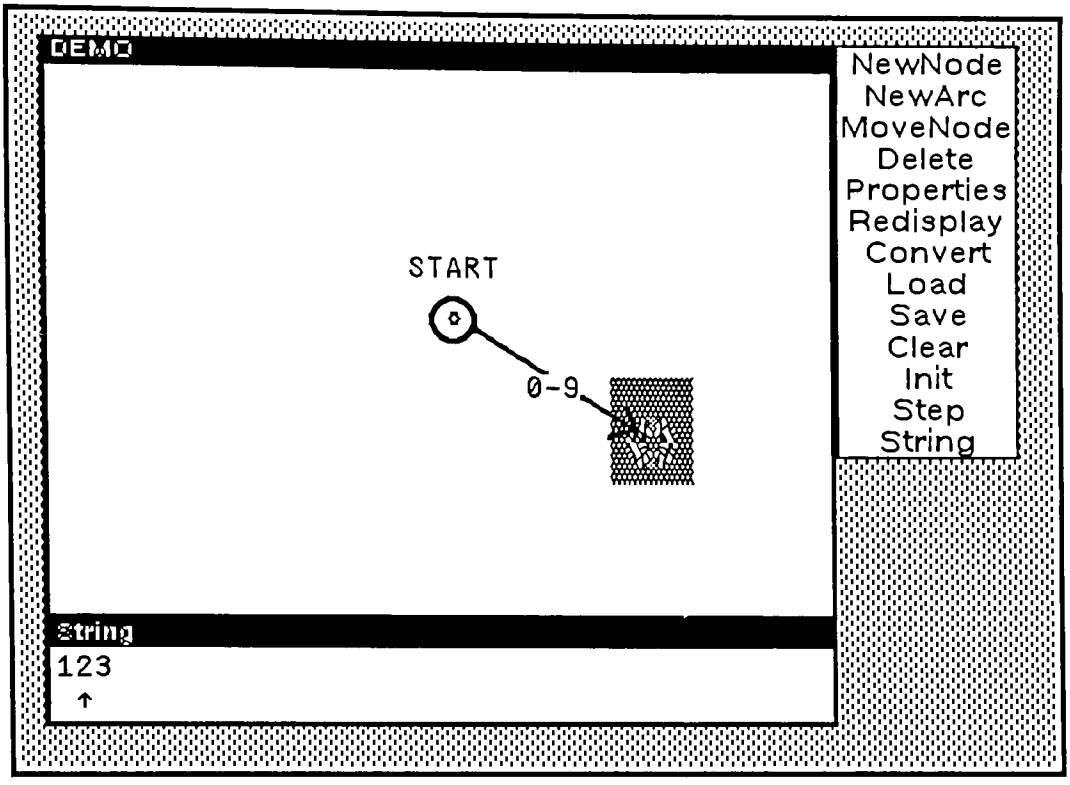

Figure 27: Second Step

However, Figure 28, after the third step of the parse simulation, shows a terrible flaw in the design. All nodes within the FSA have "died". Clearly, the ending node was incorrectly specified; an arc is missing.

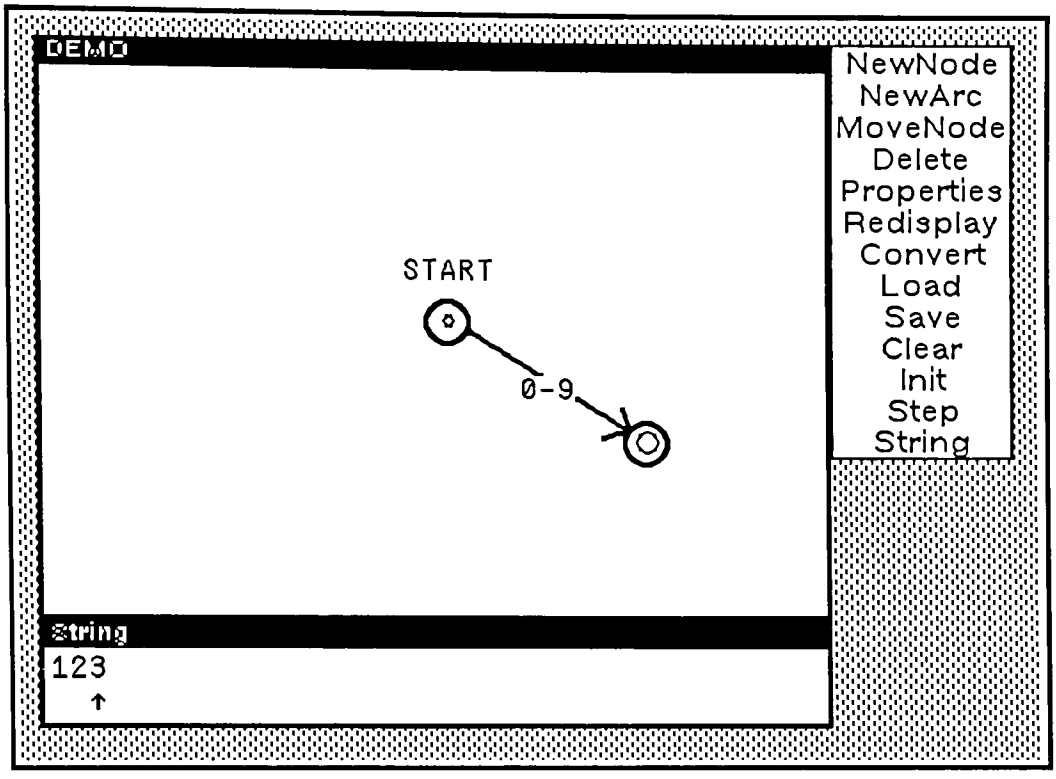

Figure 28: Third Step

Realizing his mistake, the designer in Figure 29 has modified the design to include an arc from the ending node to itself. This should allow the FSA to recognize all numeric strings.

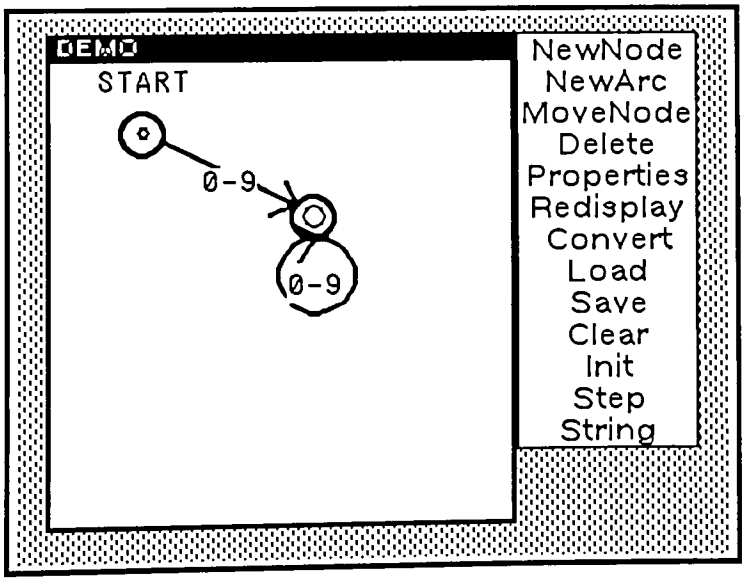

Figure 29: Altered FSA Design

In Figure 30, the designer has rerun the parser simulator. However, this time a successful parse of the input string is indicated by the flashing of the screen and the musical notes produced by the computer on a successful parse. A successful parse is shown in this figure by noting that the string pointer points to the end of the string and that an ending node of the parser is shown as "alive".

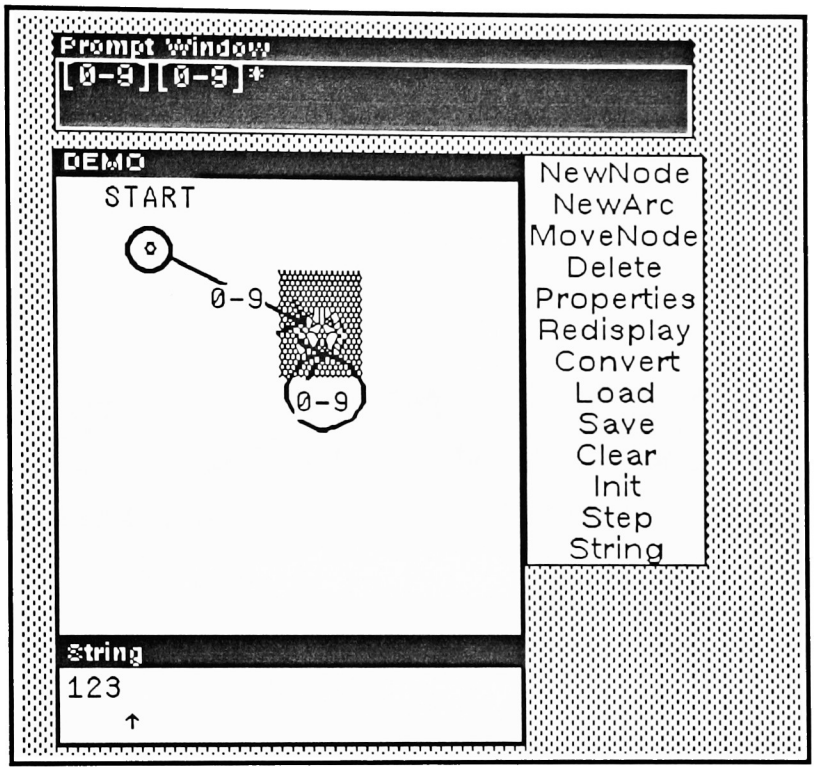

Figure 30: Successful Parse and Equivalent Regular Expression

The designer in Figure 30 has confirmed his reasoning by mousing the "Convert" menu button, producing the regular expression string shown in the Prompt Window at the top of the screen image. This expression tells the designer that this FSA recognizes one numeral followed by any number of numericals.

Proceeding in a similar fashion, the designer next adds a second major part to the FSA design. This time the original ending node is renamed "DECIMAL" and an additional mechanism is added to parse binary strings. Figure 31 shows a successful simulation of the new FSA.

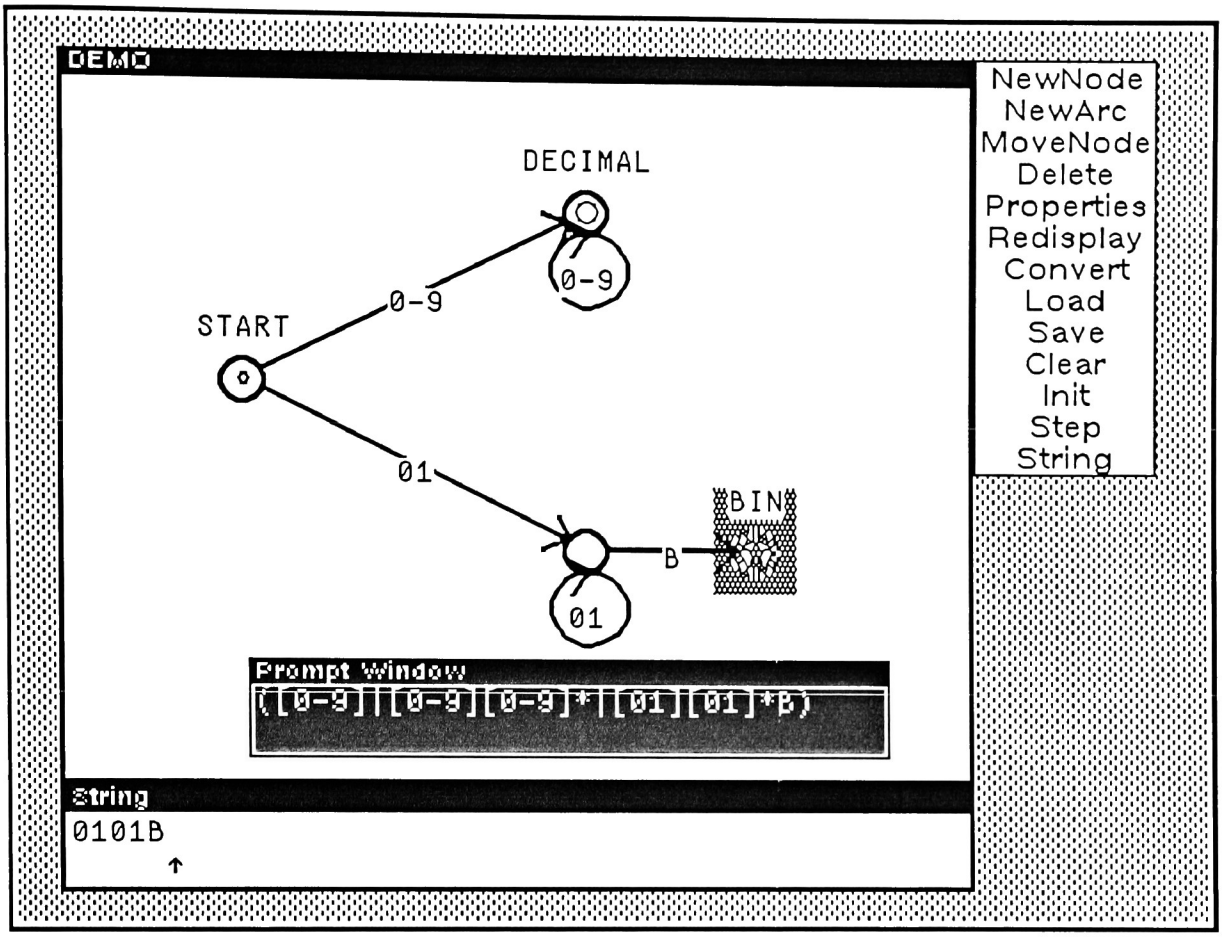

Figure 31: A More Complicated Parse, with Regular Expression

As before, the designer has confirmed the graphic design by mousing the "Convert" menu entry, causing the equivalent regular expression to be written to the Prompt Window. Clearly, this translation to an equivalent regular expression allows this FSA design to be transferred from this graphic framework to the more traditional compiler tool generation world, as might be found in an Unix environment. for instance.

5. Suggestions for Further Work

Producing an operating screen drawing package, coupled with a FSA simulator, only scratches the surface of work possible in this area. Additional topic areas suggest themselves. Among them are operational additions and debugging options.

5.1. Operational Additions

The following additions could be made to the functionality of the Visual Design Aide. These changes would allow greater functionality and improved human interface.

5.1.1. Performance Improvements

Several times during the development of this project we came up against the problem of the computer lack of performance. In Section 3.6 are several discussion of changes that were made in the design of the package to work around the worst of these performance problems. However many additional performance problems still exist. A careful review of data structures as to when they are created, how they are passed and how often would likely improve the performance of the system.

Largely developed on an earlier version of the Interlisp-D package (Koto), the Visual Design Aide was later ported to a later release (Lyric). This port, while providing increased functionality, caused the software to run even slower that before.

5.1.2. Utilization of Sub-Automata

A scheme for implementing a form of sub-automaton would allow the separate specification of large, nested FSA designs. This mechanism allows the designer to modularized and isolate components of the developing design. These sub-automata would be indicated on the graphic display as a closed box.

5.2. Diagnostic Additions

The following section lists diagnostics additions that could be made to the Visual Design Aide. The Visual Design Aide allows the user a great deal of creativity, at the risk of producing parsers that are not understandable. More importantly, subtle design errors can be committed by the designer without an assistance from the design aid.

Among the possible additions in the area of diagnostics are some of the traditional mechanisms for determining parser defects, such as:

(1) detecting unreachable states by digraph tracing;

(2) detecting redundant accepting states;

(3) detecting redundant states of any kind;

(4) detecting ambiguous state transitions;

(5) minimizing the output regular expressions.

The results of these diagnostic checks would be directly displayed on the window surface. Corrections would *not* be automatically applied. Rather, since graphic programming methods are being studied, the designer is notified of the detected problem and possible corrections. The designer implements all remedial changes, if any, to the developing FSA. Thus the user is retained as a key component of the process.

6. Summary

Design and construction of the Visual Design Aide was an interesting and challenging task. A major motivation for this work was to become familiar with a well-known, commercial LISP programming environment.

Formulating efficient updating mechanisms for the visual screen effects proved to be both a challenge and a delight; frequently both at the same time. Working with the flexible and powerful Interlisp-D proved fruitful; most all functions operated as documented and were easy to use. Without reference to the Interlisp-D source codes, determining what a function accomplished sometimes was a puzzle.

The Interlisp-D environment offers a great deal of power and wide variety of programming tools to do an effective job at building such a presentation. Coupled with the built-in automatic management of functions and variables, the work went as smoothly as the designer was able to formulate approaches to problems.

The Interlisp-D management tools are so powerful that, in fact, this project traces a clear progression from the initial sample screen handling functions to the finished project. That is, no code was ever lost, no functions were re-keyed; rather the project grew though constant iteration and refinement of the initial design concept.

On the negative side, the Xerox 1108 machine is frequently so slow that it is cumbersome. However, the small memory size (1.5 megabyte) of the machine available for this project caused very heavy swap loads. In fact, experience with larger Xerox 1186 machines (3.7 megabyte) showed encouraging improvements in performance. Also the porting of the software from the earlier Koto release to the later Lyric version also caused performance degradation. The biggest factor affecting run-time performance appears to be the primary memory size.

In closing then, the Visual Design Aide is a reasonable tool for the beginning designer to experiment with, which was the original intent. A great many things could be done to increase its effectiveness, though it will always remain speed limited on the slow processor for which it was implemented.

Appendix A. References

[Aho-72] *The Theory of Parsing, Translation, and Compiling, Volume I: Parsing,* Alfred V. Aho and Jeffrey D. Ullman, Prentice-Hall (1972).

[Aho-73] *The Theory of Parsing, Translation, and Compiling, Volume II: Compiling*, Alfred V. Aho and Jeffrey D. Ullman, Prentice-Hall (1973).

[Aho-77] *Principles of Compiler Design*, Alfred V. Aho and Jeffrey D. Ullman, Addison-Wesley (1977).

[Anklam-82] *Engineering a Compiler: VAX-11 Code Generation and Optimization*, Patricia Anklam, David Cutler, Roger Heinen, Jr., M. Donald MacLaren, Digital Press (1982).

[Brooks-75] *The Mythical Man-Month: Essays on Software Engineering*, Frederick P. Brooks, Jr. Addison-Wesley (1975).

[Brooks-75] "No Silver Bullets", Frederick P. Brooks, Jr. *Unix Review,* pp 39-48, (November 1987).

[Brown-85] "Program Visualization: Graphical Support for Software Development", Gretchen P. Brown, Richard T. Carling, Christopher F. Herot, David A. Kramlich, Paul Souza, *Computer* Vol. 18, No. 8, pp. 27-35 (August 1985).

[Cherry-86] *The PAMELA Designer's Handbook, Volume 1: Commentary and Ada PDL and Code*, George W. Cherry, Thought**Tools, Inc., Reston, Virginia (1986).

[Goldberg-84] *Smalltalk-80: The Interactive Programming Environment*, Adele Goldberg, Addison-Wesley (1984).

[Hohmeyer-71] *1700 Meta Compiler*, Robert E. Hohmeyer, La Jolla Systems Division, Control Data Corporation, Manual #3949300 (May 1971).

[HOS-85] *The USE.IT System: System Reference Manual, VAX--Release 3.0*, Higher Order Software, Inc, Cambridge, Massachusetts (July 1985).

[IBM-64] *IBM System/360 Reference Data*, International Business Machines Corporation, Form GX20-1703 (April 1964).

[Jacob-85] "A State Transition Diagram Language for Visual Programming", Robert J. K. Jacob, *Computer*, Vol. 18, No. 8, pp. 51-59 (August 1985).

[Kaisler-86] *INTERLISP: The Language and Its Usage*, Stephan H. Kaisler, John Wiley & Sons (1986).

[Kitchen-84] *The GSHELL: A Graphical Command Language for the UNIX Operating System,* Andrew Kitchen, Rochester Institute of Technology (1984).

[Lesk-75] "LEX -- A lexical Analyzer Generator", M. E. Lesk and E. Schmidt, *Computer Science Technical Report,* No. 39, Bell Laboratories, Murray Hill, New Jersey (October 1975).

[Lewis-76] *Compiler Design Theory*, Phillip M. Lewis II, Daniel J. Rosenkrantz, Richard E. Stearns, Addison-Wesley (1976).

[McKeeman-70] *A Compiler Generator*, William M. McKeeman, James J. Horning, David B. Wortman, Prentice-Hall (1970).

[Moriconi -85] "Visualizing Program Design Through PegaSys", Mark Moriconi and Dwight F. Hare, *Computer*, Vol. 18, No. 8, pp. 72-85 (August 1985).

[Quayle-84] *The Friendly Dandelion Primer*, Mary Ann Quayle, William Weil, Jeffrey Bonar and Alan Lesgold, Learning Research and Development Center, University of Pittsburgh, (November 1984, updated November 1985).

[Xerox-85] *Interlisp-D Reference Manual*, 3101272 (Vol I: *Language*), 2101273 (Vol II: *Environment*) and 3101274 (Vol III: *Input/Output)* (October 1985).

[Wirth-71] "The Design of a Pascal Compiler", N. Wirth, *Software--Practice and Experience*, Vol. 1, pp. 309-333 (1971).

Appendix B. Glossary

accepting state: one or more distinguished states of an FSA, where the machine is said to have accepted or recognized the input string.

deterministic FSA: a FSA is said to *deterministic* if there is at most one next transition for each {state, input symbol} in the machine definition.

Finite State Automaton (FSA): a finite automaton without a stack and with enough memory to record a current machine state. The FSA may be either deterministic or nondeterministic.

graphic programming method: any graphic scheme for coding a computer algorithm in a two-dimensional form. That is, icons are placed on a surface and are connected by directed arcs. Together these nodes and arcs describe a computer algorithm.

icon: a graphic representation of an object or concept.

instantiation: a concrete example of an abstract concept; a specific instance of a general class; an executing process.

INTERLISP: a specific dialect of LISP, implemented on the DECSystem 10/20 under TENEX and TOPS-20, on the VAX-11/780 under UNIX and VMS, on the IBM 30XX series under VM/SP and on the Xerox 1100 family of scientific information processors.

Mesa: A programming language, descended from Wirth's Pascal, intended for strongly-typed systems programming. As with several later modular languages, such as Modula-2 and Ada, Mesa presents an interface to the world as a series of imported module interfaces.

nondeterministic FSA: a FSA is said to be *nondeterministic* if there are more than one transition for each {state, input symbol} in the machine definition.

regular expression: a string which is constructed over an alphabet using the operations of concatenation, union and closure.

starting state: a distinguished state of an FSA, where the machine begins operation.

string: a finite length, ordered list of tokens.

token: an element of an alphabet, also frequently a character, such as an ASCII character; an atom, especially as a class of enumerated input atoms.

ViewPoint: An environment for the Xerox workstations that emphasizes ease of use and graphic images for users.

Xerox Development Environment (XDE): A programming environment supported on graphics workstations, such as the Dandelion.

Appendix C. Source Listings

In this section are the complete listings of the Interlisp-D source. Source programs for the Visual Display Aide are arranged into four files.

These files are:

File	Description
DRAW	code concerned with drawing objects on the screen
FSA	code concerned with running the FSA simulator
INIT	initializations code for global settings
WINDOW	code concerned with drawing windows and handling mouse events

C.1 DRAW Source Listing

```
(DEFINE-FILE-INFO ^^PACKAGE "INTERLISP" ^^READTABLE "INTERLISP" ^^BASE 10)
(FILECREATED " 3-Jun-88 14:05:08"
     |{POLLUX:WBST208P:XEROX}<NORMAN CROWFOOT>THESIS>DRAW.;9|

      previous date%: " 2-Jun-88 11:09:11" {DSK}<LISPFILES>DRAW.;1)


(* "Copyright (c) 1987, 1988 by Norman C. Crowfoot.  All rights reserved.")

(PRETTYCOMPRINT DRAWCOMS)

(RPAQQ DRAWCOMS
       ((FNS BEND.ARC BIAS.POS CALC.ARC.ARROW CALC.ARC.BOUND CALC.ARC.DYNAMIC
             CALC.ARC.HOTSPOT CALC.ARC.KNOTS CALC.ARC.STRINGBOX
             CALC.NODE.BOUND CENTROID CREATE.ARC CREATE.NODE DISTANCE
             DO.DELETE DO.DELETE.ARC DO.DELETE.NODE DO.MENU.ITEM DO.MOVE.NODE
             DO.NEWARC DO.PROPERTY DO.PROPERTY.CLOSE DO.PROPERTY.SETUP
             DO.SELECT DO.SNAP.GRID EXISTING.ARC INIT.DRAW INIT.MENU
             INIT.MENU.TEXT LINE.OFFSET PAINT.ARC PAINT.NODE PAINT.SELECT
             SELECT.OBJECT UNBEND.ARC)
        (RECORDS ARCOBJ DRAWPROCS NODEOBJ)
        (PROP COPYRIGHT DRAW)
        (VARS GRID)))
(DEFINEQ

(BEND.ARC
  [LAMBDA (ARC)                                (* ; "Edited  1-Jun-88 20:13 by NCC")

(* ;;; "bend an existing line into three knots")

    (LET* ((KNOTS (fetch KNOTS of ARC))
           (ONE (CAR KNOTS))
           (TWO (CAR (LAST KNOTS)))
           (X1 (fetch XCOORD of ONE))
           (Y1 (fetch YCOORD of ONE))
           (X2 (fetch XCOORD of TWO))
           (Y2 (fetch YCOORD of TWO))
           (DX (QUOTIENT (IDIFFERENCE X2 X1) 1.414))
           (DY (QUOTIENT (IDIFFERENCE Y2 Y1) 1.414))
           (MX (FIX (PLUS (TIMES 0.707 DX) TIMES 0.707 DY) X1)))
           (MY (FIX (PLUS (TIMES -0.707 DX) (TIMES 0.707 DY) Y1)))
           (NEWKNOTS (LIST ONE (CREATEPOSITION MX MY) TWO))
           (NEWHOT (CALC.ARC.HOTSPOT NEWKNOTS)))
          (replace KNOTS of ARC with NEWKNOTS)
          (replace HOTSPOT of ARC with NEWHOT)
          (CALC.ARC.ARROW ARC)
          (CALC.ARC.STRINGBOX ARC])

(BIAS.POS
  [LAMBDA (X Y)                                (* ; "Edited  9-May-88 18:43 by NCC")

(* ;;; "bias a doted pair by another doted pair")

    (CONS (PLUS (CAR X) (CAR Y)) (PLUS (CDR X) (CDR Y])
```

```
(CALC.ARC.ARROW
  [LAMBDA (ARC)                                    (* ; "Edited  1-Jun-88 18:28 by NCC")

(* ;;; "calculate arrow heads for arc")

    (LET* [(RKNOTS (REVERSE (fetch KNOTS of ARC)))
           (SRCCENT (CADR RKNOTS))
           (DSTCENT (CAR RKNOTS))
           (DSTX (fetch XCOORD of DSTCENT))
           (DSTY (fetch YCOORD of DSTCENT))
           (C (LINE.OFFSET SRCCENT DSTCENT 10))
           (CX (IDIFFERENCE (fetch XCOORD of C) DSTX))
           (CY (IDIFFERENCE (fetch YCOORD of C) DSTY))
           (DX (FIX (PLUS (TIMES 0.707 CX) (TIMES 0.707 CY) DSTX)))
           (DY (FIX (PLUS (TIMES -0.707 CX) (TIMES 0.707 CY) DSTY)))
           (EX (FIX (PLUS (DIFFERENCE (TIMES 0.707 CX) (TIMES 0.707 CY))
                          DSTX)))
           (EY (FIX (PLUS (TIMES 0.707 CX) (TIMES 0.707 CY) DSTY]
          (replace ARROW of ARC with (LIST (CREATEPOSITION DSTX DSTY)
                                           (CREATEPOSITION DX DY)
                                           (CREATEPOSITION EX EY])

(CALC.ARC.BOUND
  [LAMBDA (WINDOW ARC)                             (* ; "Edited 13-May-88 13:30 by NCC")

(* ;;; "calculate bounding box for arc")

    (LET* [(KNOTS (fetch KNOTS of ARC))
           (ARROW (fetch ARROW of ARC))
           (POINTS (APPEND KNOTS ARROW))
           (XCOORDS (for POINT in POINTS collect (fetch XCOORD of POINT)))
           (YCOORDS (for POINT in POINTS collect (fetch YCOORD of POINT)))
           (LEFT (DIFFERENCE (APPLY (FUNCTION MIN) XCOORDS) 2))
           (RIGHT (PLUS (APPLY (FUNCTION MAX) XCOORDS) 2))
           (BOTTOM (DIFFERENCE (APPLY (FUNCTION MIN) YCOORDS) 2))
           (TOP (PLUS (APPLY (FUNCTION MAX) YCOORDS) 2))
           (WIDTH (DIFFERENCE RIGHT LEFT))
           (HEIGHT (DIFFERENCE TOP BOTTOM))
           (STRINGBOX (fetch STRINGBOX of ARC))
           (BOUNDBOX (UNIONREGIONS STRINGBOX
                                   (CREATEREGION LEFT BOTTOM WIDTH HEIGHT]
          (replace BOUNDBOX of ARC with BOUNDBOX)
          (WINDOWPROP WINDOW 'EXTENT (UNIONREGIONS (WINDOWPROP WINDOW 'EXTENT)
                                                   BOUNDBOX])

(CALC.ARC.DYNAMIC
  [LAMBDA (WINDOW ARC)                             (* ; "Edited 13-May-88 13:31 by NCC")

(* ;;; "calculate dynamic properties of an arc")

    (LET* ((SRCNODE (fetch ARCSRC of ARC))
           (DSTNODE (fetch ARCDST of ARC))
           REVERSE)
          (if (EQ SRCNODE DSTNODE) then
              (replace HOTSPOT of ARC with
                   (CALC.ARC.HOTSPOT (fetch KNOTS of ARC)))
           else (if (SETQ REVERSE (EXISTING.ARC WINDOW DSTNODE SRCNODE))
```

```
                    then (BEND.ARC ARC)
                         (BEND.ARC REVERSE)
                         (CALC.ARC.BOUND WINDOW REVERSE)
                   else (UNBEND.ARC ARC)))
          (CALC.ARC.BOUND WINDOW ARC])

(CALC.ARC.HOTSPOT
  [LAMBDA (KNOTS)                                (* ; "Edited 13-May-88 13:32 by NCC")

(* ;;; "function to calculate hot spot for an arc")

    (SELECTQ (LENGTH KNOTS)
        (2 (CENTROID KNOTS))
        (3 (CADR KNOTS))
        (CENTROID KNOTS])

(CALC.ARC.KNOTS
  [LAMBDA (SRCNODE DSTNODE)                      (* ; "Edited 13-May-88 13:32 by NCC")

(* ;;; "calculate knot points for arc")

    (PROG*
     ((SRCHOT (fetch HOTSPOT of SRCNODE))
      (DSTHOT (fetch HOTSPOT of DSTNODE)))
     (RETURN
      (if (EQ SRCNODE DSTNODE)
          then [LET* ((N 18)
                      (DELTA (QUOTIENT 360 N))
                      (OFFSET 15)
                      (SRCCENT (BIAS.POS SRCHOT (CONS 8 0)))
                      (CENTER (BIAS.POS SRCCENT (CONS OFFSET 0)))
                      (SRCX (fetch XCOORD of CENTER))
                      (SRCY (fetch YCOORD of CENTER)))
                     (for I from 0 to N collect
                          (CONS [PLUS SRCX (FIX (PLUS 0.5
                                    (TIMES OFFSET
                                    (SIN (PLUS (TIMES I DELTA) 270]
                                PLUS SRCY (FIX (PLUS 0.5
                                    (TIMES OFFSET
                                    (COS (PLUS (TIMES I DELTA) 270]
        else (LET* ((SRCCENT (LINE.OFFSET DSTHOT SRCHOT 8))
                    (DSTCENT (LINE.OFFSET SRCHOT DSTHOT 8)))
                   (LIST SRCCENT DSTCENT])

(CALC.ARC.STRINGBOX
  [LAMBDA (ARC)                                  (* ; "Edited 31-May-88 11:21 by NCC")

(* ;;; "calculate stringbox of an arc")

    (LET* [(HOTSPOT (fetch HOTSPOT of ARC))
           (STRINGWIDTH (STRINGWIDTH (fetch ARCSTRING of ARC)))
           (STRINGHEIGHT (FONTPROP NIL 'HEIGHT))
           (LEFT (IDIFFERENCE (fetch XCOORD of HOTSPOT)
                              (IQUOTIENT STRINGWIDTH 2)))
           (BOTTOM (IDIFFERENCE (fetch YCOORD of HOTSPOT)
                                (IQUOTIENT STRINGHEIGHT 2]
          (replace STRINGBOX of ARC with
               (CREATEREGION LEFT BOTTOM STRINGWIDTH STRINGHEIGHT])
```

```
(CALC.NODE.BOUND
  [LAMBDA (WINDOW NODE)                          (* ; "Edited 31-May-88 11:20 by NCC")

(* ;;; "calculate bounding box for node")

    (LET* [(HOTSPOT (fetch HOTSPOT of NODE))
           (LOCX (fetch XCOORD of HOTSPOT))
           (LOCY (fetch YCOORD of HOTSPOT))
           (NODENAME (fetch NODENAME of NODE))
           (WIDTH (MAX 32 (STRINGWIDTH NODENAME)))
           (LEFT (IDIFFERENCE LOCX (IQUOTIENT WIDTH 2)))
           [HEIGHT (PLUS 32 (FONTPROP NIL 'ASCENT]
           (BOTTOM (IDIFFERENCE LOCY (IQUOTIENT 32 2)))
           (BOUNDBOX (CREATEREGION LEFT BOTTOM WIDTH HEIGHT))
           (BORDER 50)
           (BIGBOX (CREATEREGION (IDIFFERENCE LEFT BORDER)
                                 (IDIFFERENCE BOTTOM BORDER)
                                 (IPLUS WIDTH BORDER BORDER BORDER)
                                 (IPLUS HEIGHT BORDER]
          (replace BOUNDBOX of NODE with BOUNDBOX)
          (WINDOWPROP WINDOW 'EXTENT (UNIONREGIONS (WINDOWPROP WINDOW 'EXTENT)
                                                   BIGBOX])

(CENTROID
  [LAMBDA (POINTS)                               (* ; "Edited 31-May-88 11:23 by NCC")

(* ;;; "function to return the center of a list of points")

    (PROG* ((XPTS (for POINT in POINTS collect (fetch XCOORD of POINT)))
            (YPTS (for POINT in POINTS collect (fetch YCOORD of POINT)))
            (MINX (APPLY (FUNCTION MIN) XPTS))
            (MAXX (APPLY (FUNCTION MAX) XPTS))
            (MINY (APPLY (FUNCTION MIN) YPTS))
            (MAXY (APPLY (FUNCTION MAX) YPTS)))
           (RETURN (CREATEPOSITION (IQUOTIENT (IPLUS MINX MAXX) 2)
                                   (IQUOTIENT (IPLUS MINY MAXY) 2])

(CREATE.ARC
  [LAMBDA (WINDOW SRCNODE DSTNODE STRING)(* ; "Edited 13-May-88 13:34 by NCC")

(* ;;; "create ARC record with arrowhead")

    (PROG* ((KNOTS (CALC.ARC.KNOTS SRCNODE DSTNODE))
            (NEWARC (create ARCOBJ
                            OBJTYPE _ 'ARC
                            BOUNDBOX _ NIL
                            DRAWPROCS _ (WINDOWPROP WINDOW 'ARCPROCS)
                            ARCSTRING _ STRING
                            STRINGBOX _ NIL
                            HOTSPOT _ (CALC.ARC.HOTSPOT KNOTS)
                            ARCSRC _ SRCNODE
                            ARCDST _ DSTNODE
                            KNOTS _ KNOTS
                            ARROW _ NIL)))
           (CALC.ARC.ARROW NEWARC)
           (CALC.ARC.STRINGBOX NEWARC)
           (if (NOT (EXISTING.ARC WINDOW SRCNODE DSTNODE))
```

```
            then (CALC.ARC.DYNAMIC WINDOW NEWARC)
                 (WINDOWADDPROP WINDOW 'OBJECTLIST NEWARC)
                 (RETURN NEWARC)
          else (RETURN NIL])

(CREATE.NODE
  [LAMBDA (WINDOW NAME LOCX LOCY IF.START IF.END)
                                        (* ; "Edited 13-May-88 13:35 by NCC")

(* ;;; "create new node")

    (PROG* ((NEWNODE (create NODEOBJ
                             OBJTYPE _ 'NODE
                             DRAWPROCS _ (WINDOWPROP WINDOW 'NODEPROCS)
                             NODENAME _ NAME
                             HOTSPOT _ (CREATEPOSITION LOCX LOCY)
                             IFSTART _ IF.START
                             IFEND _ IF.END)))
           (CALC.NODE.BOUND WINDOW NEWNODE)
           (WINDOWPROP WINDOW 'EXTENT
                (UNIONREGIONS (WINDOWPROP WINDOW 'EXTENT)
                                        (fetch BOUNDBOX of NEWNODE)))
           (WINDOWADDPROP WINDOW 'OBJECTLIST NEWNODE)
           (RETURN NEWNODE])

(DISTANCE
  [LAMBDA (X Y LOC)                     (* ; "Edited 31-May-88 11:22 by NCC")

(* ;;; "return manhatton distance from point to location")

    (PLUS (ABS (IDIFFERENCE X (fetch XCOORD of LOC)))
          (ABS (IDIFFERENCE Y (fetch YCOORD of LOC])

(DO.DELETE
  [LAMBDA (WINDOW)                      (* ; "Edited 13-May-88 13:35 by NCC")

(* ;;; "delete any selected object")

    (LET* ((SELECTLIST (WINDOWPROP WINDOW 'SELECTLIST)) BOUNDS)
          (for OBJECT in SELECTLIST do
               (SETQ BOUNDS (fetch BOUNDBOX of OBJECT))
               [WINDOWPROP WINDOW 'OBJECTLIST
                    (REMOVE OBJECT (WINDOWPROP WINDOW 'OBJECTLIST]
               (SETQ BOUNDS
                    (SELECTQ (fetch OBJTYPE of OBJECT)
                         (NODE (DO.DELETE.NODE WINDOW OBJECT))
                         (ARC (DO.DELETE.ARC WINDOW OBJECT))
                          NIL))
               (WINDOWPROP WINDOW 'SELECTLIST NIL)
               (REPAINT.MAIN.WINDOW WINDOW BOUNDS])

(DO.DELETE.ARC
  [LAMBDA (WINDOW ARC)                  (* ; "Edited 13-May-88 13:36 by NCC")

(* ;;; "delete arc")

    (PROG* ((BOUNDBOX (fetch BOUNDBOX of ARC))
            (ARCSRC (fetch ARCSRC of ARC))
```

```
            (ARCDST (fetch ARCDST of ARC))
            (BUDDY (EXISTING.ARC WINDOW ARCDST ARCSRC)))
           [if BUDDY
               then (SETQ BOUNDBOX
                          (UNIONREGIONS BOUNDBOX (fetch BOUNDBOX of BUDDY)))
                    (CALC.ARC.DYNAMIC WINDOW BUDDY)
                    (SETQ BOUNDBOX
                          (UNIONREGIONS BOUNDBOX (fetch BOUNDBOX of BUDDY]
           (RETURN BOUNDBOX])

(DO.DELETE.NODE
  [LAMBDA (WINDOW NODE)                    (* ; "Edited 13-May-88 13:36 by NCC")

(* ;;; "delete a node and all references")

    (PROG [(BOUNDS (fetch BOUNDBOX of NODE))
           (OBJECTLIST (WINDOWPROP WINDOW 'OBJECTLIST]
          [for ARC in OBJECTLIST when (ARCP ARC)
             do (LET ((SRCARC (fetch ARCSRC of ARC))
                      (DSTARC (fetch ARCDST of ARC)))
                     (if (OR (EQ NODE SRCARC)
                             (EQ NODE DSTARC))
                         then (SETQ OBJECTLIST (REMOVE ARC OBJECTLIST))
                              (SETQ BOUNDS (UNIONREGIONS BOUNDS
                                                 (fetch BOUNDBOX of ARC]
          (WINDOWPROP WINDOW 'OBJECTLIST OBJECTLIST)
          (RETURN BOUNDS])

(DO.MENU.ITEM
  [LAMBDA (WINDOW ACTION)                  (* ; "Edited 13-May-88 13:36 by NCC")

(* ;;; "handle menu selection event")

    (LET ((NEWMODE 'SELECT)
          (NEWCURSOR T))
         (SELECTQ ACTION
             (NewNode (SETQ NEWMODE 'NEWNODE)
                      (SETQ NEWCURSOR CROSSHAIRS))
             (NewArc [LET ((NEWARC (DO.NEWARC WINDOW)))
                          (if NEWARC
                              then (REPAINT.MAIN.WINDOW WINDOW
                                                        (fetch BOUNDBOX of NEWARC])
             (MoveNode (DO.MOVE.NODE WINDOW))
             (Delete (DO.DELETE WINDOW))
             (Properties (DOPROPS WINDOW))
             (Redisplay (RESHAPE.MAIN.WINDOW WINDOW))
             (Convert (DO.CONVERT WINDOW))
             (Load (DO.FILE.OPS WINDOW 'LOAD))
             (Save (DO.FILE.OPS WINDOW 'SAVE))
             (Clear (DO.SIMULATE.CLEAR WINDOW))
             (Init (DO.SIMULATE.INIT WINDOW))
             (Step (DO.SIMULATE.STEP WINDOW))
             (String (WINDOWPROP WINDOW 'STRING NIL)
                     (DO.SIMULATE.STRING WINDOW))
             (PROGN NIL))
         (WINDOWPROP WINDOW 'MODE NEWMODE)
         (CURSOR NEWCURSOR])
```

```
(DO.MOVE.NODE
  [LAMBDA (WINDOW)                                    (* ; "Edited 13-May-88 13:37 by NCC")

(* ;;; "allow a user to move the location of a node")

    (LET* ((OBJECTS (WINDOWPROP WINDOW 'OBJECTLIST))
           (SELECTS (WINDOWPROP WINDOW 'SELECTLIST))
           (ARCS (for ARC in OBJECTS when (ARCP ARC) collect ARC))
           OLDNODE NEWNODE OLDARCS POSITION BOUND)
          (if (AND (NODEP (CAR SELECTS))
                   (EQ (LENGTH SELECTS) 1))
              then (SETQ OLDNODE (CAR SELECTS))
                   (SETQ OLDARCS
                        (for ARC in ARCS
                             when (OR (EQ OLDNODE (fetch ARCSRC of ARC))
                                      (EQ OLDNODE (fetch ARCDST of ARC)))
                             collect ARC))
                   (PROMPTPRINT "Specify new location")
                   (SETQ POSITION (DO.SNAP.GRID (GETPOSITION WINDOW)))
                   (CLRPROMPT)
                   (DO.DELETE WINDOW)
                   (SETQ NEWNODE (CREATE.NODE WINDOW
                                              (fetch NODENAME of OLDNODE)
                                              (fetch XCOORD of POSITION)
                                              (fetch YCOORD of POSITION)
                                              (fetch IFSTART of OLDNODE)
                                              (fetch IFEND of OLDNODE)))
                   (SETQ BOUND (fetch BOUNDBOX of NEWNODE))
                   [for A in OLDARCS
                      do (LET ((ARCSRC (if (EQ OLDNODE (fetch ARCSRC of A))
                                           then NEWNODE
                                         else (fetch ARCSRC of A)))
                               (ARCDST (if (EQ OLDNODE (fetch ARCDST of A))
                                           then NEWNODE
                                         else (fetch ARCDST of A)))
                               NEWARC)
                              (SETQ NEWARC
                                   (CREATE.ARC WINDOW ARCSRC ARCDST
                                               (fetch ARCSTRING of A)))
                              (SETQ BOUND (UNIONREGIONS BOUND
                                                        (fetch BOUNDBOX of NEWARC]
                   (REPAINT.MAIN.WINDOW WINDOW BOUND])

(DO.NEWARC
  [LAMBDA (WINDOW)                                    (* ; "Edited 13-May-88 13:37 by NCC")

(* ;;; "create a new arc")

    (PROG ((SELECTED (WINDOWPROP WINDOW 'SELECTLIST))
           SRC DST)
          (if (EQ 2 (LENGTH SELECTED))
              then (PAINT.SELECT WINDOW)
                   (WINDOWPROP WINDOW 'SELECTLIST NIL)
                   (SETQ SRC (CADR SELECTED))
                   (SETQ DST (CAR SELECTED))
                   (if (AND (EQ 'NODE (fetch OBJTYPE SRC))
                            (EQ 'NODE (fetch OBJTYPE of DST)))
                       then (RETURN (CREATE.ARC WINDOW SRC DST ""))
```

```
                    else (RETURN NIL))
            else (RETURN NIL])

(DO.PROPERTY
  [LAMBDA (WINDOW OBJECT)                    (* ; "Edited  2-Jun-88 10:53 by NCC")

(* ;;; "put up the property sheet for an object")

    (LET [(PROPSHEET (DO.PROPERTY.SETUP OBJECT))
          (HOTSPOT (fetch HOTSPOT of OBJECT))
          (EXTENT (DSPCLIPPINGREGION NIL WINDOW))
          (REGION (WINDOWPROP WINDOW 'REGION]
         (MOVEW PROPSHEET REGION)                (* ; "place over the window")

         [RELMOVEW PROPSHEET (create POSITION
                                     XCOORD _ (IDIFFERENCE
                                                (fetch XCOORD of HOTSPOT)
                                                (fetch LEFT of EXTENT))
                                     YCOORD _ (IDIFFERENCE
                                                (fetch YCOORD of HOTSPOT)
                                                (fetch BOTTOM of EXTENT]
         (RELMOVEW PROPSHEET '(16 . 16))
         (WINDOWPROP PROPSHEET 'CLOSEFN (LIST (FUNCTION FM.ENDEDIT)
                                              (FUNCTION DO.PROPERTY.CLOSE)))
         (WINDOWPROP PROPSHEET 'OBJECT OBJECT)
         (WINDOWPROP PROPSHEET 'WINDOW WINDOW)
         (OPENW PROPSHEET)
         (FM.EDITITEM (CAR (FM.GETSTATE PROPSHEET))
                PROPSHEET])

(DO.PROPERTY.CLOSE
  [LAMBDA (PARM1 PARM2)                      (* ; "Edited  1-Jun-88 19:07 by NCC")

(* ;;; "update node/arc properties when window closes")

    (LET* [(PROPSHEET (OR (WINDOWP PARM1)  PARM2))
           (STATE (FM.GETSTATE PROPSHEET))
           (OBJECT (WINDOWPROP PROPSHEET 'OBJECT))
           [OBJTYPE (fetch OBJTYPE of (WINDOWPROP PROPSHEET 'OBJECT]
           (OLDBOUNDBOX (fetch BOUNDBOX of OBJECT))
           (WINDOW (WINDOWPROP PROPSHEET 'WINDOW]
          (SELECTQ OBJTYPE
              (NODE (replace NODENAME of OBJECT with
                          (LISTGET STATE 'NODENAME))
                    (replace IFSTART of OBJECT with (LISTGET STATE 'IFSTART))
                    (replace IFEND of OBJECT with (LISTGET STATE 'IFEND))
                    (if (LISTGET STATE 'IFALIVE)
                        then (LET ((ARCLIST (for ARC in
                                               (WINDOWPROP WINDOW 'OBJECTLIST)
                                               when (ARCP ARC) collect ARC)))
                                  (BIRTH WINDOW OBJECT ARCLIST)))
                    (replace IFALIVE of OBJECT with (LISTGET STATE 'IFALIVE))
                    (CALC.NODE.BOUND WINDOW OBJECT))
              (ARC (replace ARCSTRING of OBJECT with (LISTGET STATE 'ARCNAME))
                   (CALC.ARC.STRINGBOX OBJECT)
                   (CALC.ARC.DYNAMIC WINDOW OBJECT))
              NIL)
          (WINDOWDELPROP PROPSHEET 'CLOSEFN (FUNCTION DO.PROPERTY.CLOSE))
```

```
          (CLOSEW PROPSHEET)
          (REPAINT.MAIN.WINDOW WINDOW (UNIONREGIONS OLDBOUNDBOX
                                              (fetch BOUNDBOX of OBJECT])

(DO.PROPERTY.SETUP
  [LAMBDA (OBJECT)                                  (* ; "Edited  1-Jun-88 19:20 by NCC")

(* ;;; "setup property menu for type of object")

    (LET (MENU)
         (SELECTQ (fetch OBJTYPE of OBJECT)
             (NODE (if (NOT (BOUNDP 'NODEMENU))
                       then (SETCURSOR WAITINGCURSOR)
                            (PROMPTPRINT "Creating menu")
                            [SETQ NODEMENU
                             (FREEMENU '(((PROPS FORMAT ROW)
                                          (LABEL "Node Property Sheet"
                                                 TYPE DISPLAY FONT
                                                 (MODERN 12 BOLD)
                                                 HJUSTIFY CENTER))
                                         ((PROPS FORMAT ROW)
                                          (LABEL "Node name:" TYPE DISPLAY)
                                          (LABEL "          " TYPE EDIT ID
                                                 NODENAME))
                                         ((PROPS FORMAT ROW)
                                          (LABEL CLOSE TYPE MOMENTARY
                                                 SELECTEDFN DO.PROPERTY.CLOSE
                                                 HJUSTIFY CENTER BOX 2
                                                 BOXSPACE 4))
                                         ((PROPS FORMAT COLUMN COLUMNSPACE 20)
                                          (LABEL "Start?" TYPE TOGGLE ID
                                                 IFSTART BOX 1)
                                          (LABEL "End?" TYPE TOGGLE ID IFEND
                                                 BOX 1)
                                          (LABEL "Alive?" TYPE TOGGLE ID
                                                 IFALIVE BOX 1]
                            (SETCURSOR T)
                            (CLRPROMPT))
                    (SETQ MENU (COPY NODEMENU))

                    (CLOSEW MENU)             (* ; "hack to cut screen flicker")

                    (FM.RESETMENU MENU)
                    (FM.CHANGESTATE 'NODENAME (fetch NODENAME of OBJECT) MENU)
                    (FM.CHANGESTATE 'IFSTART (fetch IFSTART of OBJECT) MENU)
                    (FM.CHANGESTATE 'IFEND (fetch IFEND of OBJECT) MENU)
                    (FM.CHANGESTATE 'IFALIVE (fetch IFALIVE of OBJECT) MENU))
             (ARC (if (NOT (BOUNDP 'ARCMENU))
                      then (SETCURSOR WAITINGCURSOR)
                           (PROMPTPRINT "Creating menu")
                           [SETQ ARCMENU
                            (FREEMENU '(((PROPS FORMAT ROW)
                                         (LABEL "Arc Property Sheet" TYPE
                                                DISPLAY FONT
                                                (MODERN 12 BOLD)
                                                HJUSTIFY CENTER))
                                        ((PROPS FORMAT ROW)
                                         (LABEL "Arc Name:" TYPE DISPLAY)
```

```
                                        (LABEL "          " TYPE EDIT ID
                                               ARCNAME))
                                       ((PROPS FORMAT ROW)
                                        (LABEL CLOSE SELECTEDFN
                                               DO.PROPERTY.CLOSE TYPE
                                               MOMENTARY HJUSTIFY CENTER
                                               BOX 2 BOXSPACE 4]
                              (SETCURSOR T)
                              (CLRPROMPT))
                        (SETQ MENU (COPY ARCMENU))
                        (CLOSEW MENU)                    (* ; "hack to cut flicker")

                        (FM.RESETMENU MENU)
                        (FM.CHANGESTATE 'ARCNAME (fetch ARCSTRING of OBJECT)
                               MENU))
                NIL)
          MENU])

(DO.SELECT
  [LAMBDA (WINDOW X Y)                          (* ; "Edited 13-May-88 13:38 by NCC")

(* ;;; "general object select routine")

    (LET* ((OBJECTLIST (WINDOWPROP WINDOW 'OBJECTLIST))
           (SELECT/OLD (WINDOWPROP WINDOW 'SELECTLIST))
           SELECT/NOW SELECT/NEW)
          (PAINT.SELECT WINDOW)
          (SETQ SELECT/NOW (for OBJECT in OBJECTLIST
                              when (INSIDEP (fetch BOUNDBOX of OBJECT) X Y)
                              collect OBJECT))
          [SETQ SELECT/NOW (SORT SELECT/NOW (FUNCTION (LAMBDA (A B)
                              (LEQ (DISTANCE X Y (fetch HOTSPOT of A))
                                   (DISTANCE X Y (fetch HOTSPOT of B]
          [LET ((OLD/OBJ (CAR SELECT/OLD))
                (NOW/OBJ (CAR SELECT/NOW)))
               (if NOW/OBJ
                   then (if (AND OLD/OBJ (EQ (fetch OBJTYPE of OLD/OBJ)
                                             'NODE)
                                 (EQ (fetch OBJTYPE of NOW/OBJ)
                                     'NODE))
                            then (SETQ SELECT/NEW (LIST NOW/OBJ OLD/OBJ))
                          else (SETQ SELECT/NEW (LIST NOW/OBJ]
          (WINDOWPROP WINDOW 'SELECTLIST SELECT/NEW)
          (PAINT.SELECT WINDOW])

(DO.SNAP.GRID
  [LAMBDA (POSITION)                            (* ; "Edited  9-May-88 17:34 by NCC")

(* ;;; "snap an x y position to a grid")

    (LET* ((HALF (IQUOTIENT GRID 2))
           (X (fetch XCOORD of POSITION))
           (Y (fetch YCOORD of POSITION))
           (XREM (IREMAINDER X GRID))
           (YREM (IREMAINDER Y GRID))
           (XLOW (IDIFFERENCE X XREM))
           (YLOW (IDIFFERENCE Y YREM)))
          (if (IGREATERP GRID 0)
```

```
            then (SETQ X (if (IGREATERP XREM HALF)
                             then (IPLUS XLOW GRID)
                           else XLOW))
                 (SETQ Y (if (GREATERP YREM HALF)
                             then (IPLUS YLOW GRID)
                           else YLOW)))
          (CREATEPOSITION X Y])

(EXISTING.ARC
  [LAMBDA (WINDOW SRCNODE DSTNODE)        (* ; "Edited 13-May-88 13:39 by NCC")

(* ;;; "predicate for existing arc from src to dst")

    (PROG* ((OBJECTLIST (WINDOWPROP WINDOW 'OBJECTLIST))
            (RESULT NIL))
           (for ARC in OBJECTLIST when (ARCP ARC)
              do (if (AND (EQ SRCNODE (fetch ARCSRC of ARC))
                          (EQ DSTNODE (fetch ARCDST of ARC)))
                     then (SETQ RESULT ARC)))
           (RETURN RESULT])

(INIT.DRAW
  [LAMBDA (WINDOW)                        (* ; "Edited 13-May-88 13:39 by NCC")

(* ;;; "setup initial object draw records")

    (WINDOWPROP WINDOW 'NODEPROCS (create DRAWPROCS
                                          DRAWPROC _ (FUNCTION PAINT.NODE)
                                          SELECTPROC _ (FUNCTION
                                                         SELECT.OBJECT)))
    (WINDOWPROP WINDOW 'ARCPROCS (create DRAWPROCS
                                         DRAWPROC _ (FUNCTION PAINT.ARC)
                                         SELECTPROC _ (FUNCTION SELECT.OBJECT])

(INIT.MENU
  [LAMBDA (WINDOW)                        (* ; "Edited 13-May-88 13:40 by NCC")

(* ;;; "function to setup menu bars")

    (LET* [(MENU (INIT.MENU.TEXT WINDOW))
           (OLDMENU (WINDOWPROP WINDOW 'MENUWINDOW))
           (MENUWINDOW (if OLDMENU
                           then OLDMENU
                         else (ADDMENU MENU NIL NIL T)))
           (REGION (WINDOWPROP MENUWINDOW 'REGION))
           (SIZE (CONS (fetch WIDTH of REGION)
                       (fetch HEIGHT of REGION]
          (WINDOWPROP MENUWINDOW 'MINSIZE SIZE)
          (WINDOWPROP MENUWINDOW 'MAXSIZE SIZE)
          (REMOVEWINDOW MENUWINDOW)
          (ATTACHWINDOW MENUWINDOW WINDOW 'RIGHT 'TOP)
          (WINDOWPROP WINDOW 'MENUWINDOW MENUWINDOW])

(INIT.MENU.TEXT
  [LAMBDA (WINDOW)                        (* ; "Edited 13-May-88 13:40 by NCC")

(* ;;; "function to setup menu bars")
```

```
    (create MENU
           ITEMS _ `((NewNode (DO.MENU.ITEM ,WINDOW 'NewNode)
                              "Create node")
                     (NewArc (DO.MENU.ITEM ,WINDOW 'NewArc)
                              "Create new arc")
                     (MoveNode (DO.MENU.ITEM ,WINDOW 'MoveNode)
                              "Move node")
                     (Delete (DO.MENU.ITEM ,WINDOW 'Delete)
                              "Delete selected item")
                     (Properties (DO.MENU.ITEM ,WINDOW 'Properties)
                              "Update property sheet")
                     (Redisplay (DO.MENU.ITEM ,WINDOW 'Redisplay)
                              "Redraw window")
                     (Convert (DO.MENU.ITEM ,WINDOW 'Convert)
                              "Convert to regular expression")
                     (Load (DO.MENU.ITEM ,WINDOW 'Load)
                           "Load a design")
                     (Save (DO.MENU.ITEM ,WINDOW 'Save)
                           "Save a design")
                     (Clear (DO.MENU.ITEM ,WINDOW 'Clear)
                              "Clear window")
                     (Init (DO.MENU.ITEM ,WINDOW 'Init)
                           "Init simulation")
                     (Step (DO.MENU.ITEM ,WINDOW 'Step)
                           "Step simulator")
                     (String (DO.MENU.ITEM ,WINDOW 'String)
                              "Change parse string"))
           CENTERFLG _ T])

(LINE.OFFSET
  [LAMBDA (SRCPOS DSTPOS OFFSET)          (* ; "Edited 31-May-88 11:25 by NCC")

(* ;;; "return a position on line from srcpos to dstpos offset from
dstpos by offset points")

    (PROG* [(SRCX (fetch XCOORD of SRCPOS))
            (SRCY (fetch YCOORD of SRCPOS))
            (DSTX (fetch XCOORD of DSTPOS))
            (DSTY (fetch YCOORD of DSTPOS))
            (ROLL (IDIFFERENCE SRCX DSTX))
            (RISE (IDIFFERENCE SRCY DSTY))
            [SCALE (QUOTIENT OFFSET (SQRT (IPLUS (ITIMES ROLL ROLL)
                                                 (ITIMES RISE RISE]
            (CX (FIX (PLUS (TIMES ROLL SCALE) DSTX 0.5)))
            (CY (FIX (PLUS (TIMES RISE SCALE) DSTY 0.5]
           (RETURN (CREATEPOSITION CX CY])

(PAINT.ARC
  [LAMBDA (WINDOW ARC)                    (* ; "Edited 13-May-88 13:40 by NCC")

(* ;;; "paint arc from one node to another")

    (LET* ((ARCSTRING (fetch ARCSTRING of ARC))
           (STRINGBOX (fetch STRINGBOX of ARC))
           (KNOTS (fetch KNOTS of ARC))
           (ARROW (fetch ARROW of ARC))
           (MODE 'REPLACE)
           (WIDTH 2)
```

```
            (HOME (CAR KNOTS))
            (REST (CDR KNOTS)))
           (MOVETO (fetch XCOORD of HOME)
                  (fetch YCOORD of HOME) WINDOW)
           (for KNOT in REST do (DRAWTO (fetch XCOORD of KNOT)
                                        (fetch YCOORD of KNOT)
                                        WIDTH MODE WINDOW))
           (DRAWBETWEEN (CAR ARROW)
                  (CADR ARROW) WIDTH MODE WINDOW)
           (DRAWBETWEEN (CAR ARROW)
                  (CADDR ARROW) WIDTH MODE WINDOW)
           (MOVETO (fetch LEFT of STRINGBOX)
                  (fetch BOTTOM of STRINGBOX) WINDOW)
           (PRIN1 ARCSTRING WINDOW])

(PAINT.NODE
  [LAMBDA (WINDOW NODE)                    (* ; "Edited 31-May-88 11:19 by NCC")

(* ;;; "paint in a node")

    (LET* ((BOUNDBOX (fetch BOUNDBOX of NODE))
           (NODENAME (fetch NODENAME of NODE))
           (HOTSPOT (fetch HOTSPOT of NODE))
           (XLOC (fetch XCOORD of HOTSPOT))
           (YLOC (fetch YCOORD of HOTSPOT)))
          (DRAWCIRCLE XLOC YLOC 8 2 NIL WINDOW)
          (if (fetch IFSTART of NODE)
              then (DRAWCIRCLE XLOC YLOC 2 1 NIL WINDOW))
          (if (fetch IFEND of NODE)
              then (DRAWCIRCLE XLOC YLOC 4 1 NIL WINDOW))
          (if (fetch IFALIVE of NODE)
              then (DSPFILL BOUNDBOX GRAYSHADE 'INVERT WINDOW))
          (MOVETOUPPERLEFT WINDOW BOUNDBOX)
          (DSPXPOSITION (IDIFFERENCE XLOC (IQUOTIENT
                                           (STRINGWIDTH NODENAME) 2))
                 WINDOW)
          (PRIN1 NODENAME WINDOW])

(PAINT.SELECT
  [LAMBDA (WINDOW)                         (* ; "Edited 13-May-88 13:41 by NCC")

(* ;;; "paint select marks around objects,")

    (LET* [(OBJECTLIST (WINDOWPROP WINDOW 'OBJECTLIST))
           (SELECTLIST (WINDOWPROP WINDOW 'SELECTLIST]
          (for OBJECT in OBJECTLIST when (MEMBER OBJECT SELECTLIST)
             do (APPLY (fetch SELECTPROC of (fetch DRAWPROCS of OBJECT))
                       (LIST WINDOW OBJECT])

(SELECT.OBJECT
  [LAMBDA (WINDOW OBJECT)                  (* ; "Edited 31-May-88 11:18 by NCC")

(* ;;; "draw highlights around any object")

    (LET* ((SIZE 8)
           (BOUNDBOX (fetch BOUNDBOX of OBJECT))
           (LEFT (fetch LEFT of BOUNDBOX))
           (BOTTOM (fetch BOTTOM of BOUNDBOX))
```

```
            (WIDTH (fetch WIDTH of BOUNDBOX))
            (HEIGHT (fetch HEIGHT of BOUNDBOX))
            (RIGHT (IDIFFERENCE (IPLUS LEFT WIDTH) SIZE))
            (TOP (IDIFFERENCE (IPLUS BOTTOM HEIGHT) SIZE))
            (MODE 'INVERT))
           (BLTSHADE BLACKSHADE WINDOW LEFT BOTTOM SIZE SIZE MODE)
           (BLTSHADE BLACKSHADE WINDOW RIGHT BOTTOM SIZE SIZE MODE)
           (BLTSHADE BLACKSHADE WINDOW LEFT TOP SIZE SIZE MODE)
           (BLTSHADE BLACKSHADE WINDOW RIGHT TOP SIZE SIZE MODE])

(UNBEND.ARC
  [LAMBDA (ARC)                                (* ; "Edited 13-May-88 13:42 by NCC")

(* ;;; "unbend an existing line into two knots")

    (LET* ((KNOTS (fetch KNOTS of ARC))
           (ONE (CAR KNOTS))
           (TWO (CAR (LAST KNOTS)))
           (X1 (fetch XCOORD of ONE))
           (X2 (fetch XCOORD of TWO))
           (Y1 (fetch YCOORD of ONE))
           (Y2 (fetch YCOORD of TWO))
           (NEWKNOTS (LIST ONE TWO))
           (NEWHOT (CALC.ARC.HOTSPOT NEWKNOTS)))
          (replace KNOTS of ARC with NEWKNOTS)
          (replace HOTSPOT of ARC with NEWHOT)
          (CALC.ARC.ARROW ARC)
          (CALC.ARC.STRINGBOX ARC]))
(DECLARE%: EVAL@COMPILE

(RECORD ARCOBJ (OBJTYPE BOUNDBOX DRAWPROCS ARCSTRING STRINGBOX HOTSPOT ARCSRC
ARCDST KNOTS ARROW))

(RECORD DRAWPROCS (DRAWPROC SELECTPROC))

(RECORD NODEOBJ (OBJTYPE BOUNDBOX DRAWPROCS NODENAME STRINGBOX HOTSPOT IFSTART
IFEND IFALIVE)))

(PUTPROPS DRAW COPYRIGHT ("Norman C. Crowfoot" 1987 1988))

(RPAQQ GRID 32)
```

C.2 FSA Source Listing

```
(DEFINE-FILE-INFO ^^READTABLE "INTERLISP" ^^PACKAGE "INTERLISP")
(FILECREATED " 2-Jun-88 10:59:51" {DSK}<LISPFILES>FSA.;1 17584


      changes to%:  (FNS DO.SIMULATE.STEP NUMBER-MATCH ARCP REGULAR.MATCH
                         ALIVELIST ALIVEP BIRTH DEADP DO.CONVERT DO.FORMAT
                         DO.GETARCS DO.GETEXIT DO.GETPATH DO.GETSTR DO.NODUP
                         DO.PREFIX DO.SIMULATE.CLEAR DO.SIMULATE.INIT
                         DO.SIMULATE.STRING.DRAW DO.SIMULATE.STRING ENDP
                         GETCHAR KILLALL MIN-MATCH NODEP PARSE.STEP SETALIVE
                         SETDEAD STARTP)

      previous date%: " 1-Jun-88 20:16:35" {DSK}<LISPFILES>FSA.;4)


(* "Copyright (c) 1987, 1988 by Norman C. Crowfoot.  All rights reserved.")

(PRETTYCOMPRINT FSACOMS)

(RPAQQ FSACOMS
       ((VARS)
        (FNS ALIVELIST ALIVEP ARCP BIRTH DEADP DO.CONVERT DO.FORMAT DO.GETARCS
             DO.GETEXIT DO.GETPATH DO.GETSTR DO.NODUP DO.PREFIX
             DO.SIMULATE.CLEAR DO.SIMULATE.INIT DO.SIMULATE.STEP
             DO.SIMULATE.STRING.DRAW DO.SIMULATE.STRING ENDP GETCHAR KILLALL
             MIN-MATCH NODEP NUMBER-MATCH PARSE.STEP REGULAR.MATCH SETALIVE
             SETDEAD STARTP)))
(DEFINEQ

(ALIVELIST
  [LAMBDA (NODELIST)                                  (* ; "Edited 13-May-88 13:42 by NCC")

(* ;;; "return list of live nodes")

    (for NODE in NODELIST when (ALIVEP NODE) collect NODE])

(ALIVEP
  [LAMBDA (NODE)                                      (* ; "Edited 13-May-88 13:42 by NCC")

(* ;;; "predicate function for liveliness")

    (if (NODEP NODE)
        then (fetch IFALIVE of NODE])

(ARCP
  [LAMBDA (ARC)                                       (* ; "Edited 24-May-88 09:04 by NCC")

(* ;;; "predicate function for node")

    (EQ 'ARC (fetch OBJTYPE of ARC])

(BIRTH
  [LAMBDA (WINDOW NODE ARCS)                          (* ; "Edited 13-May-88 13:43 by NCC")

(* ;;; "add a new node to the alive list")
```

```
    (if (DEADP NODE)
        then (SETALIVE WINDOW NODE)
             (for ARC in ARCS when (AND (EQ NODE (fetch ARCSRC of ARC))
                                        (EQUAL "" (fetch ARCSTRING of ARC)))
                do (BIRTH WINDOW (fetch ARCDST of ARC) ARCS])

(DEADP
  [LAMBDA (NODE)                          (* ; "Edited 13-May-88 13:43 by NCC")

(* ;;; "predicate function for health of a node")

    (NOT (ALIVEP NODE])

(DO.CONVERT
  [LAMBDA (WINDOW)                        (* ; "Edited 13-May-88 13:44 by NCC")

(* ;;; "convert graph to regular expression")

    (LET* ((OBJS (WINDOWPROP WINDOW 'OBJECTLIST))
           (NODES (for X in OBJS when (NODEP X) collect X))
           (ARCS (for X in OBJS when (ARCP X) collect X))
           [RESULT (DO.FORMAT (for X in NODES when (STARTP X)
                                 collect (DO.GETEXIT X NIL NODES ARCS]
           FILEID OLDFILE OLDPRETTYFLG)
          (PROMPTPRINT RESULT)
          (if (SETQ FILEID (PROMPTINWINDOW
                             "File to contain regular expression?"))
              then (SETQ OLDFILE (OUTFILE FILEID))
                   (SETQ OLDPRETTYFLG SYSPRETTYFLG)
                   (SETQ SYSPRETTYFLG NIL)
                   (SHOWPRINT RESULT)
                   (SETQ SYSPRETTYFLG OLDPRETTYFLG)
                   (CLOSEF (OUTFILE OLDFILE])

(DO.FORMAT
  [LAMBDA (THINGS)                        (* ; "Edited 13-May-88 13:44 by NCC")

(* ;;; "format a list into regular expressions")

    (LET (FACTOR WORK)
         (SETQ THINGS (DO.NODUP THINGS))
         (if (LISTP THINGS)
             then (SETQ FACTOR (DO.PREFIX THINGS))
                  [if FACTOR
                      then (CONCAT (CAR FACTOR)
                                   (if (CDR FACTOR)
                                       then (DO.FORMAT (CDR FACTOR))
                                     else ""))
                    else (if (EQ (LENGTH THINGS) 1)
                             then (CAR THINGS)
                           else (SETQ WORK (CONCAT "(" (CAR THINGS)))
                                (for STR in (CDR THINGS)
                                   do (SETQ WORK (CONCAT WORK "|" STR)))
                                (SETQ WORK (CONCAT WORK ")"]
           else THINGS])

(DO.GETARCS
```

```
  [LAMBDA (ROOT ARCS)                                    (* ; "Edited 13-May-88 13:44 by NCC")

(* ;;; "collect arcs leaving this root node")

    (for ARC in ARCS when (EQ ROOT (fetch ARCSRC of ARC)) collect ARC])

(DO.GETEXIT
  [LAMBDA (ROOT AVOID ENDS ARCS)                         (* ; "Edited  1-Jun-88 19:44 by NCC")

(* ;;; "find a path from root to an ending node")

    (LET (EXITS LOOPS ANS NEXT RESULT STRING (SKIP (CONS ROOT AVOID)))
         [if (NOT (ENDP ROOT))
             then (for ARC in (DO.GETARCS ROOT ARCS)
                         do (SETQ NEXT (fetch ARCDST of ARC))
                            (SETQ STRING (DO.GETSTR ARC))
                            (if (ENDP NEXT)
                                then (SETQ EXITS (CONS STRING EXITS)))
                            (if (AND (NOT (MEMBER NEXT SKIP))
                                     (SETQ RESULT (DO.GETEXIT NEXT SKIP
                                                              ENDS ARCS)))
                                then (SETQ EXITS (CONS (CONCAT STRING
                                                               RESULT)
                                                       EXITS]
         (SETQ EXITS (REVERSE EXITS))
         (SETQ LOOPS (DO.GETPATH ROOT ROOT AVOID ENDS ARCS))
         (if LOOPS
             then (SETQ ANS (CONCAT (DO.FORMAT LOOPS) "*")))
         [if EXITS
             then (SETQ ANS (if ANS
                                then (CONCAT ANS (DO.FORMAT EXITS))
                              else (DO.FORMAT EXITS]
         ANS])

(DO.GETPATH
  [LAMBDA (SRC DST AVOID ENDS ARCS)                      (* ; "Edited 13-May-88 13:45 by NCC")

(* ;;; "get all paths leading to a specific node")

    (LET (PATHS ANS NEXT RESULT STRING (SKIP (CONS SRC AVOID)))
         [for ARC in (DO.GETARCS SRC ARCS)
            do (SETQ NEXT (fetch ARCDST of ARC))
               (SETQ STRING (DO.GETSTR ARC))
               (if (EQ DST NEXT)
                   then (SETQ PATHS (CONS STRING PATHS))
                 else (if (AND (NOT (MEMBER NEXT SKIP))
                               (SETQ RESULT (DO.GETPATH NEXT DST
                                                        SKIP ENDS ARCS)))
                          then (SETQ PATHS (CONS (CONCAT STRING RESULT)
                                                 PATHS]
         (SETQ PATHS (REVERSE PATHS))
         (if PATHS
             then (SETQ ANS (DO.FORMAT PATHS)))
         ANS])

(DO.GETSTR
  [LAMBDA (ARC)                                          (* ; "Edited 13-May-88 13:45 by NCC")
```

```
(* ;;; "return the string associated with an arc")

    (LET ((STRING (fetch ARCSTRING of ARC)))
         (if (EQ 1 (NCHARS STRING))
             then STRING
           elseif (IGREATERP (NCHARS STRING) 1)
             then (CONCAT "[" STRING "]")
           else STRING])

(DO.NODUP
  [LAMBDA (L)                                    (* ; "Edited 13-May-88 13:46 by NCC")

(* ;;; "remove all duplicates from list")

    (if (LISTP L)
        then (for X on L when (NOT (MEMBER (CAR X) (CDR X))) collect (CAR X))
      else L])

(DO.PREFIX
  [LAMBDA (L)                                    (* ; "Edited 13-May-88 13:46 by NCC")

(* ;;; "return the prefix of a set of lists, cons'ed with the adjusted lists")

    (LET (SIZE)
         (if (ATOM L)
             then NIL
           else (SETQ SIZE (MIN-MATCH L))
                (if (IGREATERP SIZE 0)
                    then (CONS (SUBSTRING (CAR L) 1 SIZE)
                               (for X in L when (ILESSP SIZE (NCHARS X))
                                  collect (SUBSTRING X (ADD1 SIZE])

(DO.SIMULATE.CLEAR
  [LAMBDA (WINDOW)                               (* ; "Edited 13-May-88 13:46 by NCC")

(* ;;; "initialize the simulator back to the start state")

    (LET* [(OBJECTS (WINDOWPROP WINDOW 'OBJECTLIST]
          (for OBJECT in OBJECTS when (NODEP OBJECT) do
               (SETDEAD WINDOW OBJECT))
          (REMOVEWINDOW (WINDOWPROP WINDOW 'STRINGWINDOW NIL])

(DO.SIMULATE.INIT
  [LAMBDA (WINDOW)                               (* ; "Edited 13-May-88 13:46 by NCC")

(* ;;; "initialize the simulator back to the start state")

    (LET* ((OBJECTS (WINDOWPROP WINDOW 'OBJECTLIST))
           (NODES (for OBJECT in OBJECTS when (NODEP OBJECT) collect OBJECT))
           (ARCS (for OBJECT in OBJECTS when (ARCP OBJECT) collect OBJECT)))
          (for NODE in NODES do (SETDEAD WINDOW NODE))
          (for NODE in NODES when (fetch IFSTART of NODE)
             do (BIRTH WINDOW NODE ARCS))
          (WINDOWPROP WINDOW 'STRINGPOINTER 0)
          (DO.SIMULATE.STRING WINDOW])

(DO.SIMULATE.STEP
  [LAMBDA (WINDOW)                               (* ; "Edited 31-May-88 11:25 by NCC")
```

```
(* ;;; "step the simulator one step forward")

    (LET* ((STRING (WINDOWPROP WINDOW 'STRING))
           (STRINGSIZE (NCHARS STRING))
           (STRINGPOINTER (WINDOWPROP WINDOW 'STRINGPOINTER))
           (STRINGPOINTER (MIN (IPLUS STRINGPOINTER 1) STRINGSIZE))
           (OBJECTS (WINDOWPROP WINDOW 'OBJECTLIST))
           (NODES (for OBJECT in OBJECTS when (NODEP OBJECT) collect OBJECT))
           (ARCS (for OBJECT in OBJECTS when (ARCP OBJECT) collect OBJECT)))
          (WINDOWPROP WINDOW 'STRINGPOINTER STRINGPOINTER)
          (DO.SIMULATE.STRING.DRAW WINDOW)
          (if (LEQ STRINGPOINTER STRINGSIZE)
              then (PARSE.STEP WINDOW
                               (SUBSTRING STRING STRINGPOINTER STRINGPOINTER)
                               NODES ARCS))
          (if (EQ STRINGPOINTER STRINGSIZE)
              then (for NODE in NODES when (AND (ALIVEP NODE) (ENDP NODE))
                                          do (RINGBELLS])

(DO.SIMULATE.STRING.DRAW
  [LAMBDA (WINDOW)                          (* ; "Edited 13-May-88 13:47 by NCC")

(* ;;; "draw an accurate representation of the parse so far")

    (LET* [(STRING (WINDOWPROP WINDOW 'STRING))
           (STRINGWINDOW (WINDOWPROP WINDOW 'STRINGWINDOW))
           (STRINGPOINTER (WINDOWPROP WINDOW 'STRINGPOINTER]
          (DSPRESET STRINGWINDOW)
          (MOVETOUPPERLEFT STRINGWINDOW)
          (PRIN1 STRING STRINGWINDOW)
          (TERPRI STRINGWINDOW)
          (for I from 1 to STRINGPOINTER do (PRIN1 " " STRINGWINDOW))
          (PRIN1 "^" STRINGWINDOW])

(DO.SIMULATE.STRING
  [LAMBDA (WINDOW)                          (* ; "Edited 13-May-88 13:47 by NCC")

(* ;;; "update the parse string from the user input")

    (LET* ((STRING (WINDOWPROP WINDOW 'STRING))
           (NEWSTRING (if STRING
                          then STRING
                        else (PROMPTINWINDOW "String to parse")))
           [STRINGREGION (CREATEREGION 0 0
                                       (IPLUS (STRINGWIDTH NEWSTRING) 8)
                                       (IPLUS
                                            (TIMES
                                                 (FONTPROP DEFAULTFONT 'HEIGHT) 2) 8
                                                 (FONTPROP WindowTitleDisplayStream
                                                      'HEIGHT]
           (STRINGWINDOW (CREATEW STRINGREGION "String" NIL T)))
          (WINDOWPROP WINDOW 'STRING NEWSTRING)
          (WINDOWPROP WINDOW 'STRINGPOINTER 0)
          (REMOVEWINDOW (WINDOWPROP WINDOW 'STRINGWINDOW STRINGWINDOW))
          (ATTACHWINDOW STRINGWINDOW WINDOW 'BOTTOM 'JUSTIFY)
          (DO.SIMULATE.STRING.DRAW WINDOW])
```

```
(ENDP
  [LAMBDA (NODE)                                    (* ; "Edited 13-May-88 13:48 by NCC")

(* ;;; "predicate function for end node")

    (if (NODEP NODE)
        then (fetch IFEND of NODE])

(GETCHAR
  [LAMBDA (ARC)                                     (* ; "Edited 13-May-88 13:48 by NCC")

(* ;;; "return a character from the arc's string")

    (fetch ARCSTRING of ARC])

(KILLALL
  [LAMBDA (WINDOW NODES)                            (* ; "Edited 13-May-88 13:48 by NCC")

(* ;;; "kill off all nodes on list")

    (for NODE in NODES do (SETDEAD WINDOW NODE])

(MIN-MATCH
  [LAMBDA (THING)                                   (* ; "Edited 13-May-88 13:49 by NCC")

(* ;;; "return the size of the minimum match of a list of strings")

    (LET* ((HEAD (CAR THING))
           (TAIL (CDR THING))
           (SIZE (NCHARS HEAD)))
          [for X in TAIL do (SETQ SIZE (MIN SIZE (NUMBER-MATCH HEAD X]
          SIZE])

(NODEP
  [LAMBDA (NODE)                                    (* ; "Edited 13-May-88 13:49 by NCC")

(* ;;; "predicate funtion for node")

    (EQ 'NODE (fetch OBJTYPE of NODE])

(NUMBER-MATCH
  [LAMBDA (S1 S2)                                   (* ; "Edited  1-Jun-88 20:15 by NCC")

(* ;;; "return the number of matching characters between strings")

    (LET* [[SIZE (for SIZE from 1 to (MIN (NCHARS S1) (NCHARS S2))
                    count (STREQUAL (SUBSTRING S1 1 SIZE)
                                    (SUBSTRING S2 1 SIZE]
           [LC (for I from 1 to SIZE count (STREQUAL "[" (SUBSTRING S1 I I]
           (RC (for I from 1 to SIZE count (STREQUAL "]" (SUBSTRING S1 I I]
          (if (EQ LC RC)
              then SIZE
            else 0])

(PARSE.STEP
  [LAMBDA (WINDOW CH NODES ARCS)                    (* ; "Edited 13-May-88 13:49 by NCC")

(* ;;; "perform one step in the parse process")
```

```
    (LET [(OLDNODES (ALIVELIST NODES))
          (NEWNODES (for ARC in ARCS
                       when (AND (ALIVEP (fetch ARCSRC of ARC))
                                 (GETCHAR ARC)
                                 (REGULAR.MATCH CH (GETCHAR ARC)))
                       collect (fetch ARCDST of ARC]
         (for NODE in NEWNODES when (NOT (MEMBER NODE OLDNODES))
            do (BIRTH WINDOW NODE ARCS))
         (for NODE in OLDNODES when (NOT (MEMBER NODE NEWNODES))
            do (SETDEAD WINDOW NODE])

(REGULAR.MATCH
  [LAMBDA (CH STRING)                    (* ; "Edited 24-May-88 09:03 by NCC")

(* ;;; "predicate function to match character against regular string
expression")

    (SETQ CH (CHCON1 CH))
    (for I C from 1 to (NCHARS STRING)
       do (SETQ C (NTHCHARCODE STRING I))
          (if (EQ CH C)
              then (RETURN T)
           elseif (AND (EQ C (CHCON1 "-"))
                       (IGREATERP I 1))
              then (if [AND (ILEQ (NTHCHARCODE STRING (SUB1 I)) CH)
                            (ILEQ CH (NTHCHARCODE STRING (ADD1 I]
                       then (RETURN T)
                    else (SETQ I (ADD1 I])

(SETALIVE
  [LAMBDA (WINDOW NODE)                  (* ; "Edited 13-May-88 13:50 by NCC")

(* ;;; "set a node to dead state")

    (if (DEADP NODE)
        then (replace IFALIVE of NODE with T)
             (REPAINT.MAIN.WINDOW WINDOW (fetch BOUNDBOX of NODE])

(SETDEAD
  [LAMBDA (WINDOW NODE)                  (* ; "Edited 13-May-88 13:50 by NCC")

(* ;;; "set a node to dead state")

    (if (ALIVEP NODE)
        then (replace IFALIVE of NODE with NIL)
             (REPAINT.MAIN.WINDOW WINDOW (fetch BOUNDBOX of NODE])

(STARTP
  [LAMBDA (NODE)                         (* ; "Edited 13-May-88 13:50 by NCC")

(* ;;; "predicate function for end node")

    (if (NODEP NODE)
        then (fetch IFSTART of NODE]))
(PUTPROPS FSA COPYRIGHT ("Norman C. Crowfoot" 1987 1988))
```

C.3 INIT Listing

```
(FILECREATED "31-Mar-88 19:01:06" {AUDI:CS:RIT}<NCC6933>LISP>INIT.;41

      previous date: "27-Mar-88 18:44:20" {AUDI:CS:RIT}<NCC6933>LISP>INIT.;40)


(* Copyright (c) 1987, 1988 by Norman C. Crowfoot. All rights reserved.)

(PRETTYCOMPRINT INITCOMS)

(RPAQQ INITCOMS ((VARS (* Norm's very own startup file)
                           CLEANUPOPTIONS DEFAULTPRINTHOST DEditLinger INITIALS
                           USERGREETFILES  DIRECTORIES LISPUSERSDIRECTORIES
                           DEFAULTPRINTINGHOST DEFAULTPRINTERTYPE)
                     (* font vars)
                     (VARS DISPLAYFONTDIRECTORIES DISPLAYFONTEXTENSIONS
                           INTERPRESSFONTDIRECTORIES
                           PRESSFONTWIDTHSFILES)
                     (* bring in the project files)
                     (FILES DRAW.DCOM FREEMENU.DCOM FSA.DCOM GRAPHER.DCOM
                            WINDOW.DCOM)
                     (P (* LOGIN for subsequent access to user's GREET file)
                        (LOGIN NIL (QUOTE QUIET)))
                     (PROP COPYRIGHT INIT)))

(RPAQQ CLEANUPOPTIONS (RC F LIST))

(RPAQQ DEFAULTPRINTHOST "JAGUAR:CS:RIT")

(RPAQQ DEditLinger NIL)

(RPAQQ INITIALS edited:)

(RPAQQ USERGREETFILES (({AUDI:}< USER >LISP>INIT. COM)
                       ({AUDI:}< USER >LISP>INIT)
                       ({AUDI:}< USER >INIT. COM)
                       ({AUDI:}< USER >INIT.LISP)))

(RPAQQ DIRECTORIES ({DSK} {ROLLS:CS:RIT}<KOTO>LISPLIB>
{AUDI:}<KOTO>LISPUSERS>))

(RPAQQ LISPUSERSDIRECTORIES {AUDI:CS:RIT}<KOTO>LISPUSERS>)

(RPAQQ DEFAULTPRINTINGHOST (JAGUAR:CS:RIT))

(RPAQQ DEFAULTPRINTERTYPE PRESS)


(* font vars)

(RPAQQ DISPLAYFONTDIRECTORIES ({DSK} {ROLLS:CS:RIT}<KOTO>FONTS>))

(RPAQQ DISPLAYFONTEXTENSIONS (DISPLAYFONT STRIKE AC))
```

```
(RPAQQ INTERPRESSFONTDIRECTORIES ({DSK} {ROLLS:CS:RIT}<KOTO>FONTS>))

(RPAQQ PRESSFONTWIDTHSFILES ({DSK}FONTS.WIDTHS
{ROLLS:CS:RIT}<KOTO>FONTS>FONTS.WIDTH))



(* bring in the project files)

(FILESLOAD DRAW.DCOM FREEMENU.DCOM FSA.DCOM GRAPHER.DCOM WINDOW.DCOM)
(* LOGIN for subsequent access to user's personal GREET file)
(LOGIN NIL (QUOTE QUIET))

(PUTPROPS INIT COPYRIGHT ("Norman C. Crowfoot" 1987 1988))
(PUTPROPS INIT COPYRIGHT ("Norman C. Crowfoot" 1987 1988))
```

C.4 WINDOW Listing

```
(DEFINE-FILE-INFO ^^READTABLE "INTERLISP" ^^PACKAGE "INTERLISP")
(FILECREATED " 2-Jun-88 10:56:15" {DSK}<LISPFILES>WINDOW.;1 9073

      changes to%:  (FNS CREATE.MAIN.WINDOW BUTTON.LEFT.WINDOW
                         BUTTON.MAIN.WINDOW DO.FILE.OPS DOPROPS PROMPT/YES/NO
                         REPAINT.MAIN.WINDOW RESHAPE.MAIN.WINDOW)

      previous date%: " 1-Jun-88 19:28:47" {DSK}<LISPFILES>WINDOW.;2)


(* "Copyright (c) 1987, 1988 by Norman C. Crowfoot.  All rights reserved.")

(PRETTYCOMPRINT WINDOWCOMS)

(RPAQQ WINDOWCOMS [(FNS BUTTON.LEFT.WINDOW BUTTON.MAIN.WINDOW
                        CREATE.MAIN.WINDOW DO.FILE.OPS DOPROPS PROMPT/YES/NO
                        REPAINT.MAIN.WINDOW RESHAPE.MAIN.WINDOW)
                       (PROP COPYRIGHT WINDOW)
                       (VARS MAIN.MENU)
                       (DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY
                                  COMPILERVARS
                              (ADDVARS (NLAMA)
                                       (NLAML CREATE.MAIN.WINDOW)
                                       (LAMA])
(DEFINEQ

(BUTTON.LEFT.WINDOW
  [LAMBDA (WINDOW X Y)                          (* ; "Edited 13-May-88 13:50 by NCC")

(* ;;; "handle left button event")

    (SELECTQ (WINDOWPROP WINDOW 'MODE)
        (SELECT (DO.SELECT WINDOW X Y)
                (CLRPROMPT))
        (NEWNODE (LET ((NEWNODE (CREATE.NODE WINDOW "" X Y NIL NIL)))
                      (REPAINT.MAIN.WINDOW WINDOW
                                           (fetch BOUNDBOX of NEWNODE)))
                 (CLRPROMPT))
        (PROGN NIL))
    (WINDOWPROP WINDOW 'MODE 'SELECT)
    (CURSOR T])

(BUTTON.MAIN.WINDOW
  [LAMBDA (WINDOW)                              (* ; "Edited 13-May-88 13:51 by NCC")

(* ;;; "handle button down events in window")

    (LET ((X (LASTMOUSEX WINDOW))
          (Y (LASTMOUSEY WINDOW)))
         (if (MOUSESTATE (ONLY LEFT))
             then (BUTTON.LEFT.WINDOW WINDOW X Y])

(CREATE.MAIN.WINDOW
  [NLAMBDA (WINDOWNAME)                         (* ; "Edited 31-May-88 11:26 by NCC")
```

```
(* ;;; "create main window")

    (LET* ((MINWIDTH 100)
           (MINHEIGHT 100)
           (MINSIZE (CONS MINWIDTH MINHEIGHT))
           (REGION (GETREGION MINWIDTH MINHEIGHT))
           (WINDOW (CREATEW REGION WINDOWNAME)))
          (WINDOWPROP WINDOW 'MINSIZE MINSIZE)
          (WINDOWPROP WINDOW 'REPAINTFN (FUNCTION REPAINT.MAIN.WINDOW))
          (WINDOWPROP WINDOW 'RESHAPEFN (FUNCTION RESHAPE.MAIN.WINDOW))
          (WINDOWPROP WINDOW 'SCROLLFN (FUNCTION SCROLLBYREPAINTFN))
          (WINDOWPROP WINDOW 'SCROLLEXTENTUSE 'LIMIT)
          (WINDOWPROP WINDOW 'BUTTONEVENTFN (FUNCTION BUTTON.MAIN.WINDOW))
          (WINDOWPROP WINDOW 'EXTENT (WINDOWPROP WINDOW 'REGION))
          (WINDOWPROP WINDOW 'MODE 'SELECT)
          [WINDOWPROP WINDOW 'PAINTBRUSH '(PAINT 32800 (SQUARE 16]
          (INIT.DRAW WINDOW)
          (INIT.MENU WINDOW)
          (LET* ((REGION (WINDOWPROP WINDOW 'REGION))
                 (XLOC (IQUOTIENT (fetch WIDTH of REGION) 2))
                 (YLOC (IQUOTIENT (fetch HEIGHT of REGION) 2)))
                (CREATE.NODE WINDOW "START" XLOC YLOC T NIL))
          (RESHAPE.MAIN.WINDOW WINDOW)
          (CURSOR T)
          (SET WINDOWNAME WINDOW])

(DO.FILE.OPS
  [LAMBDA (WINDOW OPER)                    (* ; "Edited  1-Jun-88 18:44 by NCC")

(* ;;; "perform load and save operations on a window")

    (LET (FILEID PROPS OLDPRETTYFLG OLDFILE)
         (SETQ FILEID (PROMPTINWINDOW (CONCAT "File to be "
                                              (SELECTQ OPER ('LOAD "loaded")
                                                            ('SAVE  "saved")
                                                            (PROG NIL))
                                              "?")))
         (SELECTQ OPER
             ('LOAD
                  (SETQ OLDFILE (INFILE FILEID))
                  (SETQ PROPS (READ))
                  (for ARC in PROPS when (ARCP ARC)
                     do (for NODE in PROPS
                               when (AND (NODEP NODE)
                                         (EQUAL NODE (fetch ARCSRC of ARC)))
                               do (replace ARCSRC of ARC with NODE))
                        (for NODE in PROPS
                           when (AND (NODEP NODE)
                                     (EQUAL NODE (fetch ARCDST of ARC)))
                           do (replace ARCDST of ARC with NODE)))
                  (WINDOWPROP WINDOW 'OBJECTLIST PROPS)
                  (CLOSEF (INFILE OLDFILE))
                  (RESHAPE.MAIN.WINDOW WINDOW))
             ('SAVE
                  (SETQ OLDFILE (OUTFILE FILEID))
                  (SETQ PROPS (WINDOWPROP WINDOW 'OBJECTLIST))
                  (SETQ OLDPRETTYFLG SYSPRETTYFLG)
                  (SETQ SYSPRETTYFLG NIL)
```

```
                (SHOWPRINT PROPS)
                (SETQ SYSPRETTYFLG OLDPRETTYFLG)
                (CLOSEF (OUTFILE OLDFILE)))
            (PROG NIL])

(DOPROPS
  [LAMBDA (WINDOW)                                    (* ; "Edited 13-May-88 13:52 by NCC")

(* ;;; "put up property sheets for all selected objects")

    (for OBJECT in (WINDOWPROP WINDOW 'SELECTLIST)
       do (if OBJECT
              then (APPLY (fetch SELECTPROC of (fetch DRAWPROCS of
OBJECT))
                          (LIST WINDOW OBJECT))
                   (DO.PROPERTY WINDOW OBJECT)))
    (WINDOWPROP WINDOW 'SELECTLIST NIL])

(PROMPT/YES/NO
  [LAMBDA (PROMPTSTRING)                              (* ; "Edited 13-May-88 13:52 by NCC")

(* ;;; "prompt for simple boolean value")

    (COND
       ((MEMBER (PROMPTINWINDOW PROMPTSTRING)
                (LIST '"YES" '"Y" '"yes" '"y"))
        T)
       (T NIL])

(REPAINT.MAIN.WINDOW
  [LAMBDA (WINDOW CLIPREGION)                         (* ; "Edited  1-Jun-88 18:38 by NCC")

(* ;;; "repaints selected portions of the window")

    (PROG [(OBJECTLIST (WINDOWPROP WINDOW 'OBJECTLIST))
           (SELECTLIST (WINDOWPROP WINDOW 'SELECTLIST]
          (DSPFILL CLIPREGION WHITESHADE 'REPLACE WINDOW)
          [for OBJECT in OBJECTLIST
               when (AND (NODEP OBJECT)
                         (REGIONSINTERSECTP CLIPREGION
                              (fetch BOUNDBOX of OBJECT)))
               do (APPLY (fetch DRAWPROC of (fetch DRAWPROCS of OBJECT))
                         (LIST WINDOW OBJECT))
                  (SETQ CLIPREGION (UNIONREGIONS CLIPREGION
                                         (fetch BOUNDBOX of OBJECT]
          (for OBJECT in OBJECTLIST
               when (AND (ARCP OBJECT)
                         (REGIONSINTERSECTP CLIPREGION
                                            (fetch BOUNDBOX of OBJECT)))
             do (APPLY (fetch DRAWPROC of (fetch DRAWPROCS of OBJECT))
                       (LIST WINDOW OBJECT)))
          (for OBJECT in OBJECTLIST
               when (AND (MEMBER OBJECT SELECTLIST)
                         (REGIONSINTERSECTP CLIPREGION
                                            (fetch BOUNDBOX of OBJECT)))
             do (APPLY (fetch SELECTPROC of (fetch DRAWPROCS of OBJECT))
                       (LIST WINDOW OBJECT])
```

```
(RESHAPE.MAIN.WINDOW
  [LAMBDA (WINDOW)                          (* ; "Edited 13-May-88 13:53 by NCC")

(* ;;; "reshape entire main window after clearing to white")

    (LET* [(CLIP (DSPCLIPPINGREGION NIL WINDOW))
           (OBJECTS (WINDOWPROP WINDOW 'OBJECTLIST]
          (DSPRESET WINDOW)
          (REPAINT.MAIN.WINDOW WINDOW CLIP)
          [for OBJECT in OBJECTS
               do (SETQ CLIP (UNIONREGIONS CLIP (fetch BOUNDBOX of OBJECT]
          (WINDOWPROP WINDOW 'EXTENT CLIP)
          (REPOSITIONATTACHEDWINDOWS WINDOW)
          (DSPLEFTMARGIN (fetch LEFT of CLIP) WINDOW)
          (DSPRIGHTMARGIN (IPLUS (fetch LEFT of CLIP)
                                 (fetch WIDTH of CLIP))
                 WINDOW])
)
```