A MENU-BASED INTERFACE FOR EXPERT SYSTEM RULES

James J. Tyhurst
Kerry J. Glover

Hughes Aircraft Company
P.O. Box 902, E4/M156
El Segundo, CA 90245

Presented at:
Expert Systems 88: Solutions in Manufacturing
Detroit, Michigan
April 12-14, 1988

Abstract

The Syntax-driven Menu-based Interfaces for Lex and Yacc grammars (SMILY) system is a tool for generating menu-based interfaces for context-free grammars. We present the interface which has been developed for the HICLASS™ Software System as a sample output of the SMILY interface generator. This particular interface helps users to write an expert system rule by prompting them with menus that contain all and only the valid options at each point in the creation of the rule. Thus, users are constantly aware of the available options and they are guided in writing a syntactically correct rule. The SMILY system provides important functional improvements over previous menu-based systems for context-free natural language input. SMILY continues to provide menus through all stages of editing a rule and it can handle an arbitrary LALR(1) context-free grammar.

WRITING EXPERT SYSTEM RULES

Someone learning to use an expert system shell must learn to write rules in a very specific format. If the shell offers many different basic data types, then there will be particular syntactic constructions to manipulate those data structures. Therefore, the person writing the rules must learn about the available features and learn how to reference those features through the given syntax. In practice, we have found that this learning process is difficult, especially for non-programmers. Even experienced users tend not to use the full power of the expert system shell, because it is difficult to remember the entire range of grammatical expressions. Instead, users tend to rely on familiar rule constructions, even though they may not be the most efficient for the task at hand.

For example, the HICLASS™ Software System* is an expert system shell [1], which has been used to develop a variety of manufacturing applications [2,3,4,5]. It uses a procedural language in which it is easy to express simple If-Then rules which check variable values and make value assignments. However, there are also several syntactic constructions for accessing frames and lists, both of which are basic data types in the HICLASS system. These constructions are conceptually more difficult, particularly since most first-time users are not familiar with the notion of frames, slots, and instances.

In this paper, we present a syntax-driven menu-based editor for HICLASS rules which addresses the problems of the inexperienced user. It is *syntax-driven*, because at each stage of writing a rule, it offers choices which will lead to a syntactically correct rule. It is *menu-based*, because choices are presented to the user on menus. Typing is therefore minimized, since rules are constructed by using a mouse to make menu selections. Editing commands such as Save, Abort, and Undo are also chosen by pointing and clicking with the mouse.

HELPING THE USER TO WRITE RULES

Many of the difficulties encountered by an inexperienced rule writer are also encountered by someone trying to query a database [6]. An interesting solution for databases is to use a limited fragment of English as a query language such that a menu-based interface can be created [7,8]. In the following sections, we will describe a system which extends this notion of a menu-based interface to the domain of expert system rules. Although this

* HICLASS is a trademark of Hughes Aircraft Company.

system relies on the insights of previous database query systems, the functional capabilities of the system exceed the abilities of NLMenu as presented by Tennant. In particular, the Syntax-driven Menu-based Interfaces for Lex and Yacc grammars (SMILY) system, which is described below, has more powerful editing capabilities. We first present an overview of the SMILY system and then compare it to NLMenu.

PROMPT THE RULE WRITER WITH VALID CHOICES - When inserting text into a rule, SMILY always has a set of active menus which contain valid options for the current position in the rule. For example, Figure 1 shows the valid options for the beginning of a rule. Three menus are active: 'Rule Type', 'Statement', and '{Comment}'. Items are selected from the menu by using a mouse. Suppose the user chooses 'ASSERT' from the 'Rule Type' menu. The text window is updated to show the newly inserted word and the menus are updated to reflect the words which may begin an Assert rule (see Figure 2). Notice that the 'Rule Type' menu is no longer visible, because there are no entries in that menu which would be acceptable at this point. The 'Statement' menu is still active, but it now contains many more options which are valid at this new position in the rule. The rule writer continues making selections until an entire rule has been constructed.

Some people prefer to type, rather than select items from menus. Therefore, SMILY offers both options. At any point during the insertion process, the user may choose to type text manually, rather than making menu selections. When the 'Edit' icon is selected, a new window appears where text may be entered using the computer system's text editor. Once that window is closed, the entire text from that window is inserted at the current insertion position. If this new text contains syntax errors, the system stops before the first word which causes an error and displays the menus of valid options at that point in the rule. If there are no errors in the new text, then the insertion position moves to the end of the inserted text and new selection menus are displayed.

When the user has completed entering the rule, selecting the 'Save' icon causes the rule to be saved in the knowledge base. The save operation includes a validity check. If the current rule is not complete (and therefore is syntactically incorrect), the user is given a warning message and asked to confirm that the bad rule really should be saved.

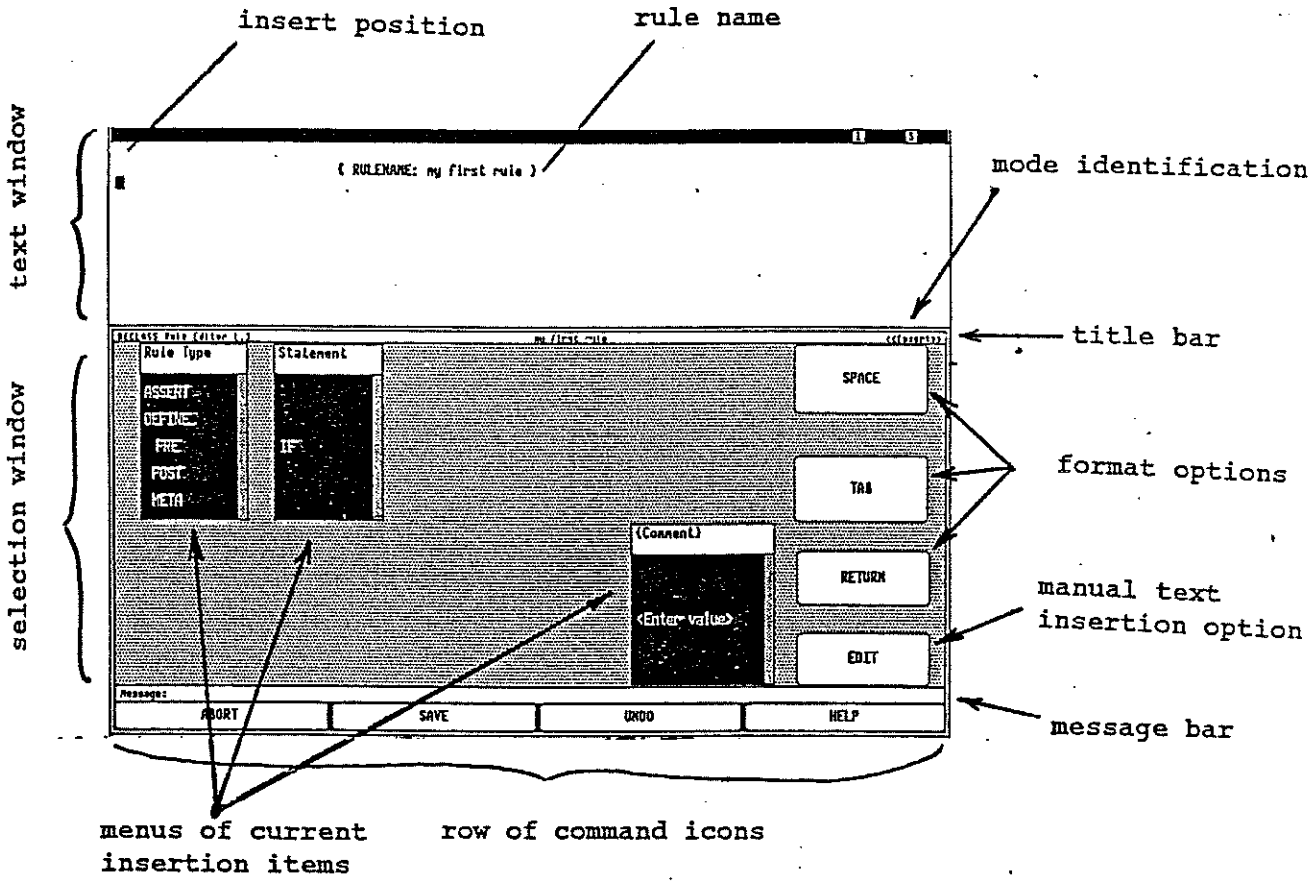

Figure 1. The anatomy of the rule editor window.

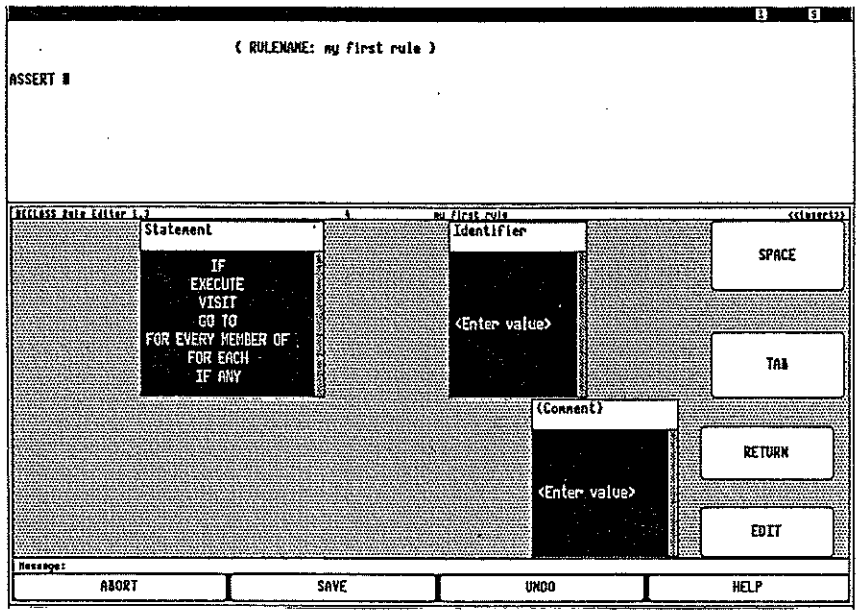

Figure 2. Rule display after inserting 'ASSERT'.

MODIFYING A RULE - Rules may be modified at any stage of processing. It is possible to call up an existing rule for editing. Or, the user can change any part of a rule as the rule is being entered for the first time. Modifications are accomplished through three modes of operation.

Insert Mode - The insertion location is indicated by a cursor in the body of the text. This can be moved simply by pointing to the desired location in the text window and clicking the mouse button. Before SMILY moves the cursor to this new location, it checks the text for syntactic correctness up to the desired location. If an error is found, the cursor is place just before the first word that causes the error. This means that the insertion position can never be moved past a grammatical error. The selection menus are modified to reflect the options available at the new insertion position.

Delete Mode - In this mode, the cursor, which indicates the insertion location, disappears and one or more deletions may be performed. A range of text is selected by pointing to the beginning of the text, depressing the mouse button and dragging the selection pointer to the end of the text. The selected text is shown in reverse video. When the mouse button is released, the selected text is removed from the rule. It is possible to introduce syntactic errors into the rule by performing such a deletion. The text is *not* checked for syntactic validity until the user changes from Delete mode to either Insert or Substitute mode. When the user changes out of Delete mode, the system attempts to place the insertion location at the site of the last deletion. However, if a syntactic error is detected before that position, the cursor will be placed just before the first error encountered.

Substitute Mode - A range of text is selected as for the Delete mode. However, when the mouse button is released and the range of text is deleted, a new window appears with the deleted text. The user may then modify the text using the computer system's text editor. When the user has finished and closes the edit window, the new text is substituted into the rule at the place of the original deletion. Substituted text is checked for errors in the same manner as was described above for manually typed text.

In all three modes of operation (Insert, Delete, and Substitute), an 'Undo' icon is available to undo previous modifications. The Rule Editor currently saves the last 32 changes, where each menu selection or deletion range counts as one change. A substitution counts as multiple changes, one deletion followed by a separate insertion for each lexical item. Movement of the insertion position or change of mode does *not* count as a change with respect to the Undo function.

This concludes our overview of how the system appears to the user. In the following section, we discuss the algorithm for generating menus.

DETERMINING VALID MENU ENTRIES - It is necessary to have a Yacc* generated shift-reduce parser in order to build a SMILY interface. At each state in the parser, particular lexical categories may be identified as causing a shift action. This transition information is extracted from the parser in order to build a look-ahead table for the SMILY interface. For every state of the parser, the look-ahead table contains a list of lexical categories which will lead to a valid shift operation. Unfortunately, this list may not be complete. When shift and reduce actions are both possible in one state**, the complete list of lexical categories leading to a valid shift must be determined dynamically at run-time. Therefore, SMILY uses a run-time "parse-ahead" procedure to supplement the look-ahead table for those cases.

The parser always starts at the beginning of the rule and is fed tokens one at a time by the lexical analyzer. The parser moves through a sequence of states as various shift and reduce operations are performed. When the insertion position is reached, the lexical analyzer switches to a look-ahead mode. It uses the current state of the parser as an index into the look-ahead table to determine which lexical categories will lead to valid transitions. Menus are created containing the words or entry options that correspond to those lexical categories. The interface is *syntax-driven*, because the contents of the menus are determined directly from the current state of the parser.

When the user selects one of the menu items, the lexical analyzer returns that token to the parser. Based on the value of the token, the parser moves to a new state and requests a new token from the lexical analyzer. New menus are calculated for this new state and the process continues until the entire rule has been constructed.

Menu choices can be of two types. A fixed string, such as 'IF', can be selected by the user to be directly inserted into the text. A second type of entry on a menu is a token category name. For example, a menu might

---

* Yacc (Yet Another Compiler-Compiler) accepts a set of augmented context-free grammar rules as input and it generates a shift-reduce parser written in the C programming language. Consult the UNIX Programmer's Manual [9] for more information about Yacc.

** A shift/reduce conflict is a special case of this condition. However, even when no shift/reduce conflict occurs, a dynamic parse ahead is still required if certain tokens would cause a shift, while others cause a reduce.

contain an entry labeled 'Integer', which further prompts the user to enter an integer value. The value typed by the user is verified by the lexical analyzer to make sure that it is a valid sequence of characters for the chosen token category. In the case of 'Integer', typing something other than an integer, such as '3.14', will result in an error message. Otherwise, the value typed by the user is inserted into the text at the insertion location.

A single menu may contain entries for more than one lexical category. For example, there could be a 'Constants' menu containing three category names: 'Integer', 'Real', and 'String'. Choosing any one of these entries will result in a prompt for the user to type an appropriate value for the chosen category.

A SMILY interface may also be designed to include a special type of dynamic menu, which is limited to a single lexical category. For example, the interface for the HICLASS Software System is initialized with an 'Identifier' menu, which has a single entry that is labeled '<Enter Value>'. When '<Enter Value>' is chosen from the menu, another prompt pops up with space for the user to type an identifier. Every time the user enters a valid identifier through this menu, the new identifier is added as an additional choice in the menu. The next time that this menu appears, the user's last identifier shows up at the top of the accumulated choices, which are all listed under the original '<Enter Value>' choice.

To summarize, there are two types of menus. One type is restricted to entries of a single lexical category. However, this type of menu is dynamic and accumulates values which have been entered by the user. Another type of menu allows for entries of different lexical categories, but it cannot add entries dynamically. This type of menu can have two types of menu entries. The first type of entry is a simple fixed string which is inserted directly into the text. The second type of entry is a category name which leads to a pop-up prompt, so that the user can enter a value meeting the restrictions of that category.

ADVANTAGES OF A MENU-BASED INTERFACE

The SMILY menu-based interface offers several advantages to the user as compared to a simple text entry interface. Inexperienced and experienced users both benefit from the presentation of all the available choices for valid rule construction. The inexperienced user is guided through the variety of syntactic constructions and, thereby, is taught the rule grammar. Likewise, the more experienced user is reminded of the full

extent of the rule grammar. This tends to prevent a user from relying on a familiar construction when a more appropriate one is available.*

Proficient rule writers can still benefit from the SMILY editor, even if they type the rules rather than using the menus. When the typed text is inserted into the rule, error detection is immediate and the offending token can easily be compared with the displayed valid choices. This makes the SMILY interface a valuable rule debugger. Error detection and extended editing capabilities combine to provide a useful tool for users of all levels.

An advantage which is not apparent to the user is the fact that the SMILY interface is generated using a parser written with the aid of Yacc and Lex. These tools are provided by most UNIX™ environments for constructing parsers.** Consequently, many applications needing a user input language specify the language using Yacc and Lex. A SMILY interface can by created for each user input language written with these UNIX tools.

THE CURRENT INTERFACE HAS SOME LIMITATIONS

Several limitations exist in the current SMILY interface. First, the graphics routines that manipulate the menus and icons are all internal to a software package that only runs on Apollo workstations. This limits the SMILY editor in its portability. However, the code for managing menus is contained in a single module that can be replaced by a functional equivalent which is less machine-dependent. For example, the module could be rewritten to use the Curses screen handling functions available on UNIX systems.

The context-free grammar rules accepted by Yacc do not allow one to express context-sensitive features in the input language. However, each rule is associated with an action. Therefore, flags and other data structures can be manipulated in the actions in order to achieve some context-sensitivity in the resulting parser. The problem is that SMILY currently ignores the action portion of Yacc rules. This means that it cannot generate context-sensitive menu selections based on the context checking in the action portion of those rules. Therefore, SMILY can only handle grammar specifications which are truly context-free.

---

* It would be extremely useful (and relatively easy) to link Help files to each menu in order to aid the user in choosing between different commands. However, this ability has not been implemented yet in the SMILY system.

** UNIX is a trademark of Bell Telephone Laboratories, Inc.

A final limitation is that a SMILY interface is not fully generated automatically. Yacc and Lex grammar specifications are not uniform across applications, since they may contain user-defined functions or global flags which affect the operation of the resulting parser and lexical analyzer. This means that some manual intervention is currently necessary to prepare a Lex/Yacc grammar specification before a SMILY interface is generated.

Even though the generation process is not fully automated, it is possible to create an entirely new menu-based interface in less than one working day. For example, it took one person about 6 hours to create the interface for the HICLASS Software System. The process of generating the interface included: (1) manual modifications to the Lex and Yacc grammar specifications; (2) manual generation of a menu table that assigns each lexical category to the appropriate menu; (3) programmatic generation of a look-ahead table; and (4) running a shell script to compile and link all of the required modules. Revisions to the interface are accomplished in considerably less time, since it is not necessary to completely redo the initial manual steps.

COMPARING SMILY TO OTHER SYSTEMS

There are many similarities between SMILY and NLMenu (Tennant 1984). However, there are a number of reasons why it was necessary to develop the additional capabilities of SMILY, rather than using NLMenu for expert system rules.

NLMenu is a very effective tool for generating database queries. However, its output consists of single, relatively short commands. Therefore, it does not contain any editing features other than a rubout key. This lack of editing capabilities is unacceptable for handling expert system rules, which may be quite lengthy. Expert system rules are modified frequently during the initial prototype and development phases of an expert system. Therefore, an expert system interface must contain an effective rule editor. SMILY integrates editing capabilities into the menu-based approach. Therefore, it preserves the benefits of menus, while allowing the user to deal with a long sequence of text.

Another advantage to SMILY is the ability to integrate it with existing applications. SMILY is written in C and makes use of Yacc which is a widely available tool on UNIX operating systems. Before developing SMILY, we already had a number of parsers written with the help of Yacc (e.g. parsers for the SQL database query language, engineering notes, HICLASS rules, and a rule-driven graphics generator). SMILY allows us

to use those Yacc grammars directly. This means that a single grammar specification is maintained for both the menu-based interface and the parsers (which are also used in non-interactive processing). On the other hand, if we were to use NLMenu, it would involve a rewrite of existing grammars and it would probably be necessary to maintain the Yacc and NLMenu grammars separately.

The class of grammars accepted by the NLMenu system is not mentioned in the publications that we have seen. SMILY generates interfaces for the same class of grammars accepted by Yacc, which is context-free LALR(1) grammars with disambiguating rules.

SUMMARY

SMILY can be used to generate a menu-based interface for an arbitrary LALR(1) grammar specified in the format required by the UNIX tool Yacc. A look-ahead table is extracted from the transition information in the Yacc generated parser. In the Insertion mode, an interface uses the look-ahead table to produce menus of valid tokens for the current position in the text. Delete and Substitute modes are also available for editing the text.

SMILY interfaces include a manual entry option for those users who prefer to type, rather than use menus. Even when the menus are not used for text entry, a SMILY interface provides effective feedback to the user. It checks the typed input and uses the menus to show the valid options that exist at the location of a syntactic error.

One application of SMILY is a menu-based interface for expert system rules. This interface helps users to learn the rule syntax while guiding them to write syntactically well-formed rules. Thus, the interface satisfies the user's desire to know when particular constructions are appropriate. It also satisfies the expert system shell's requirement that rules be syntactically correct prior to insertion in the knowledge base.

ACKNOWLEDGMENTS

We would like to thank Jim Cheung and Julie Irvine for their guidance and suggestions while we were designing the SMILY system. We also appreciate the constructive criticism that we received from Atul Bajpai and Diane Haig, who commented on the first draft of this paper.

REFERENCES

[1] Lam, Dennis L., and Carolyn R. Estes. Expert systems in manufacturing applications. Ultratech Conference Proceedings (Vol. 1), Long Beach, California (1986). pp. 2-1 to 2-14.

[2] Liu, David. Intelligent manufacturing planning systems. AUTOFACT '85 Conference Proceedings Supplement, MS85-1070 (1985).

[3] Liu, David. Expert systems for process planning. Computer Aided Engineering 4, 65-72 (1985).

[4] Tyhurst, James J. Applying linguistic knowledge to engineering notes. In S. C-Y. Lu and R. Komanduri (eds.), Knowledge-Based Expert Systems for Manufacturing (PED-Vol. 24). The American Society of Mechanical Engineers, New York (1986). pp. 131-136.

[5] Zucherman, Mark I. A knowledge base development for producibility analysis in mechanical design. Ultratech Conference Proceedings (Vol. 1), Long Beach, California (1986). pp. 2-15 to 2-36.

[6] Tennant, Harry R. Menu-based natural language understanding. AFIPS Conference Proceedings, Vol. 53. 1984 National Computer Conference, Las Vegas, Nevada (1984). pp. 629-635.

[7] Tennant, Harry R., Kenneth M. Ross, and Craig W. Thompson. Usable natural language interfaces through menu-based natural language understanding. Proceedings of the Conference on Human Factors in Computing Systems, Cambridge, MA (1983). pp. 154-160.

[8] Thompson, Craig W., Kenneth M. Ross, Harry R. Tennant, and Richard M. Saenz. Building usable menu-based natural language interfaces to databases. Proceedings of the 9th International Conference on Very Large Databases, Florence, Italy (1983). pp. 43-55.

[9] UNIX Programmer's Manual (Seventh Edition). Bell Telephone Laboratories, Incorporated, Murray Hill, N.J. (1979).