R1: an Expert in the Computer Systems Domain[1]

John McDermott
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

INTRODUCTION. R1[2] is a rule-based system that has much in common with other domain-specific systems that have been developed over the past several years [1, 8]. It differs from these systems primarily in its use of Match rather than Generate-and-Test as its central problem solving method [2]; rather than exploring several hypotheses until an acceptable one is found, it exploits its knowledge of its task domain to generate a single acceptable solution. R1's domain of expertise is configuring Digital Equipment Corporation's VAX-11/780 systems. Its input is a customer's order and its output is a set of diagrams displaying the spatial relationships among the components on the order; these diagrams are used by the technician who physically assembles the system. Since an order frequently lacks one or more components required for system functionality, a major part of R1's task is to notice what components are missing and add them to the order. R1 is currently being used on a regular basis by DEC's manufacturing organization.[3]

THE DOMAIN. The VAX-11/780 is the first implementation of DEC's VAX-11 architecture. The VAX-11/780 uses a high speed synchronous bus, the sbi, as its primary interconnect; the central processor, one or two memory control units, up to four massbus interfaces, and up to four unibus interfaces can be connected to the sbi. The massbuses and particularly the unibuses can support a wide variety of peripheral devices. A typical system contains about 90 components; these include cabinets, periperal devices, drivers for the devices, and cables. There are a large number of rules that constrain the ways in which these components may be associated.

R1'S DEFINING CHARACTERISTICS. R1 is implemented in OPS4, a production system language developed at Carnegie-Mellon University [3, 7]. An OPS4 production system consists of a set of productions held in *production memory* and a set of data elements (eg, state descriptions) held in *working memory*. A production is a rule composed of conditions and actions:

$$P_i \quad (C_1 \;\; C_2 \;\; . \;. \;. \;\; C_n \;\; \text{-->} \;\; A_1 \;\; A_2 \;\; . \;. \;. \;\; A_m)$$

Conditions are forms that are instantiated by memory elements. Actions add elements to working memory or modify existing elements. The *recognize-act* cycle repeatedly finds all production instantiations and executes one of them.[4] R1 exploits this recognition match. Its rules have conditions that recognize situations in which a particular type of extension to a particular type of partial configuration is permissable or required; the actions then effect that extension.

OPS4's two memories have been augmented, for this application, with a third. This memory, the *data base*, contains descriptions of each of the 420 components currently supported for the VAX. Each data base entry consists of the name of a component and a set of eight or so attribute/value pairs that indicate the properties of the component that are relevant for the configuration task. As R1 begins to configure an order, it retrieves the relevant component descriptions. As the configuration is generated, working memory grows to contain descriptions of partial configurations, results of various computations, and context symbols that identify the current subtask.

Production memory contains all of R1's permanent knowledge about how to configure VAX systems. R1 currently has 772 rules that enable it to perform the task.[5] These rules can be viewed as state transition operators. The conditional part of each rule describes features that a state must possess in order for the rule to be applied. The action part of the rule indicates what features of that state have to be modified or what features have to be added in order for a new state that is on a solution path to be generated. Each rule is a more or less autonomous piece of knowledge that watches for a state that it recognizes to be generated. Whenever

[1] This paper describes R1 as it exists in June of 1980; it is a highly condensed version of [5].

[2] Four years ago I couldn't even say "knowledge engineer", now I ...

[3] The development of R1 was supported by Digital Equipment Corporation. The research that led to the development of OPS4, the language in which R1 is written, was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, and monitored by the Air Force Avionics Laboratory under Contract F33615-78-C-1151. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of Digital Equipment Corporation, the Defense Advanced Research Projects Agency, or the U.S. Government. VAX, PDP-11, UNIBUS, and MASSBUS are trademarks of Digital Equipment Corporation.

[4] OPS4's cycle time, though it is essentially independent of the size of both production memory and working memory [4], depends on particular features of the production system (eg, the number and complexity of the conditions and actions in each production); the average cycle time for OPS4 interpreting R1 is about 150 milliseconds. OPS4 is implemented in MACLISP; R1 is run on a PDP-10 (model KL) and loads in 412 pages of core.

[5] Only 480 of these rules are "configuration rules"; the remainder contain more general (non domain-specific) knowledge that R1 needs in order to use the configuration rules.

that happens, it can effect a state transition. If all goes well, this new state will, in turn, be recognized by one or more rules; one of these rules will effect another state transition, and so on until the system is configured. English translations of two sample rules are shown in Figure 1.

```
ASSIGN-UB-MODULES-EXCEPT-THOSE-CONNECTING-TO-PANELS-4

    IF:  THE CURRENT CONTEXT IS ASSIGNING DEVICES
                 TO UNIBUS MODULES
         AND THERE IS AN UNASSIGNED DUAL PORT DISK DRIVE
         AND THE TYPE OF CONTROLLER IT REQUIRES IS KNOWN
         AND THERE ARE TWO SUCH CONTROLLERS NEITHER
                 OF WHICH HAS ANY DEVICES ASSIGNED TO IT
         AND THE NUMBER OF DEVICES THAT THESE
                 CONTROLLERS CAN SUPPORT IS KNOWN

  THEN:  ASSIGN THE DISK DRIVE TO EACH OF THE CONTROLLERS
         AND NOTE THAT THE TWO CONTROLLERS HAVE BEEN
                 ASSOCIATED AND THAT EACH SUPPORTS ONE DEVICE

PUT-UB-MODULE-6

    IF:  THE CURRENT CONTEXT IS PUTTING UNIBUS MODULES
                 IN BACKPLANES IN SOME BOX
         AND IT HAS BEEN DETERMINED WHICH MODULE TO TRY
                 TO PUT IN A BACKPLANE
         AND THAT MODULE IS A MULTIPLEXER TERMINAL INTERFACE
         AND IT HAS NOT BEEN ASSOCIATED WITH ANY PANEL SPACE
         AND THE TYPE AND NUMBER OF BACKPLANE SLOTS
                 IT REQUIRES IS KNOWN
         AND THERE ARE AT LEAST THAT MANY SLOTS AVAILABLE
                 IN A BACKPLANE OF THE APPROPRIATE TYPE
         AND THE CURRENT UNIBUS LOAD ON THAT BACKPLANE
                 IS KNOWN
         AND THE POSITION OF THE BACKPLANE IN THE BOX IS KNOWN

  THEN:  ENTER THE CONTEXT OF VERIFYING PANEL SPACE
                 FOR A MULTIPLEXER
```

Figure 1: Two Sample Rules

It is usual to distinguish the matching of forms and data from search; for example, in discussing the amount of search occurring in a resolution theorem prover, the unification of clauses is considered to be part of the elementary search step. But Match is also a method for doing search in a state space [6]; it is analogous to methods such as Hill Climbing or Means-ends Analysis, though much more powerful. The characteristic that distinguishes Match from other Heuristic Search methods is that in the case of Match the conditions (tests) associated with each state are sufficient to guarantee that if a state transition is permissible, then the new state will be on a solution path (if there is a solution path). Thus with Match, false paths are never generated, and so backtracking is never required. Match is well suited for the configuration task because, with a single exception, the knowledge that is available at each step is sufficient to distinguish between acceptable and unacceptable paths. The subtask that cannot always be done with Match alone is placing modules on the unibus in an acceptable sequence; to perform this subtask, R1 must occassionally generate several candidate sequences.

The fan-in and fan-out of R1's rules provide a measure of the degree of conditionality in the configuration task. The *fan-in* of a rule is the number of distinct rules that could fire immediately before that rule; the *fan-out* is the number of distinct rules that could fire immediately after the rule. The average fan-in and fan-out of R1's rules is 3. The graph of possible rule firing sequences, then, has 666 nodes, one for each of the rules (excluding the 106 output generation rules); each of these nodes has, on the average, three edges coming into it and three going out. It should be clear that unless the selection of which edge to follow can be highly constrained, the cost (in nodes visited) of finding an adequate configuration (an appropriate path through the rules) will be enormous. It is in this context that the power of the Match method used by R1 becomes apparent. When R1 can configure a system without backtracking, it finds a single path that consists, on the average, of about 800 nodes. When R1 must backtrack, it visits an additional N nodes, where N is the product of the number of unsuccessful unibus module sequences it tries (which is rarely more than 2) and the number of nodes that must be expanded to generate a candidate unibus module configuration (which is rarely more than 300).

R1'S EVOLUTION. In a period of less than a year, R1 went from an idea, to a demonstration system that had most of the basic knowledge required in the domain but lacked the ability to deal with complex orders, to a system that possesses true expertise. Its development parallels, in many respects, the development of the several domain-specific systems engineered by Stanford University's Heuristic Programming Project [2]. R1's implementation history divides quite naturally into two stages. During the first stage, which began in December of 1978 and lasted for about four months, I spent five or six days being tutored in the basics of VAX system configuration, read and reread the two manuals that describe many of the VAX configuration constraints, and implemented an initial version of R1 (consisting of fewer than 200 domain rules) that could configure the simplest of orders correctly, but made numerous mistakes when it tried to tackle more complex orders.[6] The second stage, which lasted for another four months, was spent in asking people who were expert in the VAX configuration task to examine R1's output, point out R1's mistakes, and indicate what knowledge R1 was lacking. R1 was sufficiently ignorant that finding mistakes was no problem. Given a criticism of some aspect of the configuration by an expert, all that was necessary in order to refine R1's knowledge was to find the offending rule, ask the expert to point out the problem with the condition elements in the rule, and then either modify the rule or split it into two rules that would discriminate between two previously undifferentiated states. During this stage, R1's domain knowledge almost tripled.

VALIDATION. During October and November of 1979, R1 was involved in a formal validation procedure. Over the two month period, R1 was given 50 orders to configure. A team of six experts

[6] During this first stage, R1's name was XCON.

examined R1's output, spending from one to two hours on each order. In the course of examining the configurations, 12 pieces of errorful knowledge were uncovered. The rules responsible for the errors were modified and the orders were resubmitted to R1 and were all configured correctly. Each of these 50 orders contained, on the average, 90 components; R1 fired an average of 1056 rules and used an average of 2.5 minutes of cpu time in configuring each order. Since January of 1980, R1 has configured over 500 orders. It is now integrated into DEC's manufacturing organization. It has also begun to be used by DEC's sales organization to configure orders on the day they are booked.

CONCLUDING REMARKS. R1 has proven itself to be a highly competent configurer of VAX-11/780 systems. The configurations that it produces are consistently adequate, and the information that it makes available to the technicians who physically assemble systems is far more detailed than that produced by the humans who do the task. There are, however, some obvious ways in which to enlarge its domain so that it can become a more helpful system. Work has already begun on augmenting R1's knowledge to enable it to configure other computer systems manufactured by DEC. In addition, we plan to augment its knowledge so that it will be able to help with the scheduling of system delivery dates. We also plan to augment R1's knowledge so that it will be able to provide interactive assistance to a customer or salesperson that will allow him, if he wishes, to specify some of the capabilities of the system he wants and let R1 select the set of components that will provide those capabilities. Ultimately we hope to develop a salesperson's assistant, an R1 that can help a customer identify the system that best suits his needs.

ACKNOWLEDGEMENTS. Many people have provided help in various forms. Jon Bentley, Scott Fahlman, Charles Forgy, Betsy Herk, Jill Larkin, Allen Newell, Paul Rosenbloom, and Mike Rychener gave me much encouragement and many valuable ideas. Dave Barstow, Bruce Buchanan, Bob Englemore, Penny Nii, Ted Shortliffe, and Mark Stefik contributed their knowledge engineering expertise. Finally, Jim Baratz, Alan Belancik, Dick Caruso, Sam Fuller, Linda Marshall, Kent McNaughton, Vaidis Mongirdas, Dennis O'Connor, and Mike Powell, all of whom are at DEC, assisted in bringing R1 up to industry standards.

REFERENCES

1. Amarel, S. et al. Reports of panel on applications of artificial intelligence. Proceedings of the Fifth International Joint Conference on Artificial Intelligence, MIT, 1977, pp. 994-1006.

2. Feigenbaum, E. A. *The art of artificial intelligence.* Proceedings of the Fifth International Joint Conference on Artificial Intelligence, MIT, 1977, pp. 1014-1029.

3. Forgy, C. L. and J. McDermott. OPS, A domain-independent production system language. Proceedings of the Fifth International Joint Conference on Artificial Intelligence, MIT, 1977, pp. 933-939.

4. Forgy, C. L. RETE: A fast algorithm for the many pattern/many object pattern match problem. Carnegie-Mellon University, Department of Computer Science, 1980.

5. McDermott, J. R1: a rule-based configurer of computer systems. Carnegie-Mellon University, Department of Computer Science, 1980.

6. Newell, A. Heuristic programming: ill-structured problems. In *Progress in Operations Research*, Aronofsky, J. S., Ed.,John Wiley and Sons, 1969, pp. 361-414.

7. Newell, A. Knowledge representation aspects of production systems. Proceedings of the Fifth International Joint Conference on Artificial Intelligence, MIT, 1977, pp. 987-988.

8. Waterman, D. A. and F. Hayes-Roth. *Pattern-Directed Inference Systems.* Academic Press, 1978.