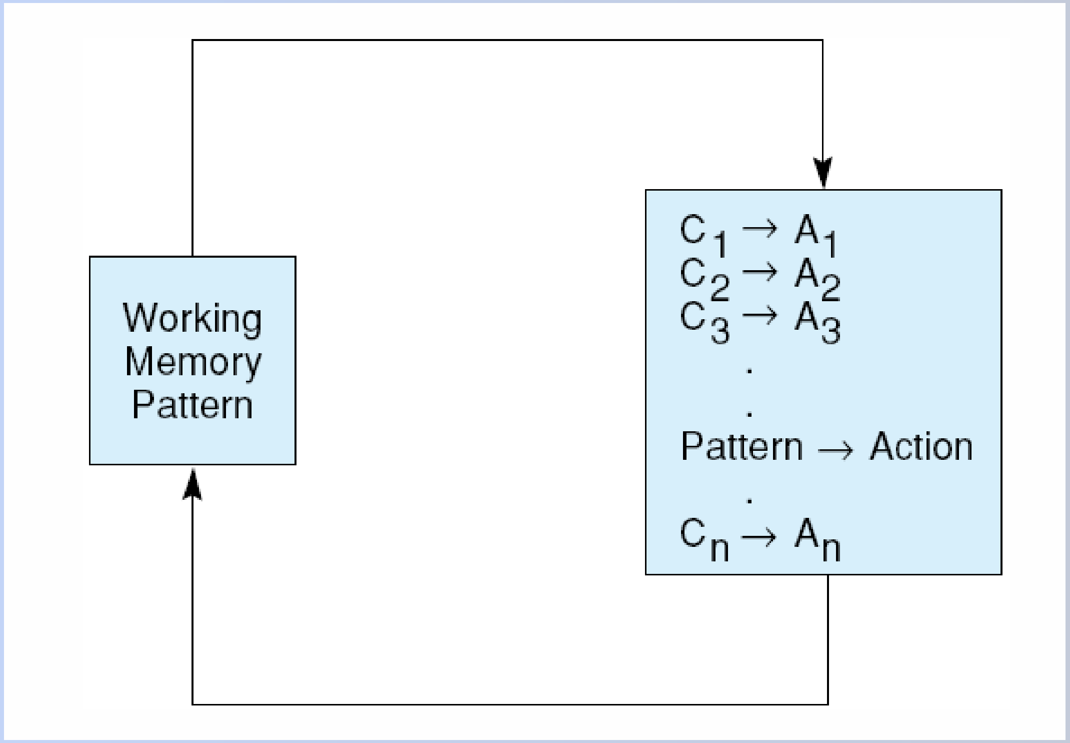
Production Systems

- A production system is
  - a set of rules (if-then or condition-action statements)
  - working memory
    - the current state of the problem solving, which includes new pieces of information created by previously applied rules
  - inference engine (the author calls this a “recognize-act” cycle)
    - forward-chaining, backward-chaining, a combination, or some other form of reasoning such as a sponsor-selector, or agenda-driven scheduler
  - conflict resolution strategy
    - when it comes to selecting a rule, there may be several applicable rules, which one should we select? the choice may be based on a conflict resolution strategy such as “first rule”, “most specific rule”, “most salient rule”, “rule with most actions”, “random”, etc

Production System Cycle

- Select a rule whose left hand side matches a pattern in working memory
- Fire the right hand side
  - this usually manipulates working memory (removing item(s), modifying item(s), adding item(s)



We need an algorithm to match conditions to working memory

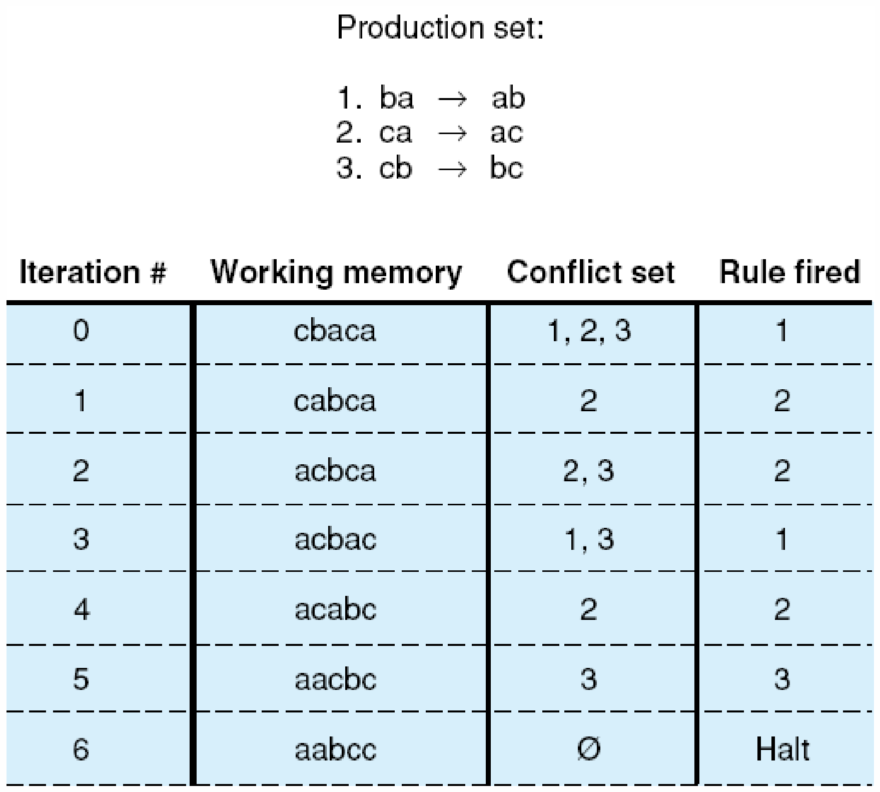
As with predicate calculus, we may not have an exact match for instance, if working memory stores
   dog(fido).
this does not match dog(X)

Conditions will often have multiple parts and-ed or or-ed together

Chaining

- The idea behind a production system's reasoning is that rules will describe steps in the problem solving space where a rule might
  - be an operation in a game like a chess move
  - translate a piece of input data into an intermediate conclusion
  - piece together several intermediate conclusions into a specific conclusion
  - translate a goal into substeps
- So a solution using a production system is a collection of rules that are chained together
  - forward chaining – reasoning from data to conclusions where working memory is sought for conditions that match the left-hand side of the given rules
  - backward chaining – reasoning from goals to operations where an initial goal is unfolded into the steps needed to solve that goal, that is, the process is one of subgoaling

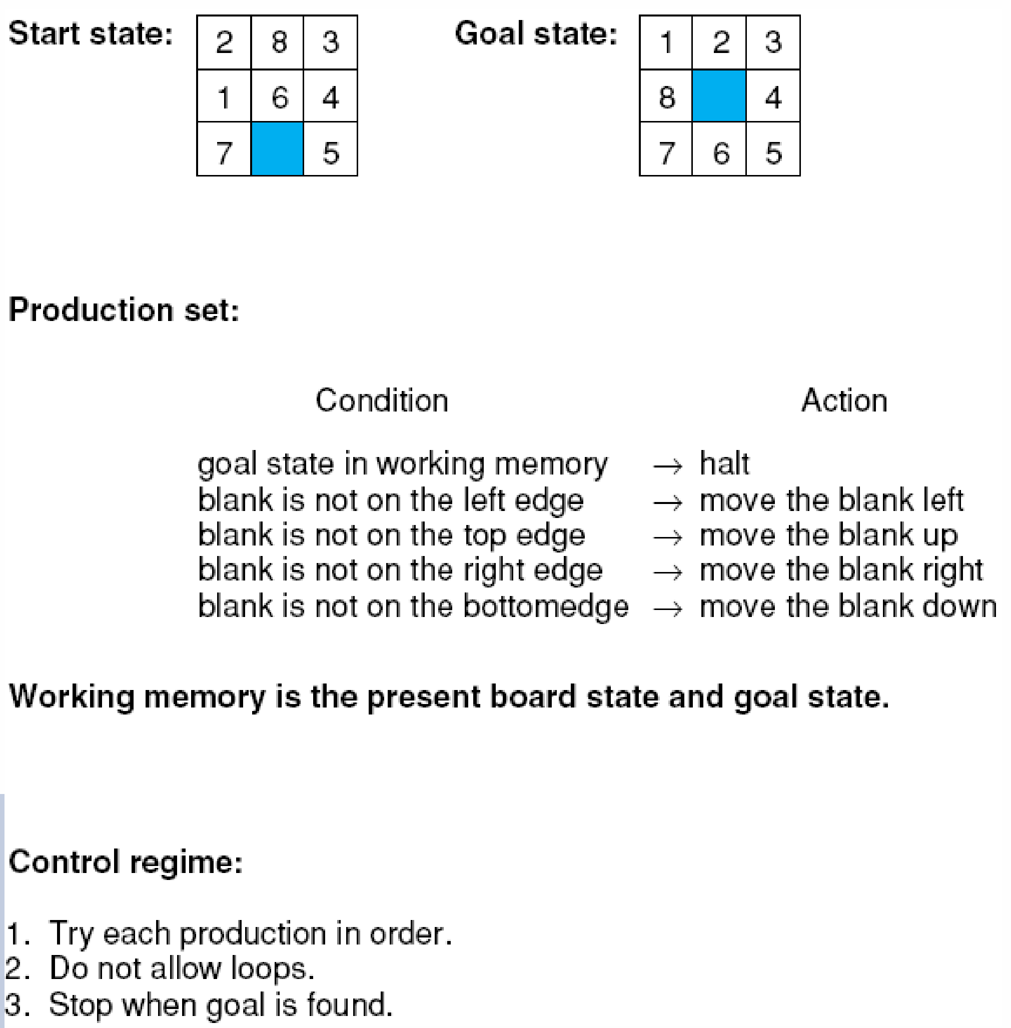
Two Example Production Systems



Production set:

1. ba → ab
2. ca → ac
3. cb → bc

Iteration #	Working memory	Conflict set	Rule fired
0	cbaca	1, 2, 3	1
1	cabca	2	2
2	acbca	2, 3	2
3	acbac	1, 3	1
4	acabc	2	2
5	aacbc	3	3
6	aabcc	Ø	Halt



Start state:

2	8	3
1	6	4
7		5

Goal state:

1	2	3
8		4
7	6	5

Production set:

Condition		Action
goal state in working memory	→	halt
blank is not on the left edge	→	move the blank left
blank is not on the top edge	→	move the blank up
blank is not on the right edge	→	move the blank right
blank is not on the bottomedge	→	move the blank down

Working memory is the present board state and goal state.

Control regime:

1. Try each production in order.
2. Do not allow loops.
3. Stop when goal is found.

Forward Chaining Example

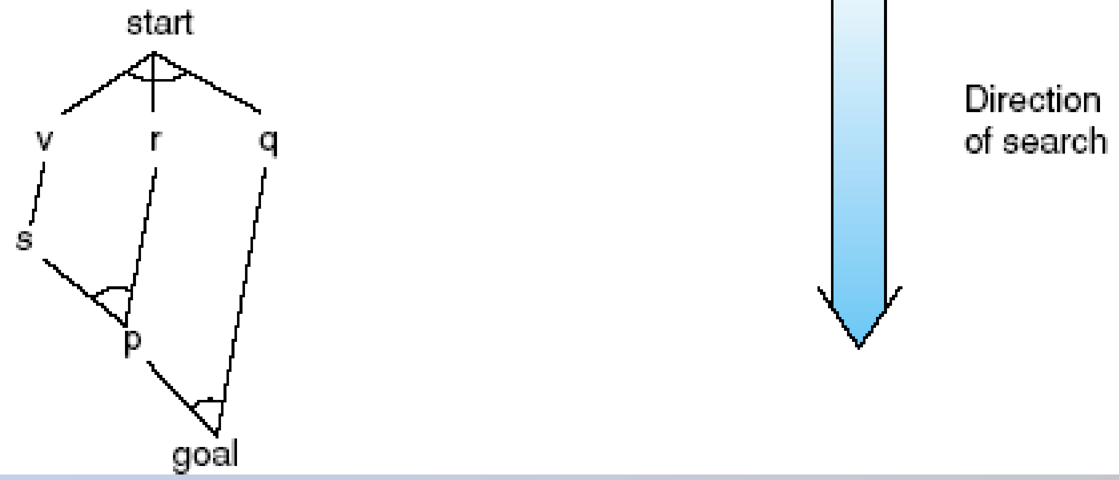
Production set:

1. $p \wedge q \rightarrow goal$
2. $r \wedge s \rightarrow p$
3. $w \wedge r \rightarrow q$
4. $t \wedge u \rightarrow q$
5. $v \rightarrow s$
6. $start \rightarrow v \wedge r \wedge q$

Trace of execution:

Iteration #	Working memory	Conflict set	Rule fired
0	start	6	6
1	start, v, r, q	6, 5	5
2	start, v, r, q, s	6, 5, 2	2
3	start, v, r, q, s, p	6, 5, 2, 1	1
4	start, v, r, q, s, p, goal	6, 5, 2, 1	halt

Space searched by execution:



Backward Chaining Example

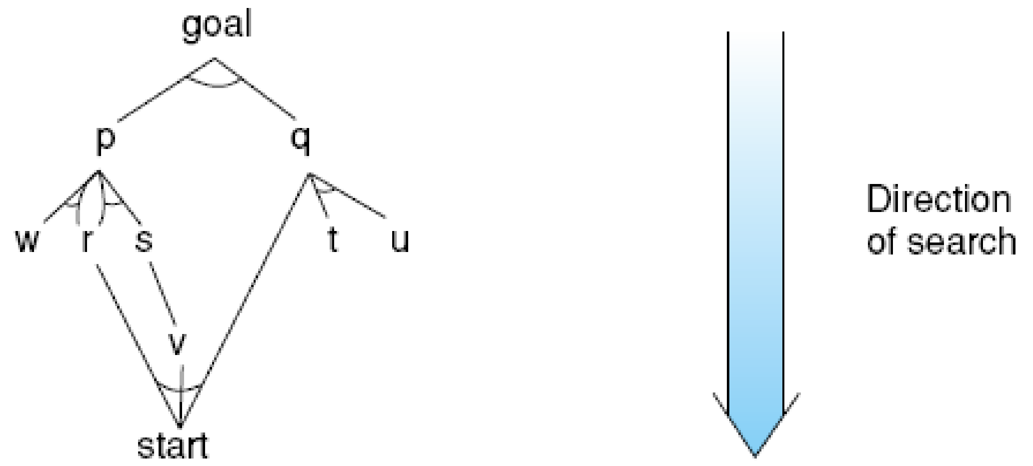
Production set:

1. $p \wedge q \rightarrow goal$
2. $r \wedge s \rightarrow p$
3. $w \wedge r \rightarrow p$
4. $t \wedge u \rightarrow q$
5. $v \rightarrow s$
6. $start \rightarrow v \wedge r \wedge q$

Trace of execution:

Iteration #	Working memory	Conflict set	Rule fired
0	goal	1	1
1	goal, p, q	1, 2, 3, 4	2
2	goal, p, q, r, s	1, 2, 3, 4, 5	3
3	goal, p, q, r, s, w	1, 2, 3, 4, 5	4
4	goal, p, q, r, s, w, t, u	1, 2, 3, 4, 5	5
5	goal, p, q, r, s, w, t, u, v	1, 2, 3, 4, 5, 6	6
6	goal, p, q, r, s, w, t, u, v, start	1, 2, 3, 4, 5, 6	halt

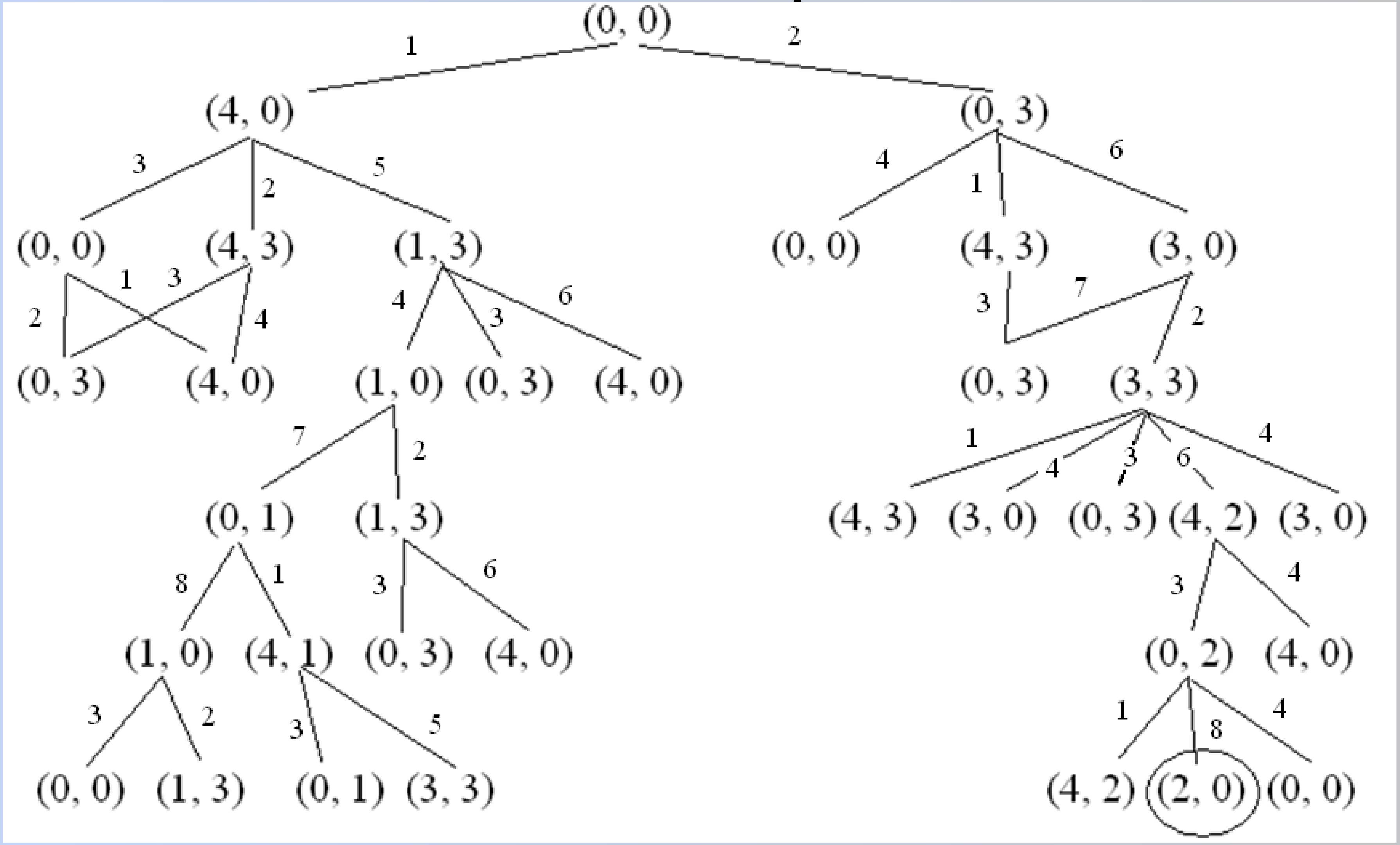
Space searched by execution:



Example System: Water Jugs

- Problem: given a 4-gallon jug (X) and a 3-gallon jug (Y), fill X with exactly 2 gallons of water
  - assume an infinite amount of water is available
- Rules/operators
  - 1. If X = 0 then X = 4 (fill X)
  - 2. If Y = 0 then Y = 3 (fill Y)
  - 3. If X > 0 then X = 0 (empty X)
  - 4. If Y > 0 then Y = 0 (empty Y)
  - 5. If X + Y >= 3 and X > 0 then X = X – (3 – y) and Y = 3 (fill Y from X)
  - 6. If X + Y >= 4 and Y > 0 then X = 4 and Y = Y – (4 – X) (fill X from Y)
  - 7. If X + Y <= 3 and X > 0 then X = 0 and Y = X + Y (empty X into Y)
  - 8. If X + Y <= 4 and Y > 0 then X = X + Y and Y = 0 (empty Y into X)
    - rule numbers used on the next slide

Solution Space



Note: the solution space does not show cycles (for instance, the second (4, 0) on the left does not have a subtree underneath it, we assume we will not continue from that point because (4, 0) is in the closed list

Eliza

The program would generate an English response/question based on a group of patterns

- if the user sentence matched a pattern, this pattern would be used to generate the next sentence/question

- Eliza algorithm
  - repeat
    - input a sentence
    - match a rule in the Eliza knowledge-base
      - attempt to perform pattern match (see next slide)
      - attempt to perform segment match (see two slides)
    - if rule found, select a response randomly (some patterns have multiple responses)
    - fill in variables, substitute values
  - until user quits

Eliza Rules

```
(defparameter *eliza-rules*
 '((((?* ?x) hello (?* ?y))
    (How do you do.  Please state your problem.))
   (((?* ?x) I want (?* ?y))
    (What would it mean if you got ?y)
    (Why do you want ?y) (Suppose you got ?y soon))
   (((?* ?x) if (?* ?y))
    (Do you really think its likely that ?y)
    (Do you wish that ?y)
    (What do you think about ?y) (Really-- if ?y))
   (((?* ?x) no (?* ?y))
    (Why not?) (You are being a bit negative)
    (Are you saying "NO" just to be negative?))
   (((?* ?x) I was (?* ?y))
    (Were you really?)
    (Perhaps I already knew you were ?y)
    (Why do you tell me you were ?y now?))
   (((?* ?x) I feel (?* ?y))
    (Do you often feel ?y ?))
   (((?* ?x) I felt (?* ?y))
    (What other feelings do you have?))))
```

If the input contains
  something hello *something*,
Eliza responds with
    “How do you do.”

If the input contains
   something if *something else*
Eliza responds with
     Do you really think its likely
     that *something else*?
or with
    Do you wish that
     something else?

Eliza Pattern Matching

pat → var	match any one expression to a variable
constant	or to a constant (see below)
segment-pat	match against a sequence
single-pat	match against one expression
(pat . pat)	match the first and the rest of a list
single-pat →	
(?is var predicate)	test predicate on one expression
(?or pat1 pat2 …)	match on any of the patterns
(?and pat1 pat2 …)	match on every of the expressions
(?not pat)	match if expression does not match
segment-pat →	
((?* var) …)	match on zero or more expressions
((?+ var) …)	match on one or more expressions
((?? var) …)	match zero or one expression
((?if expr) …)	test if expression is true
var → ?chars	variables of the form ?name
constant → atom	constants are atoms (symbols, #, chars)

Conflict Resolution Strategies

- What happens when more than one rule matches?
  - a conflict resolution strategy dictates how to select from between multiple matching rules
- Simple conflict resolution strategies include
  - random
  - first match
  - most/least recently matched rule
  - rule which has matched for the longest/shortest number of cycles (refractoriness)
  - most salient rule (each rule is given a salience before you run the production system)
- More complex resolution strategies might
  - select the rule with the most/least number of conditions (specificity/generality)
  - or most/least number of actions (biggest/smallest change to the state)
    - some additional examples are covered in the notes

Advantages of Production Systems

- Separation of knowledge and control
  - these systems contain two (or more) distinct forms of knowledge – the knowledge base (rules) and the inference engine
    - this makes it easy to update/change knowledge and debug the system
- Easy to map knowledge into rule format
  - a lot of expert knowledge is already in this form, in fact, a production system is a plausible model for human problem solving
- Rules can be grouped into logical sets
  - promotes modularity and allows meta-knowledge to select which set of rules to concentrate on
- Easy to enhance a system to explain its behavior
  - just add code to output the selected rules to demonstrate the chain of logic that led to the conclusion(s)
- Easy to construct shell languages

Disadvantages of Production Systems

- There is a lack of focus
  - that is, the system will just continue to fire rules
  - a human problem solver might discover a pattern early on so that the expert refocuses attention on some specific set of rules, this is not typically done in production systems
- Computationally complex
  - as with nearly any AI system, search means inefficiency, a production system is just another means of searching through a space of knowledge
- Difficult to debug
  - odd behavior begins to occur with thousands of rules and its hard to figure out why or just what rules should be changed
  - changing a rule may cause problems with other rules – for instance, am I altering a rule that will be needed by another rule? am I altering a rule so that it overlaps with another rule?