Contents

1 Introduction — production systems **2**

2 How production systems work **2**

3 The recognise-resolve-act cycle **2**

4 Example **3**

5 Conflict resolution **4**

6 A PS-based psychological model — seriation **4**
6.1 What's the model for? 6

7 Modularity **6**

8 So what are production systems for? **6**

9 The functional architecture of cognition **7**

10 Levels of theoretical commitment **8**

11 You can't play 20 questions with nature and win **8**

12 Production system model of skill-correction **9**

AI lecture 4

Jocelyn Ireson-Paine

February 14, 1996

1 Introduction — production systems

This lecture continues the theme of rule-based systems, applying it this week to rule-based systems as cognitive models (AI as psychology). In the early days of rule-based systems, the phrase *production system* was often used for any kind of rule-based system (or perhaps just for forward-chaining systems, I'm not sure). These days, it's reserved for forward-chaining systems intended as cognitive models.

I shall deal this week with the "classic" production-system architecture, as developed by Newell and Simon in the early 70s. Next week, I'll talk about SOAR, Newell's recent (1986–) ambitious attempt to provide a PS-based unified theory of cognition.

2 How production systems work

In naming the parts of a PS, we talk about *long-term memory* (the rulebase) and *short-term* or *working memory* (the data). The rules consist of a *condition* and an *action*. The action is sometimes called a *conclusion*. Sometimes the rules are called *productions*.

The inference engine is often called the *production-system interpreter*.

Production systems work by *forward-chaining*. As discussed last week, this is a kind of inference which works forward from data to new data. A piece of data comes into working memory. If it matches a rule, the inference engine will *fire* that rule and perform its action. This action will most commonly add some more data to memory; in turn, that may trigger more rules. It's also possible for actions to delete data from memory, initiate motor commands, and even add new rules to the rulebase. See last week's notes.

The user interface in a PS is usually trivial, designed with few frills to give the researcher a view of what's going on inside. Whereas the LTM and STM are intended to correspond to functional parts of the mind, the user-interface obviously is not.

3 The recognise-resolve-act cycle

The idea is that the system keeps running through a cycle which consists of three stages:

1. **Recognise**. Go through the rules, testing the conditions of each against short term memory. Note the ones whose conditions match, and call them the *conflict set*.
2. **Resolve**. From the conflict set, pick one rule. Deciding which to use is called *conflict resolution*, and I'll say more about it later. If the conflict set is empty, or you can't resolve any rules, stop.
3. **Act**. Obey the rule's action. Doing so is often called *firing* the rule.

4 Example

As last week, use the skiing knowledge base. A simple set of rules to infer which resort a skier should go to, given his purpose (to have fun or to practise serious skiing), the number of lessons he's had so far, and the number of press-ups he can do without collapsing.

```
1 Resort = st sartre if
      Rating = beginner and
      Purpose = fun.

2 Resort = schloss heidegger if
      Rating = beginner and
      Purpose = serious.

3 Resort = chateau derrida if
      Rating = advanced and
      Purpose = serious.

4 Resort = wittgenstein gladbach if
      Rating = advanced and
      Purpose = fun.

5 Rating = beginner if
      Lessons < 30.

6 Rating = beginner if
      Lessons >= 30 and
      Fitness = poor.

7 Rating = advanced if
      Lessons >= 30 and
      Fitness = good.

8 Fitness = poor if
      Pressups < 10.

9 Fitness = good if
      Pressups >= 10.
```

Start by assuming that STM contains these facts:

```
Pressups = 15.
Lessons = 178.
Purpose = fun.
```

We now repeat the recognise-resolve-act cycle as follows:

Recognise 1. Only rule 9 matches, so the conflict set is 9.

Resolve 1. Rather trivial, this! Pick rule 9.

Act 1. Add `Fitness = good` to STM.

Recognise 2. Rule 7 matches. Note that rule 9 also does - the new data doesn't contradict its conditions. So the conflict set consists of rules 7 and 9.

Resolve 2. Do we pick 7 or 9? There are various heuristics for conflict resolution. In this example, I'll use *refractoriness*, saying that if possible, we pick a rule

that hasn't fired before. The justification behind this to to keep attention focussed on new things, and prevent the system getting into a loop. So pick 7.

Act 2. Add `Rating = advanced` to STM.

Recognise 3. Rules 4,7 and 9 match, so put all three into the conflict set.

Resolve 3. Do we pick 4, 7, or 9? Use refractoriness again, so pick 4.

Act 3. Add `Resort = wittgenstein gladbach` to STM.

Recognise 4. Rules 4,7 and 9 match, so put all three into the conflict set.

Resolve 4. Do we pick 4, 7, or 9? Well, none of the rules would tell us anything new, so we might as well stop.

5 Conflict resolution

In deciding what rule to pick from the conflict set, most production systems use one or more of these principles:

1. *Specificity*. If two rules could be chosen, pick the one whose conditions are most specific. E.g. "it has wings and swims" over "it has wings".

2. *Refractoriness*. If a rule could be chosen, but it has matched exactly the same data before and been chosen, then ignore it. This helps to stop the system getting into infinite loops.

3. *Recency*. In most production systems, facts in short term memory are tagged to indicate how recently they were added. If two rules could be chosen, prefer the one that matches the most recent facts. I used refractoriness in my example, but recency would have given the same results.

6 A PS-based psychological model — seriation

This model was developed by Richard Young to explain the cognitive development of children in a seriation task. This involves taking blocks from a pile and lining them up by size. Piaget investigated this in *The Child's Conception of Number* (PSY ED:P 057). He found that the child progressed through three stages:

1. Child lines the blocks up in an arbitrary order; is using "pre-relative" concepts "big" and "small" rather than "bigger" and "smaller".

2. Child eventually succeeds, but only after repeated trial-and-error rearrangements. Has acquired notions of "bigger" and "smaller", plus the co-ordination "X bigger than Y but smaller than Z". But is still experimenting directly on perceptual data rather than manipulating abstract ideas.

3. Child performs task smoothly, taking the right block each time. All the time, is now considering relationships between all the items, e.g. "look for smallest overall block".

Here's a set of rules which model the start of stage 2.

```
f1: push(Goal = add first block) and
    set(State = continuing task) if
        Goal = seriate and
        State = task just started.

f2: put block at far left and pop goal stack if
        Goal = add first block and
```

```
        holding block in hand.

a1: push(Goal = add another block) if
        Goal = seriate.

a2: pop goal stack and push(Goal = place this block) if
        Goal = add another block and
        holding block in hand.

a3: pickup nearest block if
        Goal = add _ block.

p1: put block at right if
        Goal = place this block.

f3: pickup biggest block if
        Goal = add first block.

e1: examine blocks and set BlockOrder = OK or bad if
        Goal = place this block and
        new arrangement of blocks exists.

e2: pop goal stack if
        Goal = place this block and
        BlockOrder = OK.

e3: exchange last two blocks if
        Goal = place this block and
        BlockOrder = bad.

a4: pickup block similar to last if
        Goal = add another block.
```

These rules come from page 117 of *Learning and Problem Solving 3*, Open University Course D303, Block 4, Units 26–28. You can find it in PSY Oversize AA:O 2–O M and in the RSL on the Open University bookcase — 2nd bookcase on left as you go in the back entrance to the Lankester Room.

Some points about the rules:

- I've written them in roughly the same format as the skiing example. They involve more complexity of STM. Symbol clusters of the form `Goal = ?` are stored on a *stack* in last-on first-off fashion. You can have any number of goals, but only the topmost one is visible to the rule-matcher.
- The rules use a lot of perceptual tests and motor actions.
- In rule a3, `add _ block` will match either of `add first block` and `add another block`. _ is a kind of wild-card.
- The model assumes resolution by specificity.

To simulate, start with a heap of blocks and an STM whose contents is

```
State = task just started
```

6.1 What's the model for?

Piaget proposed that the change from one stage to another involved a "massive structural change". In other words, large change in behaviour must be caused by large change in structure. Unfortunately, he didn't say much about possible mechanisms. Young wanted to explore the idea that, given a suitable computational system, you can get large changes in behaviour from small changes in structure — specifically, by adding small pieces of rule-like knowledge in a way that doesn't interfere with knowledge that's been added before.

To do this, he constructed a set of rules which produced stage 1 behaviour, and then showed that you could add a few more rules to get stage 2, and a few more to get stage 3. So this is a model of the development of seriation skills in children. When a new skill is acquired, a new rule which describes it gets added to the existing productions in LTM. This does not explain *how* such new rules are created, but it's an important step towards such an explanation.

7 Modularity

Young's model illustrates one of the supposed virtues of production systems. They are *modular*. That is, since all rules are syntactically independent, you can easily add new ones without having to re-write old ones. Because of how the recognise-resolve-act cycle works, it will call these new rules automatically if their conditions match STM.

Contrast to (e.g.) improving a Pascal program. If you want to add a new procedure (or predicate definition), you can do so easily. But the program will not call it just because it's there. You must also insert into the existing old code, statements which call the new procedure (or predicate). In production systems, that isn't necessary.

Note that this is a somewhat different use of the word from e.g. cognitive modularity.

In passing, I should say that Matt Elton, in his 3rd year project, appears to have found that Young's seriation system is not as modular as he claims. He claims that if you analyse the stage 1 rules, you find they have more than the minimum number of variables etc needed to achieve stage 1 behaviour. The extra parts are "hooks" which are necessary so that the additional rules will interface smoothly with stage 1 to give stage 2 and stage 3 behaviours.

8 So what are production systems for?

The simplest way to regard them is just as a programming language. Such a language contains just one construct — the rule — and so should be easy to learn; unlike (say) Pascal, with its range of data types, loop structures, conditionals, and other statement types. There have in fact been several production-system languages, most used from the mid-70's to the mid-80's. The best-known is OPS5, promoted for writing expert systems.

Although it's easy to understand how production systems work, it is not in fact easy to write big programs, and production-system languages have fallen out of favour. The problem is that it can be hard to see exactly what the resolution strategies will do, and hence in what order rules will fire and data be added to memory. (The Prolog practical uses a PS in Eden, and I had to add some features to it which avoid such problems.)

The main reason for being interested in them is their use as cognitive models. The notion of a production system probably dates from the 40's, when some logicians used them as idealised mathematical models of computation, in the same spirit as a Turing machine. As far as psychology is concerned, they became important in the early 70's, when Newell, Simon and Shaw adopted them as a model of human cognition. The idea was that, at the lowest level, the brain is of course built from neural hardware. However, this hardware implements short-term and long-term memories, rule matchers, and other components of a production system, in the same way that the logic gates in a computer implement adders, registers, and other components of a digital computer.

We can understand how a computer program works by explaining its operation in terms of adders, registers, and so on. It is not necessary to know how these are built from logic gates. In the same way (say the production-system theorists) we can explain cognitive processes in terms of the rules that are fired, the symbols that are added to STM, and so on. It is not necessary to know what is going on at the neural level.

9 The functional architecture of cognition

This view is often expressed by saying that the the neurophysiological hardware of the brain implements a "cognitive processor", a simple symbol-manipulating machine, and that this is a production system. For a statement of this view, see *Foundations of Cognitive Science* edited by Posner (Springer-Verlag 1989; PSY BH:P 084), Chapter 3. For what Newell and Simon believed to be the eight main architectural features of the mind, see *Computer Models of Mind* by Boden (CUP 1988; PSY KD:B 063), page 164.

But there can be many different types of production system. Here's a list of some of the ways they could differ:

1. The symbols allowed — what, and how many?
2. The resolution methods.
3. The capacity of STM. Is there a maximum size, and if so, what happens when more symbols are added? Do old ones drop out? If so, how does this depend on the time they were added? Is there any kind of "chunking" mechanism, so that symbols can be grouped into a single slot?
4. The rules' conditions. Can these test perceptual data directly, or must they go via STM? What is the maximum size of pattern you can have? Can you have "wild card" patterns — so that □_∠ for example, might match any set of symbols starting with □ and ending in ∠. Can the conditions check how many of some symbol there are?
5. The rule's actions. Can these generate actions directly, or must these go via STM? How many symbols can they add? Can they delete symbols?

Different theorists have proposed different types of cognitive processor. The ways in which one differs from another: the complexity of its rules, the capacity of STM and so on, these define its *functional architecture*. In the same way, a VAX, a PC, and an Apple Mac are all computers, with the same general design. But the details of their functional architecture — the maximum size of number that can be stored, the memory capacity, and so on — are all different.

10 Levels of theoretical commitment

Not all theorists believe in the psychological reality of rules. Those who don't may still use production systems merely as convenient programming languages or notations to express the processes producing some behaviour. This is like writing your model in Pascal: it doesn't commit you to believing that memories are stored as 32 bit words, or that the brain is full of machine code. There's an article by Richard Young *Production Systems for Modelling Human Cognition* in *Expert Systems in the Microelectronic Age* edited by Michie (E.U.P. 1980; PSY KH:M 58), in which he identifies three levels of theoretical commitment. This, the least committed to PSs, he calls the *language level*.

The next level down is the *rule level*. This is the level of Young's own work. We believe in the psychological reality of rules, and perhaps in some aspects of the conflict resolution. But we don't (necessarily) extend this commitment to details of the architecture.

Finally, there's the *immediate processor level*, where the PS architecture is — as described in the previous section — a model of human cognitive architecture. Most of Newell's work has been at that level. It's interesting to note though that Newell's level of committment appears to have changed between the *Psych. Rev.* article quoted below and his PS work.

11 You can't play 20 questions with nature and win

Why did Newell and Simon develop their production system models? I have only looked briefly at the history, but here are some of the influences other authors have noted.

- The idea of PSs already existed, and was brought to Simon. Crevier p 150.
- They were consistent with the rise of cognitivism (against behaviourism); the view that psychologists should be concerned with internal representations of the world. See *Computers and Thought*, by M Sharples, D Hogg, C Hutchison, S Torrance, and D Young (MIT 1989; PSY KH:S531; RSL M92.E00068; Hooke), page 200.
- They are a natural development in the authors' views on problem-solving, first expressed in *Elements of a theory of human problem solving* by Newell, Simon and Shaw, in *Psych Rev* volume 65 (1958) pp 151–.
- Newell and Simon were interested not only in how data is represented, but also in *control* — how we move between different stages of processing. See page 89 of the OU book. There are some situations where means-ends analysis (GPS) appears to be sufficient — if there are clear goals and subgoals. But other problems aren't amenable to means-ends analysis; the subject may just plod on without establishing a clear sequence of subgoals. The idea is that the subject has acquired a certain set of operators (rules), and we're interested in the circumstances under which each is activated.

  Note that control can be very flexible, because the symbols in a production system's memory can be used to represent goals and subgoals. Bagger in Winston's *Artificial Intelligence* is a simple example of this. Bagger works through four stages, using a fact such as "Step=check-order" to keep track of which stage it's on. You could think of this fact as meaning "My goal is to finish stage check-order".

Young's seriation example makes this more obvious. He has rules with conclusions like `Push(goal=add first block)` which adds another goal; conditions like `goal = add first block and holding block in hand` which depends on it, and rules with conclusions like `pop goal-stack` which deletes it.

Contrast with GPS, where goals were added and deleted in a strictly first-on, last-off fashion. They were pushed onto a stack, and popped off when finished.

- There's an early article by Newell, detailing some of the flaws he finds in experimental psychology, and what production systems have to offer. This appears in *You can't play 20 questions with nature and win*, in *Visual Information Processing*, edited by Chase (PSY BC:C 39).

12 Production system model of skill-correction

In *Errors in Children's Subtraction* in *Cognitive Science* 1981, Young and O'Shea set up a PS model of subtraction. They say that in standard educational theory, faulty arithmetic is due to a failure to remember "number facts", aggravated by bad setting-out, fatigue, carelessness, poor concentration, etc. So standard theory has concentrated on models for recall of number facts.

But the children's errors may be due to *failures in execution*. We should regard the child as faithfully following wrong algorithms, not wrongly executing the correct one. Of course, this is a hypothesis; but it's worth taking seriously, because if true, it gives us more control over teaching.

Let's see whether a production system model could show how children might acquire wrong algorithms rather than correct ones. We'll express these as rules, of course. Young and O'Shea devised a set of PS rules which modelled correct subtraction, and then demonstrated that by slight changes to some of the rules, the model could generate most of the observed errors. This model accounts for more than 2/3 of such errors (from an analysis of 1500 sums by ten-year olds).

Some of these rules are faulty because of wrong actions; others, because of wrong conditions. For example,

```
50
46
--
16
--
```

may have been caused by a rule whose conditions don't discriminate between having the zero *under* a digit, or *above*.

What are the implications for teaching? Conventionally, a new skill or aspect of skill is viewed as a new component or procedure. So teachers "consolidate" it by repeated practice or "consolidation training". For example, language textbooks that introduce past tenses and then set translations involving all past tenses and no present tenses. But the rule model shows we must also train the learner in discriminating when to apply rules — to get the conditions right. Such training may actually be hindered by consolidation. In the language example, old present tense rules may be lost or changed in favour of past tense ones.

For a survey of this and other work on PS models of arithmetic skill, see the introduction to *Mind Bugs: The Origins of Procedural Misconceptions* by Kurt VanLehn (MIT 1990; PSY BF:V 032). Note that VanLehn treats PS models as *descriptions* in

the same way that differential equations describe circuits or grammars describe linguistic behaviour (page 3).

See *Rules of the Mind* by John Anderson (LEA 1993; PSY BH:A 547) for a recent work by somebody who still believes in the psychological reality of production rules, and who contrasts his views with those of some connectionists. His arguments for rules are on page 10.