

PARALLEL-SERIAL PRODUCTION SYSTEMS*

Leonard M. Uhr
Computer Sciences Department
University of Wisconsin
Madison, Wisconsin 53706

This paper describes several extensions to standard Production System (PS) languages that appear to make them more conveniently usable for a wider variety of perceptual and cognitive systems. The extensions are described (giving parallel productions, multiple memories, and productions that are implied by productions), and a programming language $((PS)^2M^n)$ is presented (in Uhr, 1978) that incorporates them. Sketches are given indicating how several different kinds of cognitive and perceptual systems might be coded in this extended language, and suggestions are made as to additional extensions that might further augment the power and convenience of such a language.

1. INTRODUCTION

This paper describes a Parallel-Serial Multi-Memory Production System language, $(PS)^2M^n$, that handles 1) parallel sets of productions, 2) more than one working memory in which to look for the conditions of productions, and fire the consequences of successful productions, and 3) productions that are fired by productions. These extensions would appear to be convenient for a variety of systems, where parallel processes and dispersed memories are needed. The following such systems are briefly described

- A) Several "Knowledge Sources," all working in parallel with separate working memories, but communicating through a common single "Black-board" memory.
- B) Systems for parallel computability relaxation, where a number of nodes are examined in parallel, for contextual information that might serve to disambiguate them.
- C) Discrimination nets (e.g. for "concept formation" and "EPAM"), where a sorting tree is used to find the appropriate name.
- D) Parallel-serial perceptual systems, where sensed information is successively transformed, by layers of parallel feature-detectors and characterizers, into successively more abstract memory representations.

* This research has been partially supported by National Science Foundation grant MCS76-07333.

- E) Parallel search for solution paths in problem-solving.

See Uhr, 1978, for programs (in EASEy/Snobol*) for $(PS)^2M^n$ and for a traditional (serial, single-memory) Production System, along with a short Primer description of the language.

1.1 Parallel-Serial Production Systems With Many Working Memories

It is often necessary (as in systems for perception, memory search and heuristic problem-solving) to apply a whole set of transformations, or productions, to a Working Memory store in parallel. Systems with several knowledge sources frequently use parallel processes and several memories. In multi-layered or hierarchical perceptual systems, the conditions of productions must be looked for in one WM_i and the consequent acts effected on another WM_j . Therefore a whole set of $WM, WM_1, \dots, WM_N$ are needed. Each production will look at a specified WM_i (or several WM) and, if fired (because all its conditions have been satisfied by a match with elements in that WM_i), effect its consequent acts (e.g. insertions, deletions, arithmetic operations, generations of productions, deletions, arithmetic operations, generations of productions, input-output) on the specified WM_j .

* See Uhr, 1974a; Griswold et al., 1968.

This paper describes extensions to current Production Systems that handle such processes, without the often cumbersome code otherwise needed, but also in a well-structured and simple way.

1.2 Traditional Production Systems

A Production System (PS) (Newell and McDermott, 1975; Rychener, 1976; Waterman, 1975) consists of an ordered set of Productions (P), $P_1, \dots, P_M$, each with a set of Conditions (C) and a set of consequent Acts (A), plus one Working Memory (WM). E.g.:

P1: RED BLUE $\Rightarrow$ GREEN DELETE (RED) DELETE (BLUE)
P2: BROWN YELLOW $\Rightarrow$ DIRT
P3: GREEN-DIRT = POOR-GRASS (STOP)
WM: RED BLUE YELLOW BROWN

Each Condition specifies a set of elements. An element specifies a string (or list structure) of sub-elements to be matched with strings found in WM. The match usually demands that the first sub-elements match, and that each subsequent sub-element in C_i match a subsequent (but not necessarily the next) sub-element in WM. Variables are usually bound by whatever they find in the first match where they occur (rather than have the system try all possibilities). Acts specify insertions or deletions in Working Memory, the generation of new Productions, or any of several primitive Acts, like OUTPUT, or STOP. A Production "fires" its Acts if all its Conditions are satisfied. Usually the system moves from the top Production down, until one Production fires, when it jumps back to the top, to repeat the process. Some systems use different flows, e.g. continuing on to the next production whether the prior one fired or not (note that this does not mean that Productions are applied in parallel, for each fired Production immediately effects its Acts, and therefore modifies WM). Some systems find all Productions that succeed, and then use one of a variety of criteria for choosing and firing one Production, e.g. the one that matches the most recently added element in WM (Newell and McDermott, 1975).

1.3 Parallel-Serial Multi-Memory Production Systems

Living cognitive systems effect sets of parallel transformations, and use a number of temporary buffer Working Memories. Such parallel-serial multi-layered architectures seem similarly desirable for computer-programmed systems since (once they run on the parallel-serial computers,

see e.g. Duff, 1976, Lipovski, 1977, that current technology is on the verge of making feasible) they offer great economy and power.

$(PS)^{2^n}$ extends traditional Production Systems so that they can conveniently handle parallel processes applied to more than one Memory. The key extensions include the following.

- A) Any number of Memories can be specified and used, with sets of Productions looking at and firing into each.
- B) Each Production can look at several Memories, and fire into several Memories, if desired.
- C) Sets of Productions can be applied in either a Parallel or a Serial mode, as specified, and the mode can be changed during a run.
- D) A Production can fire one or more Productions to be applied, as specified.
- E) After a Production succeeds in firing, the system can either attempt to fire the next Production, or go back to attempt to fire the first Production, as specified.

These extensions allow for relatively convenient coding of a variety of perceptual and cognitive systems, some examples of which are described below.

1.4 Some of the Uses of Multiple Memories, and of Parallel and Dynamically Implied Productions

A single Memory can always be set up to do anything that multiple memories can do (since standard Production Systems are based on Post Productions they are universal computers, and anything can be programmed in them)-but often at a heavy price. Essentially, the system can no longer use the Memories' names and attendant pointers to directly access sub-sets of information in the relevant Memories. Rather, elements in the single Memory must be given appropriate name-like tags, and the system must use the relatively slow Production-matching routines to search for the appropriately tagged elements. All the elements in a particular Memory could be grouped together, so that only one tag was needed. But that means the system must work with and try to match a large and cumbersome list of lists - and that can be a slow and costly process. Alternately, each element can be separate, but each with the same Memory tag. But that means the search for the appropriate tag must be turned into a loop, iterating through the entire Memory.

The original development of Production Systems was directed toward work with a very small

"Working Memory" for scratch-pad-like intermediate results, one that models the human "short-term memory" that is capable of handling some small number like 7 items. For such uses there is no need to add more Memories or other structures. But for a general-purpose programming language multiple Memories serve a number of purposes.

Without a parallel capability of the sort provided in $(PS)^{2M^n}$, a program must make several extra passes through a set of transforms to simulate parallel processes. The first pass would not actually change the Memory; rather, it would output its set of acts each specially tagged as a temporary output. Then the program would have to keep applying a set of Productions that looked for all these tags, use the output information from the tagged elements to modify Memory, and erase all these tagged temporary elements.

The dynamic implication of transforms gives the programmer powerful control capabilities for structuring the flow of processes. Without this feature a program must use a production that adds an element to Memory that only one other Production - the one to be fired next - will succeed in matching. This means the system must make an extra cycle through its productions to find that next production, in addition to adding and deleting the identifying element. Dynamic Productions make this process direct and simple.

The present system can use dynamically implied productions, but it cannot generate them. For it can insert newly generated productions only into its main list of Productions (as is done by standard Production Systems). But it also needs the ability to insert a Production's name into the list of Consequences of another Production, and to choose that Production correctly. This needs a capability of modifying a Production, and not simply creating one, and of accessing and conveniently storing histories of the names of Productions fired. Such an extension is discussed below, as part of a more general ability to create, modify, and erase both Productions and Memories - that is, to treat productions and memories in exactly the same way.

2. EXAMPLES OF SYSTEMS THAT COULD FRUITFULLY USE $(PS)^{2M^n}$

2.1 Parallel Sets of "Knowledge Sources" as Parallel Communicating Production Systems

An attractive structure for cognitive and perceptual systems is one that uses a set of independent "Knowledge Sources" (e.g. Reddy, 1973, Lowerre, 1976), or "demons" (e.g. Lenat, 1977). Each works at the same time, independently of the others, and therefore could be executed in parallel by one processor in a suitable network of minicomputers, or parallel processors. All communicate with one another by passing information into and out of a common "blackboard" memory. Such systems cannot be conveniently coded in a standard Production System language like PSG or PAS-2, because the knowledge sources act in parallel, and often contain parallel processes.

Such a system can quite straightforwardly be handled by $(PS)^{2M^n}$, as follows: Each knowledge Source is executed in turn, since each works with its own local memory. Then, after all have completed their processes, a set of parallel production transfers their (present, intermediate) results to the common Blackboard Memory. Then a second set of parallel productions transfers selected information from the common Memory to the individual Knowledge Sources' memories.

The parallel capabilities of $(PS)^{2M^n}$ make the transfer of information into and out of the common Memory very convenient, since there is now no danger of unwanted interactions between Knowledge Sources. And now each Knowledge Source can itself contain a mixture of parallel and serial flows of productions. And each can contain one (or more) separate memories, whereas in a standard PS each subset of memories would have to be protected (in a very cumbersome way) from all Knowledge Sources that did not have access to it.

Most large Production-oriented programs of this sort appear to be Production Systems in spirit, but without being coded in an actual Production System language. The extensions handled by $(PS)^{2M^n}$ would appear to move in the direction of a language that might be suitable for convenient coding and running of such large production Production Systems.

2.2 Compatibility Pairs

A production for relaxation using pairs of compatibility labels (e.g. Zucker, 1977; Davis and Rosenfeld, 1978; Tennenbaum and Barrow, 1976)

need only be a 2-tuple, designating the label to be looked at and the context label. When both are found the label is fired into an output Memory. But each label must be searched for in its designated memory, and many such productions must be applied in parallel. And to iterate to more distant contexts each must imply the next production to apply. The present system will handle deterministic constraints, if it raises a flag with each success, tests and lowers the flag after each iteration, and then stops iterating after one pass without raising the flag (meaning that nothing new has been accomplished). Extensions to be discussed below will handle probabilistic constraints and dynamic stopping rules, and also situations that mix compatibility pairs with other types of transforms.

2.3 A Production System that Uses Compatibility Pairs to Choose Among Productions

Zucker, 1977, has suggested that compatibility pair relaxation techniques might fruitfully be used to choose the single judged-best Production to fire next, when a serial Production System finds that more than one Production might fire. (This is the case in those standard Production Systems that, instead of firing the first Production that succeeds, get all Productions that succeed, and then use some criterion to choose among them.)

(PS)^{2M} allows such a system to be coded entirely as a parallel-serial Production System, as follows:

List all the Productions to be chosen among as a parallel set on the MASTER list (the main list of productions - see Uhr, 1978). Each Production implies all of its compatibility labels as dynamically implied Productions on its set of ACTS. Thus all Productions that succeed fire their compatibility labels onto the set of (parallel) productions that will fire next, after this parallel set has been completed. Then this parallel set of compatibility labels will be applied. This procedure continues, as in the proceeding section, until either a single production remains or no more relaxation can be done. Thus the separate relaxation phase is absorbed into and handled by the (PS)^{2M} Production System.

2.4 Discrimination Nets, Concept Formation, EPAM

The program uses a tree of Productions, each making one test. The Master list starts with a

Production that tests for the first node of the discrimination net (really a tree), followed by all Productions on its negative path. If it succeeds it implies the Production on its positive path to be immediately executed in the serial mode followed by all Productions on that node's negative path. Thus the fragment of a discrimination tree shown in Figure 1 needs the Productions shown in Table 1 (a number refers to the test for that node):

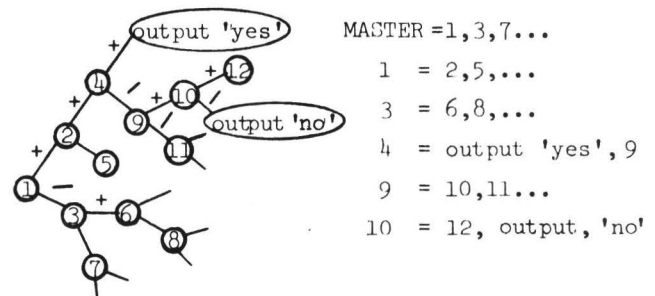


Figure 1

Table 1

Note that the only Productions on the MASTER list are the first and the successive '-' (failure) branches from it. All other Productions are implied by Productions that fire.

This program uses a true discrimination tree, where each Production tests only the single next node. This is in sharp contrast to the EPAM program (Waterman, 1975) coded in PAS-2 (which uses a separate template for each entire path in the tree, and has no ability to dynamically imply transforms).

2.5 Layered Perceptual Systems (e.g. Cones, Pyramids, Hierarchical Relaxation)

The use of a sequence of several memories between pairs of which are sandwiched sets of parallel productions can be quite straightforwardly coded as a parallel-serial production system. Productions need only be stored as parallel sets, their INPUT MEMORY and OUTPUT MEMORY specified for each such set, with the OUTPUT MEMORY for one set the INPUT MEMORY for the corresponding subsequent set.

Dynamically implied transforms can be handled as productions that are implied as consequent acts of successful productions. Iterations through relaxation can be handled by using the same INPUT MEMORY and OUTPUT MEMORY for several cycles. These would be especially attractive for parallel-serial layered perceptual systems such as the cones and pyramids being developed by Hanson and Riseman, 1974; Tanimoto, 1976;

Uhr, 1972, 1974b, and others.

Such systems would benefit greatly from several extensions (that are projected for future productions systems). These include a) weights, thresholds, and probabilistic productions, b) a convenient way to iterate through a whole array of memory locations (as in the 2-dimensional retinal mosaic), applying the same set of parallel productions in parallel to each cell in the array, and c) techniques for choosing among alternative implications (as in naming and describing), and for deciding when to choose (as in deciding when to stop relaxing).

2.6 Heuristic Problem-Solving

A heuristic problem-solver could take advantages of several levels of parallel processes, given a suitable language. Typically, a whole set of heuristic evaluation functions is to be applied in parallel to a node that might be a candidate for further search. And a whole set of such candidate nodes might conveniently be evaluated in this way, in parallel.

A $(PS)^2M^n$ program with the following structure can handle such a system:

The set of heuristics is coded as a parallel set of Productions.

For a heuristically guided search for a path from a set of Givens to a Goal (or from Goal to Givens), the Givens (or the Goal) are input as elements in the first WM. The Rules of Inference (sometimes called legal moves, transformations, etc.) are coded as a parallel set of productions and put on the MASTER list. They are applied, giving the buds (which are put into a second WM) from the presently achieved expressions, deleting elements in the first WM, and firing set of heuristics (coded as a second parallel set of productions). The heuristics now look at these budded expressions, and accumulate the evaluation of each. They are followed by a serial set of productions that chooses and puts into the first WM the most highly valued expression(s), and deletes all expressions in the second WM. The Rules of Inference are again applied, and this process continues.

I have ignored a number of important details, but almost certainly several more Working Memories will be preferable - e.g. to store the pointers back from buds to their fathers for later use in getting the solution-path, and to get that path.

3. SOME ADDITIONAL EXTENSIONS THAT WOULD SEEM TO BE USEFUL

There are several further extensions that would appear to make for substantial further improvements in the ease of coding of cognitive and perceptual systems, and the resulting transparency and clarity of structure of the resulting systems. These include the following:

A) Probabilistic transforms would allow for the emerging of multiple implications, from perceptual characterizers or problem-solving heuristics. This entails rather simple extensions: 1) Associated with each element of a condition will be a number (that can be interpreted as a weight, probability, fuzzy value, or whatever the programmer desires), associated with each production will be a threshold, and associated with each implied act will be a weight. When the combined weights of successful conditions exceeds the Production's threshold, the Production will be considered to succeed, and fire its implications with their assigned weights. 2) Several implications of the same thing will have their weights merged. 3) When this is specified in the program, then the system will choose among alternative implications from among some mutually exclusive set that the programmer has specified.

B) A set of Productions should be able to iterate conveniently through an array of memories. This would allow for very convenient and straightforward coding of perception systems, e.g. pyramids and cones, and relaxation labeling, where sets of transforms (coded as Productions) could be applied to the different cells, or regions, of the retinal image, or internal abstracted images.

C) A system should have a more general set of capabilities for creating and erasing lists (whether Productions or Memories), and adding and deleting information to and from these lists. This would allow for the convenient building, and modifying, or complex list structures and graphs of information. If done with as much generality as possible, it should also make Productions and Memories, and their manipulations, quite similar.

4. DISCUSSION

Today's production systems have a heritage much older than Post's productions, and much broader than problem-solving. For Post productions are probably the single Turing machine-equivalent formulation that most directly embodies that elusive (and possibly imaginary) entity,

"intuitive thinking." The rules of inference of Euclid's geometry, and of Aristotle's syllogism, are very similar to productions. Consider the linguist's rewrite rules and transformations, the physicist's Feynman diagrams, the psychologist's S-R mappings, the "if a,b,c, then d,e,f" lines of reasoning of detective, diagnostician or trouble-shooter.

So an embodiment of thinking in terms of productions would seem especially desirable. We could use it conveniently; we could understand it; and it might well be appropriate, and powerful. But the particular production system chosen should be sufficiently flexible and powerful for convenient programming.

The three major extensions to Production Systems incorporated into (PS)²Mⁿ - 1) the ability to apply productions in parallel, when desired, 2) the partitioning of information into several working memories, and 3) the firing of productions to be applied next by productions that are fired - often appear to go together, and to complement and enhance one another. But one might prefer to use only one, or two, of these extensions, for the following reasons:

1) If it is known that all programmers who will use the production system language will code all their programs using only serial productions, then the parallel capability is not needed. But many cognitive processes, especially in perception and associative memory search but also in heuristic deductive problem-solving, appear to have important parallel components. And although yesterday's computers are serial, today's are beginning to be parallel, and tomorrow's will become increasingly parallel.

2) From a certain esthetic perspective as to the meaning of "simplicity" the single Working Memory seems simpler than several Working Memories. But if a program uses several memory stores it seems less simple to throw them all together into one, and then do a lot of otherwise unnecessary searching and matching to find the one that is needed.

3) Productions that fire productions to be applied may well violate the simple modular structure of standard production languages, with their top-down flow through the stack of productions. But they may more closely reflect the actual operative structure and flow of the program, in the sense of the productions that actually will fire. Without this capability a program becomes cluttered up with cumbersome and time-consuming devices that make use of the addition and deletion of special data symbols to direct the program's flow.

REFERENCES

- Davis, L.S. and Rosenfeld, A. Hierarchical relaxation, In: Comp. Vision Sys. (Ed. by A.R. Hanson and E.M. Riseman), New York: Academic, 1978, 101-110.
- Davis, R. and King, J. An overview of production systems, CS AIM-271, Stanford, 1975.
- Duff, M. J. B. CLIP 4: a large scale integrated circuit array parallel processor, Proc. IJCPR-3, 1976, 4, 728-733.
- Erman, L. D. and Lesser, V. R. A multi-level organization for problem-solving, PROC. 4th IJCAI, 1975, 483-490.
- Feigenbaum, E., Buchanan, B. and Lederberg, J. On generality and problem-solving, Machine Intelligence 6, Edinburgh, 1971, 165-190.
- Griswold, R. E., Poage, J. F. and Polonsky, I. P., The SNOBOL4 Programming Language, Englewood-Cliffs: Prentice-Hall, 1968.
- Hanson, A. R. and Riseman, E. M. Pre-processing cones: a computational structure for scene analysis, COINS TR 74C-7, U. Mass., 1974.
- Lenat, D. B. Automated theory formation in mathematics, Proc. 5th IJCAI, 1977, 833-842.
- Lipovski, J. On a varistructured array of processors, IEEE Trans. Comp., 1977, 26, 125-138.
- Lowerre, B. T. The Harpy Speech Recognition System, Ph.D. Diss., CMU, 1976.
- McDermott, J., Newell, A. and Moore, J. The efficiency of certain production system implementations, CS TR, CMU, 1976.
- Newell, A. and Simon, H. A. Human Problem Solving, Prentice-Hall, 1972.
- Newell, A. and McDermott, J. PSG manual, CS TR, CMU, 1975.
- Reddy, D.R., et al. HEARSAY-1 speech understanding system, Proc. IJCAI-3, 1973, 185-193.
- Rychener, M. D. Production Systems as a Programming Language, Ph.D. Diss., CMU, 1976.
- Tanimoto, S. L. Pictorial feature distortion in a pyramid, Comp. Graphics Image Proc., 1976, 5, 333-352.
- Tennenbaum, J.M. and Barrow, H.G. IGS: A paradigm for integrated image segmentation and interpretation, Proc. IJCPR 3, 1976, 504-513.
- Uhr, L. Layered "recognition cone" networks that pre-process, classify and describe, IEEE Trans. Comp., 1972, 21, 758-768.
- Uhr, L., EASEy: An English-like programming language, CS TR 233, U. Wis., 1974. (a)
- Uhr, L., A model of form perception and scene description, CS TR 231, U. Wis., 1974. (b)
- Uhr, L., Parallel-serial production systems with several memories, CS TR 313, U. Wis., 1978.
- Waterman, D. A., Adaptive production systems, Proc. 4th IJCAI, 1975, 296-303.
- Zucker, S. W. Relaxation labeling and the reduction of local ambiguities, Proc. IJCPR-3, 1976, 4, 852-861.