YAPS: A PRODUCTION RULE SYSTEM MEETS OBJECTS*

Elizabeth Allen

University of Maryland

ABSTRACT

This paper describes an antecedent-driven production system, YAPS (Yet Another Production System) which encodes the left hand sides of production rules into a discrimination net in a manner similar to that used by Forgy ([Forgy 81], [Forgy 79]) in OPS5. YAPS, however, gives the user more flexibility in the structure of facts in the database, the kinds of tests that can appear on the left hand side of production rules and the actions that can appear on the right hand side of the rules. This flexibility is realized without sacrificing the efficiency gained by OPS5 through its discrimination net implementation. The paper also discusses how YAPS can be used in conjunction with object oriented programming systems to yield a system in which rules can talk about objects and objects can have daemons attached to them. It discusses methods of dividing YAPS into independent rule sets sharing global facts.

1. Introduction

Production systems are an important part of research in Artificial Intelligence today. In expert systems, some form of productions rules is almost always used to represent the underlying knowledge of the system. With production systems, however, comes the problem of inefficiency in the basic cycle when facts in the data base are matched against the left hand sides of productions rules to determine those rules ready to fire. This problem was addressed by Forgy [Forgy 79] in his thesis describing OPS. He observed that during a production rule cycle, only a few facts are added to or removed from the data base and, consequently, a production system could be much more efficient if it remembered between cycles what facts matched the patterns in the left hand sides of the production rules. Then, whenever a fact was added or deleted, matches would be made or deleted and rules which had been completly matched would be the rules ready to fire. To compare new facts against the set of production rules, he encoded left hand side patterns in a discrimination net. This method of saving matches between cycles of the production system cut out much of the overhead and allowed larger rule systems to run without needing to swap rules in and out of active use as many expert systems currently do.

However, Forgy's OPS5 production system [Forgy 81] has some drawbacks which are fairly serious from a user's point of view. They are:

(1) Facts in the database are restricted to flat lists of atoms and numbers; nested sublists are not supported. This restricts facts to having only one arbitrarily long field of parameters as well as preventing a user from structuring his facts conveniently.

(2) Tests that appear on the left hand side of a production rule can only use equality, inequality and arithmetic comparisons involving no more than two variables.

(3) Right hand side actions are restricted to a set of actions specified by OPS5, and though these actions cover some of the things a user might want to do, they do not allow a user to write arbitrary lisp bodies. This is an unwanted and unnecessary restriction.

(4) The syntax of OPS5 is difficult to deal with. Often, it is not at all obvious how to interpret the patterns on the left hand sides of production rules. While the syntax problem is not crucial to running a production system, it can be a problem when writing production rules and when reading production rules written in OPS5.

(5) Right hand sides of production rules are always interpreted by the OPS interpreter. There is no way to gain the speed up of compiling the rules.

(6) OPS5 is hard to run under program control. It is designed mainly to be used as a top level controller of a system of just production rules.

This paper describes the production system YAPS (Yet Another Production System) which is designed to allow greater flexibility and readability of production rules while not giving up the efficiency gained by OPS5. YAPS has none of the above restrictions and has a clear, straightforward syntax making productions much easier to read and modify. YAPS may be run conveniently under program control and can maintain and run multiple rule sets and data bases. This makes YAPS a much more gen-

----

* Funding for this project was provided by the Goddard Space Flight Center in Greenbelt, Maryland.

eral tool.

2. A User's View of YAPS

Facts in YAPS may be arbitrarily nested lisp lists of atoms and integers. Patterns may contain variables which match any constant lisp expression within a fact. Variables are atoms whose first character is a hyphen (-); a hyphen appearing alone will match anything. A sample production rule in YAPS is:

```
(p climb
        (goal (reach monkey -height)
              (location monkey -x))
        (location monkey -x)
        (reach monkey -reach)
        (box -box -boxsize)
        (location -box -x)
        (size monkey -monkeysize)
   test (>= -reach -boxsize)
        (<= -height (+ -monkeysize -boxsize))
   -->  (remove 1 3)
        (fact reach monkey ^(+ -monkeysize
                               -boxsize))
        )
```

The keyword test separates left hand side patterns from the left hand side tests. Note that the second test referneces three variables. Patterns may also be used which specify that particular facts may not be in the database are also allowed. For example,

```
(p find-largest
        (data -x)
        (~ (data -y) with (> -y -x))
   -->  (let ((-ans (calculation -x)))
             (remove 1)
             (fact calculate -x -ans)
             (fact data -ans))
        )
```

This rule guarantees that when it runs, the largest data in the data base will be bound to "-x". The keyword with separates the list of not patterns from the tests associated with them in the not clause. An arbitrary number of not clauses and tests may appear on the left hand side of a production rule. The right hand side, as can be seen in this rule, can contain arbitrary lisp bodies.

In addition, YAPS makes it easy to run production systems under program control. Whenever a goal fact (in the form of a fact whose car is "goal") is added to the YAPS data base and the production rules are not already running, the production system automatically runs until all goals are removed from the system or until there are no more productions ready to fire. In addition, if there are outstanding goals in the data base and a fact is added which allows one of the rules to fire, the system is run. This gives YAPS the desired daemon behavior.

YAPS also supports multiple rule sets and data bases which can be entered and exited by the controlling system. Any fact asserted while within a given data base will be asserted in that data base. Facts can also be asserted in a global data base causing the fact to be asserted into all the YAPS data bases and rule sets.

(For more information on YAPS, see the YAPS manual [Allen 82].)

3. Implementation of YAPS

YAPS is implemented in Franz Lisp [Foderaro 80] running under Berkeley UNIX* [Joy et al 81] and using the University of Maryland flavors package ([Wood 82], [Allen et al 82]). The most important structure in YAPS is the discrimination net which encodes left hand sides of production rules in the system. When facts are added to the data base, they are fed into the top of the discrimination net where they are compared against patterns appearing on the left hand sides of the production rules. Each node in the discrimination net has a path (say "car" or "cadr") specifing a position in the fact and an associative list of expected values and child nodes. When a fact matches all the constants of a pattern, it is unified with the pattern, and a binding is generated. All partial bindings are compared against other partial bindings for patterns in the same production rule, and new bindings are generated whenever there is a match. Like OPS5, left hand side tests are performed as soon as a potential binding has values for all the variables in the test, and a binding is only made if the test succeeds. Thus, the tests are performed as early as possible and false partial bindings are pruned early. Bindings which completely match the left hand side of some production rule are placed in the conflict set and, according to the conflict resolution algorithm, one is chosen.

When facts are removed from the data base, all the bindings in which they appear are removed. This is done by associating with each fact the list of bindings in which it appears and by mapping down the list removing bindings as the fact is removed. This differs from OPS5. In OPS5, facts are recompared against the discrimination net upon removal to find bindings in which they appear.

When a production is added to YAPS, a function is defined whose arguments are the left hand side variables and whose bodies are the right hand side bodies. Thus, left hand side variables are just local variables in the right hand side function. When a production rule is run, this function is applied to the list of values of the left hand side variables. These functions may be compiled if the file containing the YAPS productions is compiled. This speeds up the lisp code both by allowing the right hand sides to be compiled and by having macros such as fact expanded at compile time. OPS5 does not define such a function for the right hand side of a production forcing the right hand sides of rules to always be interpreted.

4. Production Systems and Flavor Objects

Object oriented programming in Artificial Intelligence using such systems as MIT's Lisp Machine Flavors [Weinreb & Moon 81] has become popular recently and with good reason. Steps taken to merge production systems with object oriented programming can yield quite useful systems in which facts and productions manipulate objects by viewing them as atomic entities. At the same time, daemons in the form of production rules can be attached to

----

* UNIX is a trademark of Bell Laboratories

objects and can run when certain messages are sent to the object. These objects can have their own individual rule sets and data bases but with the ability to add specific facts considered global information for the composite data base.

YAPS also provides a flavor, the daemon-mix-in flavor, which can be mixed into other flavors giving objects of those flavor pointers to the desired YAPS rule set and data base. The daemon-mix-in flavor defines messages like "buildp", "goal" and "fact" which manipulate its rule set and data base. Then, when a goal message is sent to an object, the desired goal it asserted into its data base and any production rules thus enabled are fired. As the rules fire, they may add more goals to either its own data base or to some other object's data base by sending "goal" messages to other objects. (Of course it may also send other messages to various objects as it so desires since there are no restrictions as to what may appear on the right hand side of a rule.) Another message defined by daemon-mix-in is "get-value". This message is used to get the value of an instance variable. If the value of the variable is "UNBOUND", then a goal is asserted into the object's data base to compute the value of the variable. This provides a mechanism for slots to be filled in as their values are needed.

As an example of using production rules together with flavors, consider the problem of monitoring the usage of files in an operating system. Suppose we want to provide users with the ability to attach daemons to their files and directories specifing actions to be taken any time the file is read from, written to, edited or executed. For example, there might be a file regularly modified by a group of people. A daemon attached to the file could warn anyone who wants to edit the file in case someone else is already editing the file. Also, a system maintainer might monitor a utility for the purpose of profiling its users. He could post a daemon on that utility that would write a message to a log file whenever someone ran the program. This message would give the name of the user and the form of the call. (An operating system which has these capabilities and more is described in [Israel 82].) YAPS in conjunction with flavors does this by defining a "file" flavor and mixing in the daemon-mix-in flavor to get daemons attached to the files. Then whenever a file request was made, a message could be sent to that object in the form of a goal and appropriate actions taken, including, most likely, filling the request.

5. Conclusion

YAPS is an alternative to OPS5 as an antecedent-driven production system. It is comparable to OPS5 in terms of efficiency but allows greater flexibility in facts in the data base and in writing production rules themselves. It is also particularly suitable as the basis for a production rule system which both manipulates objects and has objects which manipulate rules. YAPS does not make the mistake of forcing a system to be completely encoded using production rules or to be controlled at the top level by production rules living in the system. Instead, YAPS is a flexible tool, which combines with other lisp tools to build systems which can take advantage of using production rules in just those places where they are needed.

Acknowledgements

I would like to thank Randy Trigg for reading a prior draft of this paper and the Maryland AI Group for their support.

REFERENCES

[Allen et al 82]
Allen, E., R. Trigg, and R. Wood, Maryland Artificial Intelligence Group Franz Lisp Environment, University of Maryland CS TR-1226, October 1982.

[Allen 82]
Allen, E.M., YAPS: Yet Another Production System, University of Maryland CS TR-1146, February 1982.

[Foderaro 80]
Foderaro, J.K., The Franz LISP Manual, Regents of the University of California, 1980.

[Forgy 79]
Forgy, C.L, On the Efficient Implementation of Production Systems, Ph.D. Thesis, Dept. of Computer Science, Carnegie-Mellon Univ., Feb. 1979.

[Forgy 81]
Forgy, C. L., OPS5 User's Manual, Carnegie-Mellon University CMU-CS-78-116, 1981.

[Israel 82]
Israel, B., Customizing a Personal Computing Environment Through Object-Oriented Programming, University of Maryland CS TR-1158, March 1982.

[Joy et al 81]
Joy, W.N., R.S. Farby, and K. Sklower, UNIX Programmer's Manual, Dept. of Electrical Engineering and Computer Science, Univ. of California, Berkeley, CA, June 1981.

[Weinreb & Moon 81]
Weinreb, D. and D. Moon, Objects, Message Passing, and Flavors, pp. 279-313 in Lisp Machine Manual, Massachusetts Institute of Technology, Cambridge, MA, March 1981.

[Wood 82]
Wood, R.J., Franz Flavors: An Implementation of Abstract Data Types in an Applicative Language, Dept. of Computer Science, Univ. of Maryland, TR-1174, June 1982.