The variety of issues covered in this volume (fuzzy reasoning, multi-criteria decision making, propositional formalisms, dynamic game theory pattern recognition, as well as the more traditional expert systems, AI, etc.) make it unsuitable for low-level pedagogic use. It would be appropriate for an advanced seminar covering the frontiers of applications of AI in economic and managerial domains, but its primary use is more likely to be for those conducting research in the field. I infer that the workshop on which this volume is based has been held previously, and I get the sense that similar workshops will be held in the future. Thus, this volume represents a thirty-three facet snapshot of the state-of-the-art at one point in time. It is as such a valuable record.

ARTICLES

ARRAYS AND ASSIGNMENT IN PROLOG

William G. Wong
Logic Fusion Inc.
1333 Moon Drive
Yardley, PA 19067
(215) 736-2463

Conventional **Prolog** implementions support dynamic update of the data base using assert and retract. Changes to the data base allow global information to be shared throughout various parts of a program. The advantage is quick access to information which would otherwise be exchanged using shared **Prolog** variables.

The use of assert and retract is not recommended as a general data manipulation tool because of the overhead of adjusting various data base links. Using normal variables is fine as long as a single value needs to be exchanged. However, there are a number of instances where multiple values need to be exchanged and only the most recent value is of interest. This is essentially the assignable variable found in conventional languages.

The proposal presented here uses the concept of assignable variables and shows how it can be implemented, granted not in the most efficient fashion. The idea is that a single parameter is able to pass the information among various components of a program. The idea is identical to infinite streams in **Concurrent Prolog** except that the stream can only be accessed as a variable or its latest value. The code to implement this in conventional **Prolog** is:

```
new_var ( [ V : _ ], V ).

get_var ( [ V : Tail ], V ) :- var ( Tail ), !.  get_var ( [ _ :
Tail ], V ) :- get_var ( Tail, V ).

set_var ( N, V ) :- var ( N ), !, N = [ V : _ ].  set_var ( [
_ : Tail ], V ) :- set_var ( Tail, V ).
```

A new variable is essentially a list whose last element before the uninstantiated tail of the list is the current value. One functor is provided to create new variables while two others set and access the variables. An efficient implemention would only keep track of the latest value.

Arrays are in the same class as assignable variables. A special functor is necessary to allocate the arrays and access the necessary elements. Arrays and assignable variables are not really as foreign to **Prolog** as some would like them to be. It is really a matter of efficient implementation.

Automatic Menu Generation

Steve Cousins
Center for Intelligent Computer Systems
Department of Computer Science
Washington University
St. Louis, MO 63130
sbc@wucs1.UUCP

This note describes a very simple modification to Prolog's Definite Clause Grammar[1] (DCG) formalism which allows a menu system with the flavor of Texas Instruments' NLMenus[2] to be easily implemented.

Introduction

A language for interfacing with a computer program may be as simple as a command language or as complex as a subset of a natural language. In either case, a grammar for this language captures in a concise way all of the sentences in the language, and is fundamentally knowledge about that language. Compiler writers are well aware of the value of a grammar in interpreting input to a computer. This same information about the language can be used on the other end of the communications channel to assist the user in creating valid sentences in the program's input language. This note describes a technique for building a menu interface in Prolog automatically, given the grammar for a language.

The Next set

To build a menu interface to a language, it is necessary to know what the "next words" possible are at any point in a sentence. This set of "next words" is called the Next set. A menu interface simply displays the Next set at each point in a parse and allows the user to choose one member of the set. Initially, the Next set is the set of words which may occur as the first word of any sentence in the language. When one of these is chosen, the Next set for that one-word prefix is the set of second words in the set of sentences beginning with the chosen first word. In general, after a valid prefix of n words has been chosen, the set of all (n+1)th words that, when appended to the first n words forms a prefix of a valid sentence, is called the Next set, and is placed in a menu.

The method just described is essentially the basic concept behind Texas Instruments' NLMenus. A choice is made from a menu, and then another menu becomes active based on the first choice. This continues until a complete sentence has been chosen.

Having the Next sets at each point in the parse of a sentence is also useful in command interfaces. If a user is typing and the parser has a relatively small set of legal words, it can assist the user as she types by informing her immediately of an error, listing the next legal words if she requests them, or even attempting to correct simple mistakes for her in the spirit of Xerox's DWIM (Do What I Mean)[3]

In his dissertation, Masaru Tomita of CMU pointed out that the above interface advantages can be had in parsers which are strictly left-to-right[4] Fortunately, when Prolog's proof procedure is used as a parser for DCGs, the parsing is, in fact, left to right. The next section describes a modification to the internals of the Definite Clause Grammar mechanism to calculate the Next set in Prolog.

Calculating the Next set in Prolog

The Definite Clause Grammar (DCG) mechanism of Prolog is a translator from a grammar syntax to Prolog predicates. DCGs handle terminals by translating them into calls to a special predicate called **"c"** ("connects"). By carefully redefining the **"c"** operator[5] it is possible to calculate the Next sets. The connects operator, **c**, is defined simply as:

```
c([W|S],W,S).
```

This definition reads: "if the head of the first argument matches the second argument, succeed, and return the tail of the first argument as the third argument." If the match fails, the predicate fails.

Based on Prolog's backtracking, we can redefine **c** to keep track of the Next set, the set of words which can follow a legal prefix such that there exists a suffix that, when concatenated onto the prefix and the element from the next set, the resulting sentence is in the language.

Normally, **c** either succeeds or fails based on whether or not W is the head of the incoming list. This is based on the assumption that the entire input sentence is passed as the first argument. If the first argument is only a prefix of a sentence however, in parsing that string Prolog will at some point fail because legal words in the language are attempted to be matched against empty input. Whenever this happens, W is a word that should be a member of the Next set. W should be added to the Next set, but **c** should fail the test so that other words in the set "farther down the grammar" (since we are depending on Prolog's particular parsing mechanism for DCG's) can be found.

Assuming the predicate **save(W)** saves W to the Next set, the new definition of the **c** operator is as follows:

```
c([],W,[]) :- save(W),fail.
c([W|S],W,S).
```

Notice that the second line of the definition is just the original definition of **c**, and that the first line never succeeds. The new line of the definition is only active in the case described above.

The definition of save can be a simple assert, since facts asserted are not retracted during backtracking. Save is defined as:

```
save(W) :- assert( next(W) ).
```

If the relation **next** is empty before attempting to parse a sentence with a DCG (i.e. **next(X)** would fail), **next** will contain all of the words in the Next set when the parse has completed (and failed).

Using this mechanism

A simple recursive program implements the interface described above. The predicate **menu(L,W)** is assumed to take a list of words to be in the menu L and return the word chosen, W. The predicate **get_sent(Sent)** returns a sentence in some grammar through Sent. We assume the grammar starts with the non-terminal **s**. Recall that in trying to prove **s**, the DCG mechanism will automatically make calls to **c** as terminal symbols are reached. The following program implements **get_sent**:

```
get_sent(Sent) :- get_sent([],Sent).
get_sent(Prefix,Prefix) :-
        no_nexts,
        s(Prefix,[]).
get_sent(Prefix,Sent) :-
        setof(X,next(X),Menu),
        menu(Menu,Word),
        append(Prefix,[Word],NewPrefix),
        get_sent(NewPrefix,Sent).

no_nexts :- retract(next(X)),fail.
no_nexts.
```

Example

A very simple example may help to demonstrate the use of this technique. Consider a DCG for a language with 6 sentences:

```
s --> dog_name, dog_action.
s --> boy_name, boy_action.
boy_name --> [john].
dog_name --> [rover].
boy_action --> [yells].
boy_action --> action.
dog_action --> [barks].
dog_action --> action.
action --> [runs].
action --> [hides].
```

Using the new definition of **c** and the definition of **get_sent**, this grammar produces two menus automatically. The first contains the set [john,rover] and the second contains three actions, depending on the choice from the first menu. The menu predicate used in this example simply displays the set as a list and accepts a member of the list from the keyboard; in actual use of this system, the menu predicate might allow the choice to be made from a mouse, from function keys, etc.

```
| ?- get_sent(X).
Choose one:
[john,rover]
|: john.
Choose one:
[hides,runs,yells]
|: runs.
X = [john,runs]

| ?- get_sent(X).
Choose one:
[john,rover]
|: rover.
Choose one:
[barks,hides,runs]
|: barks.
X = [rover,barks]
```

Conclusions

The Next set is useful in building menu interfaces to programs. It can be calculated in Prolog with a one-line addition to the existing implementation of the Definite Clause Grammar formalism. Although the Next set approach does not solve all problems in the interface area, it demonstrates the usefulness of using input language grammars to automatically generate program interfaces. There are languages in which this technique may prove awkward, such as when some or all of the Next sets are very large. With care, however, the Next set can be useful in quickly implementing powerful interfaces using Prolog's DCG mechanism.

Notes

[1]W.F. Clocksin & C.S. Mellish, Programming in Prolog, Chapter 9, Springer-Verlag, 1981.

[2]Texas Instruments, Explorer Natural Language Menu System User's Guide, 1985.

[3]Xerox, Interlisp-D Reference Manual, Volume 2: Environment, 1985.

[4]Masaru Tomita, "An Efficient Context-free Parsing Algorithm for Natural Languages and its Applications," Computer Science Dept. Report, Carnegie-Mellon University, May 1985.

[5]Since c/3 is a predefined operator in most Prolog implementations, redefining it may not be easy. The example here was done in a version of C-Prolog with the c operator unlocked so that its original definition could be retracted. Thanks to Guillermo Simari for unlocking it for me. If c/3 cannot be unlocked, the Next set could still be calculated by defining a new DCG operator exactly like the old one, but with a different symbol (like ==>) and a different definition of connects. This is easy, since the code for DCGs is in Clocksin & Mellish.

SELECTED AI-RELATED DISSERTATIONS

Assembled by:
Susanne M. Humphrey
National Library of Medicine
Bethesda, MD 20894
and
Bob Krovetz
University of Massachusetts
Amherst, MA 01002

The following are citations selected by title and abstract as being related to AI, resulting from a computer search, using the BRS Information Technologies retrieval service, of the Dissertation Abstracts International (DAI) database produced by University Microfilms International.

The online file includes abstracts, which are not published in this listing, but the citations below do include the DAI reference for finding the abstract in the published DAI. Other elements of the citation are author; university, degree, and, if available, number of pages; title; UM order number and year-month of DAI; and DAI subject category chosen by the author of the dissertation. References are sorted first by the initial DAI subject category and second by the author. In addition the database includes masters abstracts, denoted by MAI (Masters Abstracts International) instead of DAI as the reference to the abstract in the published MAI.

Unless otherwise specified, paper or microform copies of dissertations may be ordered from University Microfilms International, Dissertation Copies, Post Office Box 1764, Ann Arbor, MI 48106; telephone for U.S. (except Michigan, Hawaii, Alaska): 1-800-521-3042, for Canada: 1-800-268-6090. Price lists and other ordering and shipping information are in the introduction to the published DAI.

SCHMOLDT, DANIEL LEE. The University of Wisconsin - Madison Ph.D. 1987, 232 pages. Evaluation of an expert system approach to forest pest management of red pine (Pinus resinosa). DAI V48(02), SecB, pp314. University Microfilms Order Number ADG87-08112. Agriculture, Forestry and Wildlife.

MARCHANT, GARRY ALLEN. The University of Michigan Ph.D. 1987, 184 pages. Analogical reasoning and error detection. DAI V48(02), SecA, pp432. University Microfilms Order Number ADG87-12168. Business Administration, Accounting.

BASU, AMIT. The University of Rochester Ph.D. 1986, 180 pages. Imprecise reasoning in intelligent decision support systems. DAI v47(12), SecA, pp4439. University Microfilms Order Number ADG87-08218. Business Administration, Management.

CHAPMAN, BRUCE LEROY. The University of Texas at Austin Ph.D. 1986, 585 pages. KISMET: a knowledge-based system for the simulation of discrete manufacturing operations. DAI v47(12), SecA, pp4440. University Microfilms Order Number ADG87-05977. Business Administration, Management.

ZACARIAS, PRUDENCE TANGCO. Purdue University Ph.D. 1986, 148 pages. A script-based knowledge representation for intelligent office information systems. DAI V48(01), SecA, pp174. University Microfilms Order Number ADG87-09879. Business Administration, Management.

ALOIMONOS, JOHN. The University of Rochester Ph.D. 1987, 261 pages. Computing intrinsic images. DAI V48(01), SecB, pp183. University Microfilms Order Number ADG87-09482. Computer Science.

CHEN, JENN-NAN. Northwestern University Ph.D. 1987, 137 pages. Verification and translation of distributed computing system software design. DAI V48(01), SecB, pp184. University Microfilms Order Number ADG87-10326. Computer Science.

CHRISTENSEN, MARGARET H. Temple University Ph.D. 1987, 276 pages. Explanation generation from algebraic specification through hyperresolution and demodulation: automated heuristic assistance. (Volumes I and II). DAI V48(02), SecB, pp493. University Microfilms Order Number ADG87-11310. Computer Science.

DEBRAY, SAUMYA KANTI. State University of New York at Stony Brook Ph.D. 1986, 246 pages. Global optimization of logic programs. DAI v47(12), SecB, pp4957. University Microfilms Order Number ADG87-07050. Computer Science.

ETHERINGTON, DAVID WILLIAM. The University of British Columbia (Canada) Ph.D. 1986. Reasoning with incomplete information: investigations of non-monotonic reasoning. DAI V48(01), SecB, pp185. This item is not available from University Microfilms International ADG05-60031. Computer Science.

FREDERKING, ROBERT ERIC. Carnegie-Mellon University Ph.D. 1986, 173 pages. Natural language dialogue in an integrated computational model. DAI V48(01), SecB, pp186. University Microfilms Order Number ADG87-09383. Computer Science.

FRISCH, ALAN MARK. The University of Rochester Ph.D. 1986, 127 pages. Knowledge retrieval as specialized inference. DAI v47(12), SecB, pp4957. University Microfilms Order Number ADG87-08227. Computer Science.

GUPTA, ANOOP. Carnegie-Mellon University Ph.D. 1986, 264 pages. Parallelism in production systems. DAI v47(12), SecB, pp4959. University Microfilms Order Number ADG87-02889. Computer Science.