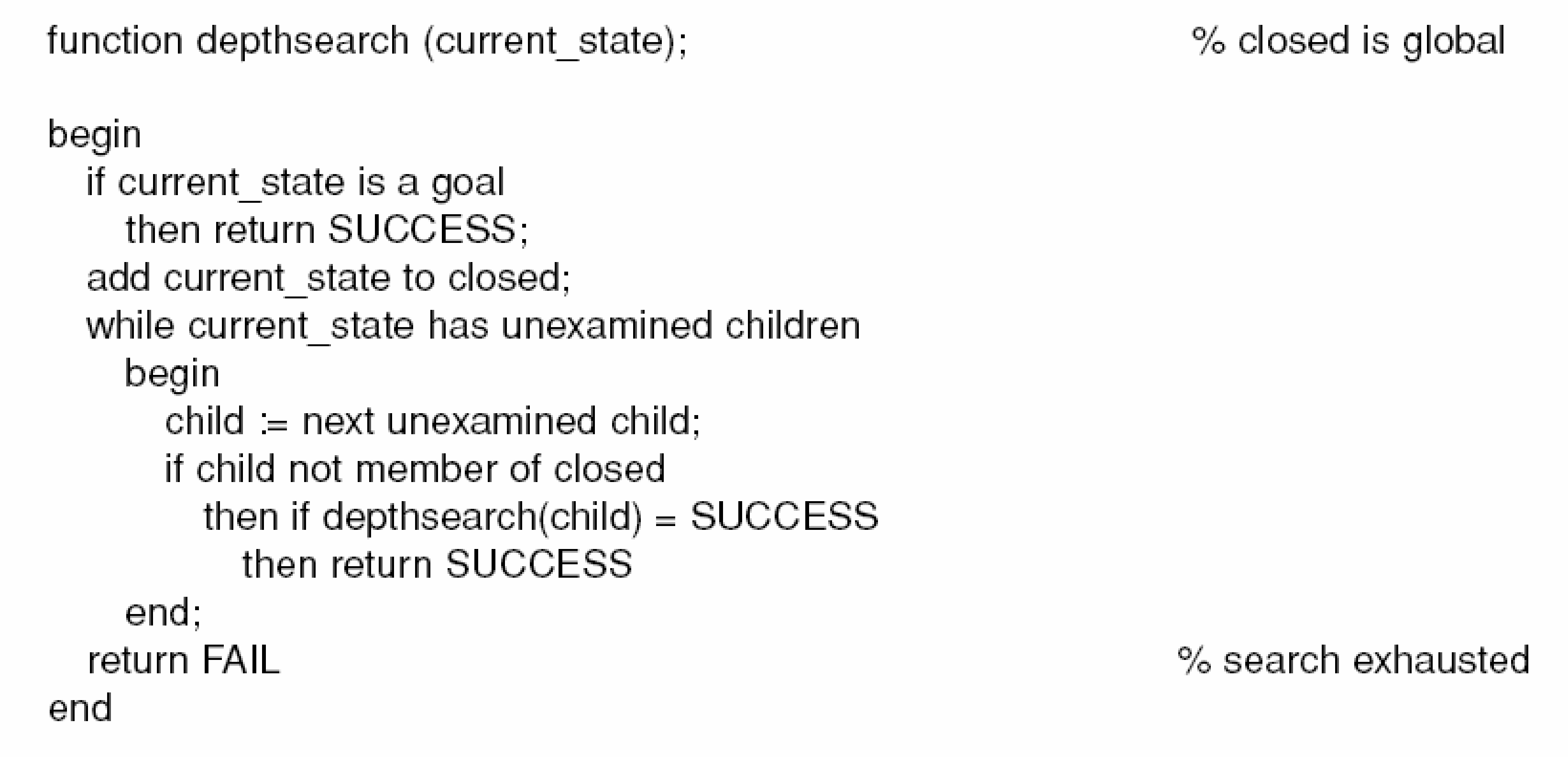
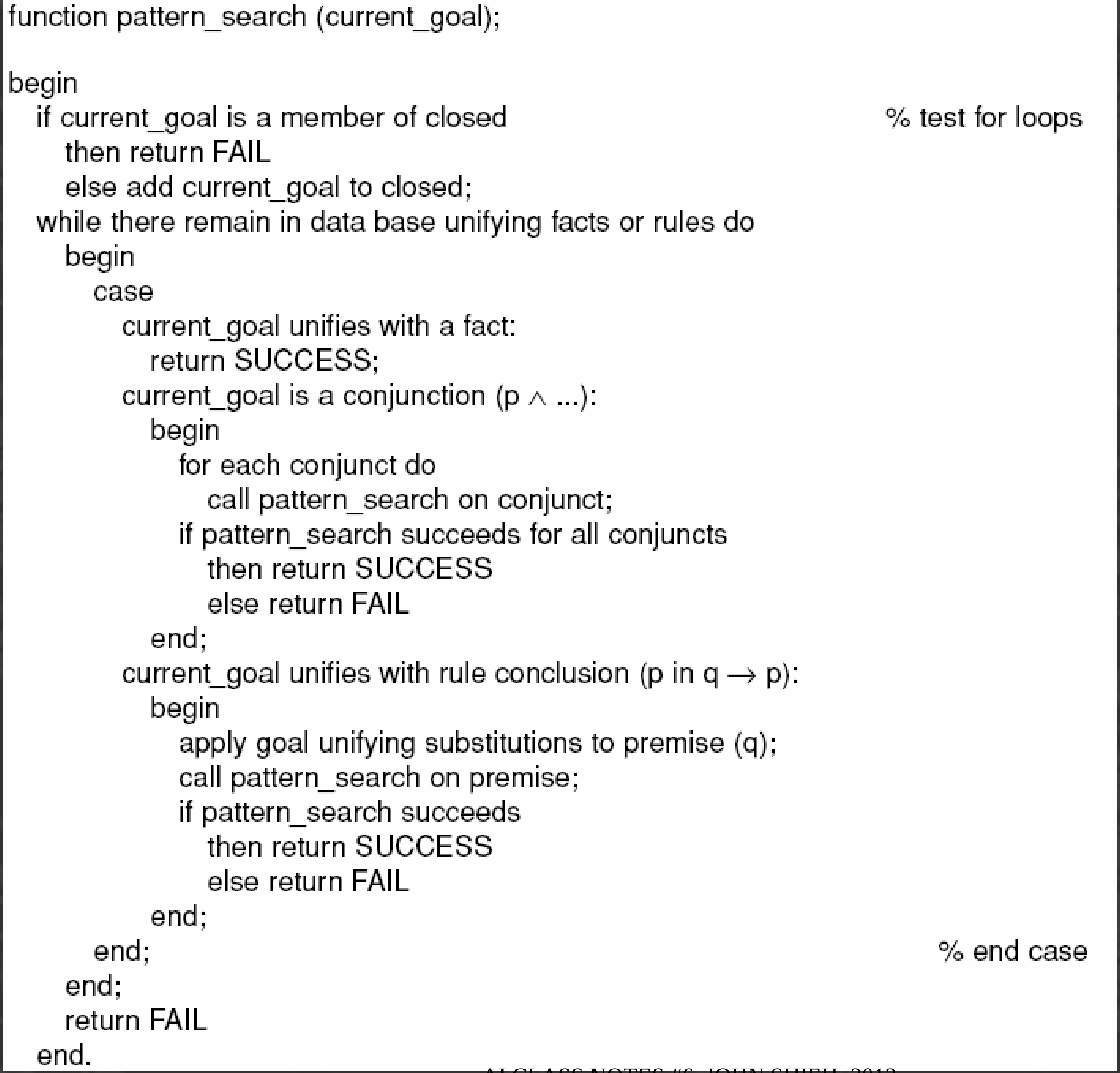
Production Systems



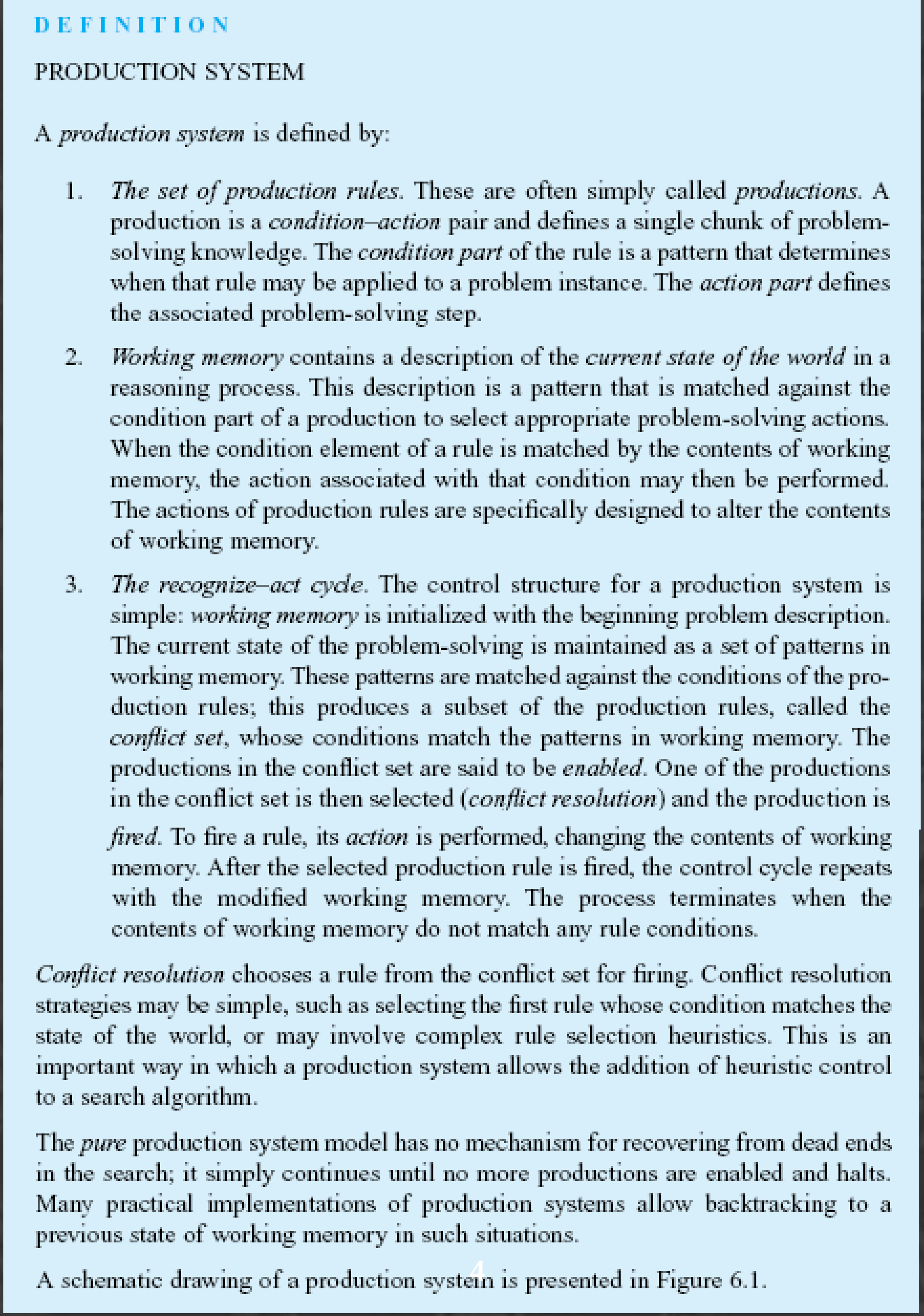
```
function depthsearch (current_state);                         % closed is global

begin
   if current_state is a goal
      then return SUCCESS;
   add current_state to closed;
   while current_state has unexamined children
      begin
         child := next unexamined child;
         if child not member of closed
            then if depthsearch(child) = SUCCESS
               then return SUCCESS
      end;
   return FAIL                                               % search exhausted
end
```



```
function pattern_search (current_goal);

begin
  if current_goal is a member of closed                          % test for loops
    then return FAIL
    else add current_goal to closed;
  while there remain in data base unifying facts or rules do
    begin
      case
        current_goal unifies with a fact:
          return SUCCESS;
        current_goal is a conjunction ($p \wedge ...$):
          begin
            for each conjunct do
              call pattern_search on conjunct;
            if pattern_search succeeds for all conjuncts
              then return SUCCESS
              else return FAIL
          end;
        current_goal unifies with rule conclusion ($p$ in $q \rightarrow p$):
          begin
            apply goal unifying substitutions to premise (q);
            call pattern_search on premise;
            if pattern_search succeeds
              then return SUCCESS
              else return FAIL
          end;
      end;                                                        % end case
    end;
    return FAIL
  end.
```



DEFINITION

PRODUCTION SYSTEM

A *production system* is defined by:

1. *The set of production rules.* These are often simply called *productions.* A production is a *condition–action* pair and defines a single chunk of problem-solving knowledge. The *condition part* of the rule is a pattern that determines when that rule may be applied to a problem instance. The *action part* defines the associated problem-solving step.

2. *Working memory* contains a description of the *current state of the world* in a reasoning process. This description is a pattern that is matched against the condition part of a production to select appropriate problem-solving actions. When the condition element of a rule is matched by the contents of working memory, the action associated with that condition may then be performed. The actions of production rules are specifically designed to alter the contents of working memory.

3. *The recognize–act cycle.* The control structure for a production system is simple: *working memory* is initialized with the beginning problem description. The current state of the problem-solving is maintained as a set of patterns in working memory. These patterns are matched against the conditions of the production rules; this produces a subset of the production rules, called the *conflict set,* whose conditions match the patterns in working memory. The productions in the conflict set are said to be *enabled.* One of the productions in the conflict set is then selected (*conflict resolution*) and the production is *fired.* To fire a rule, its *action* is performed, changing the contents of working memory. After the selected production rule is fired, the control cycle repeats with the modified working memory. The process terminates when the contents of working memory do not match any rule conditions.

Conflict resolution chooses a rule from the conflict set for firing. Conflict resolution strategies may be simple, such as selecting the first rule whose condition matches the state of the world, or may involve complex rule selection heuristics. This is an important way in which a production system allows the addition of heuristic control to a search algorithm.

The *pure* production system model has no mechanism for recovering from dead ends in the search; it simply continues until no more productions are enabled and halts. Many practical implementations of production systems allow backtracking to a previous state of working memory in such situations.

A schematic drawing of a production system is presented in Figure 6.1.

Fig 6.1 A production system. Control loops until working memory pattern no longer matches the conditions of any productions.

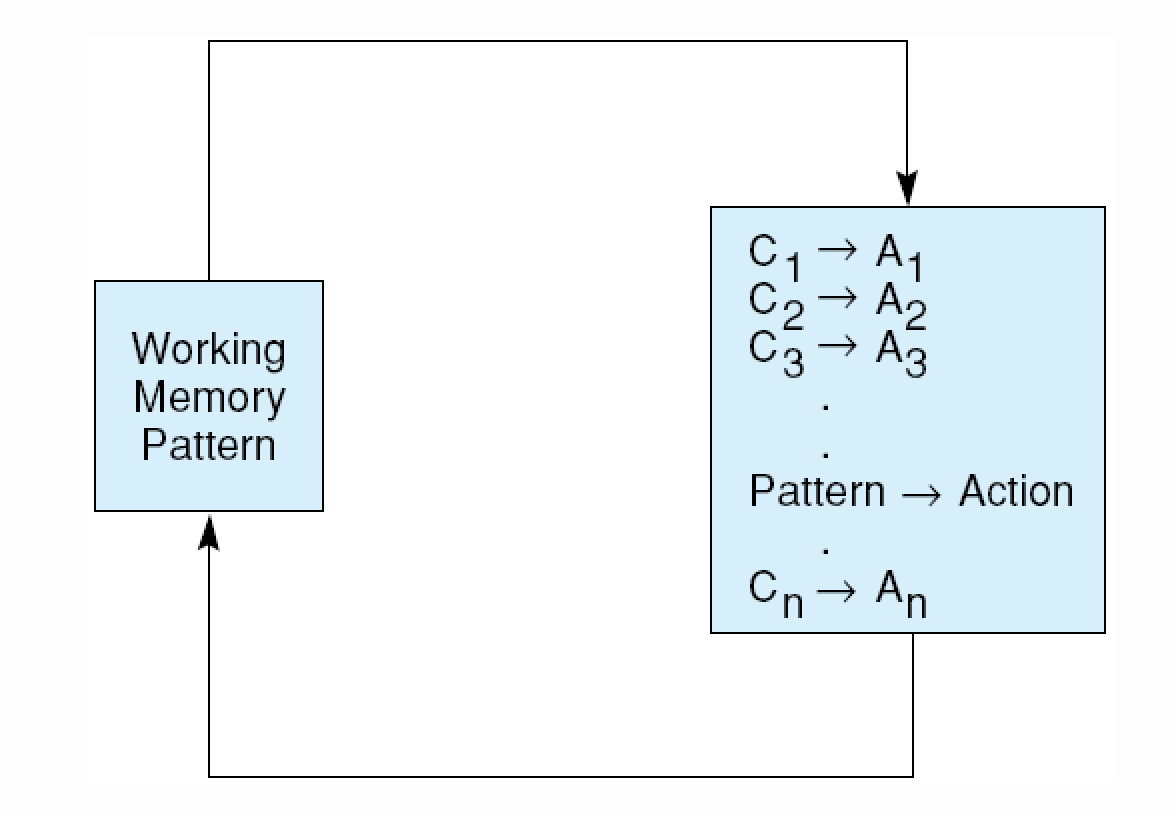
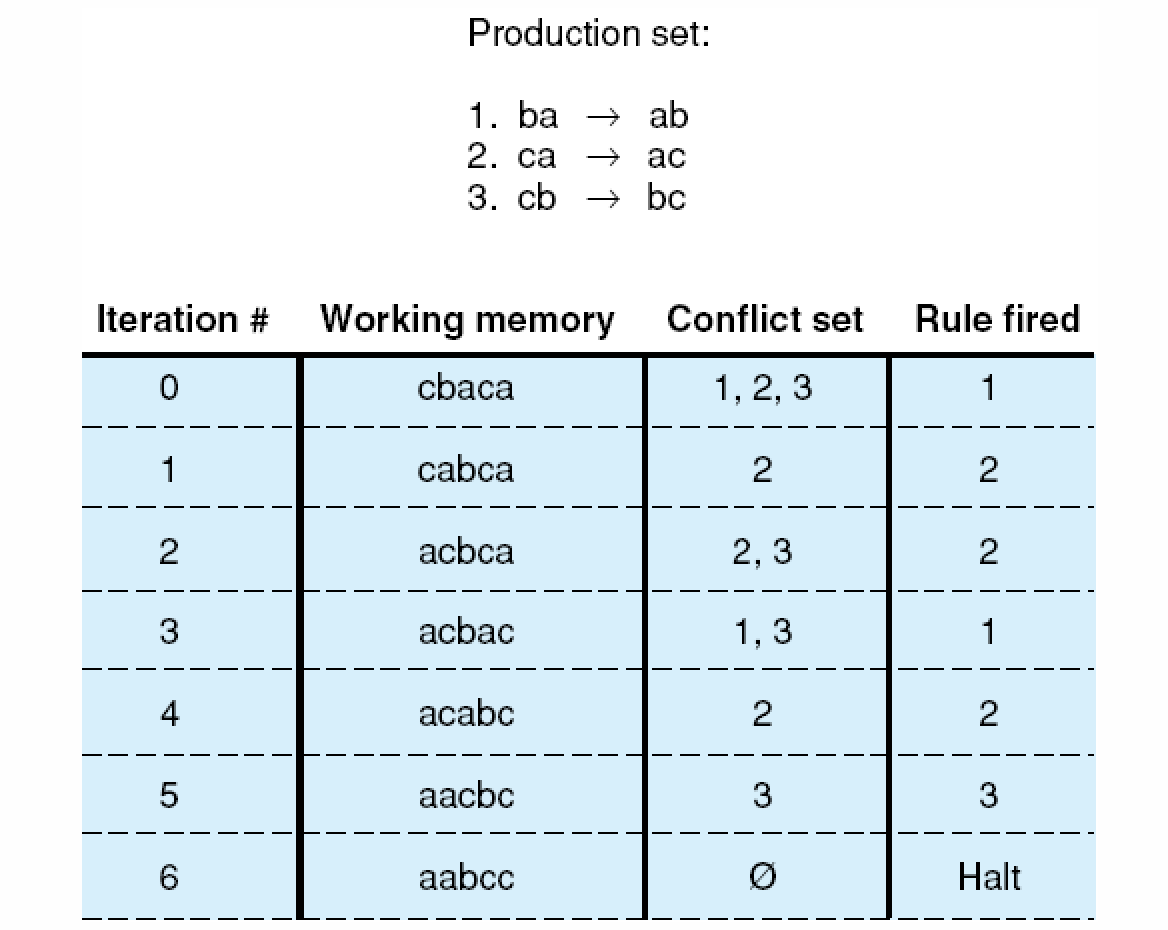


Fig 6.2 Trace of a simple production system.

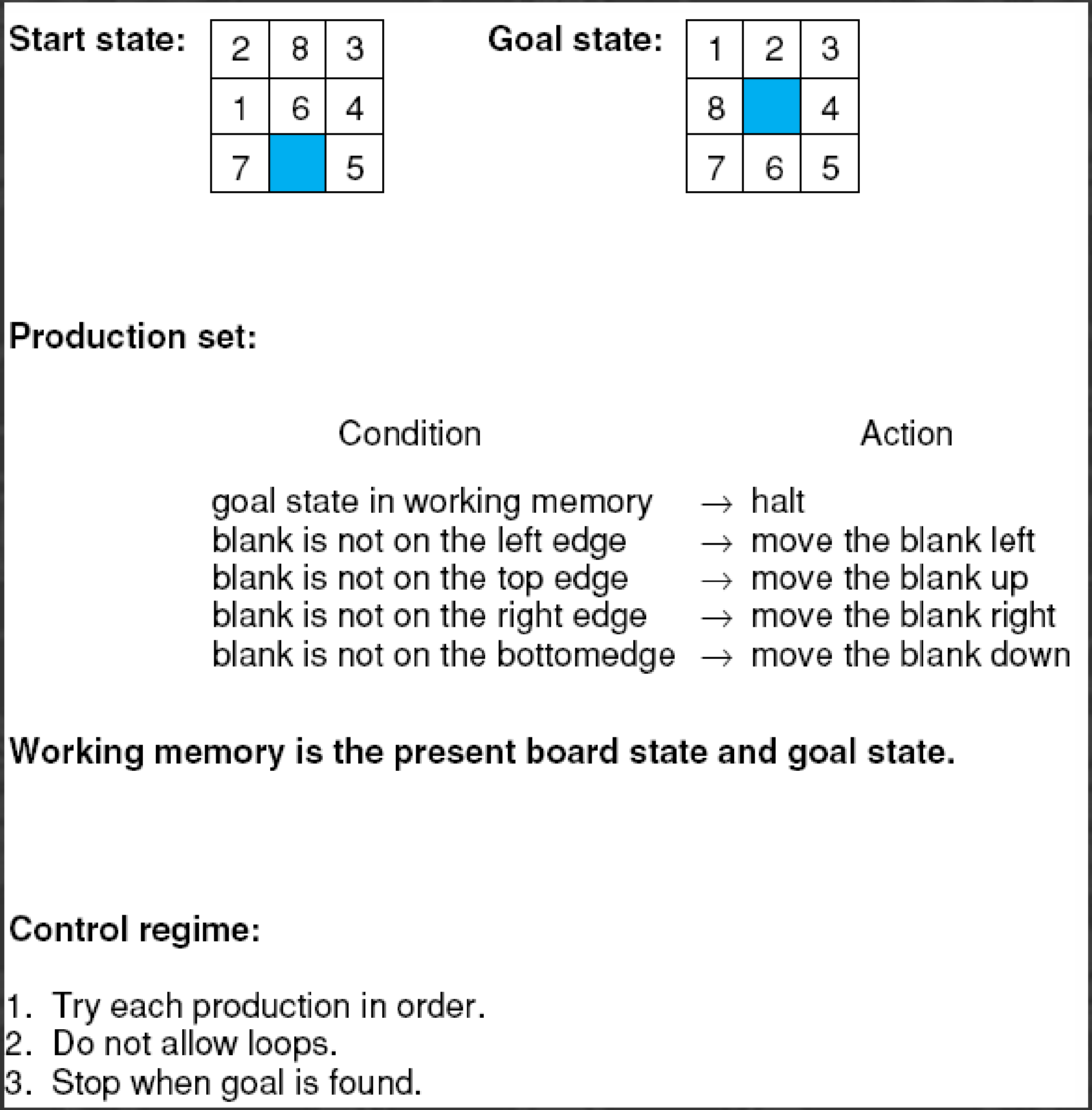


Production set:

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

Fig 6.3 The 8-puzzle as a production system.



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.

Fig 6.4 The 8-puzzle searched by a production system with loop detection and depth-bound , from Nilsson (1971).

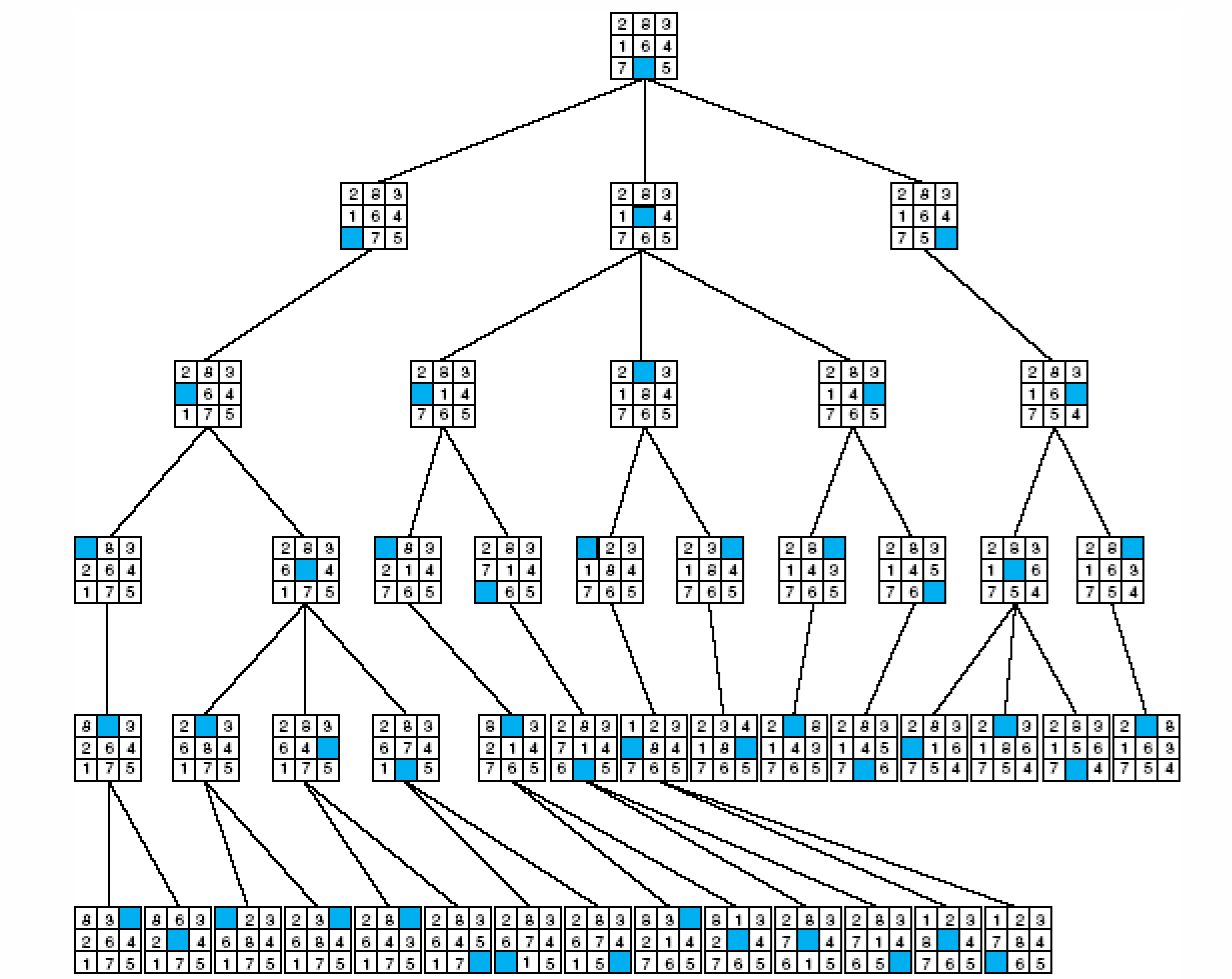


Fig 6.5   Legal moves of a chess knight.

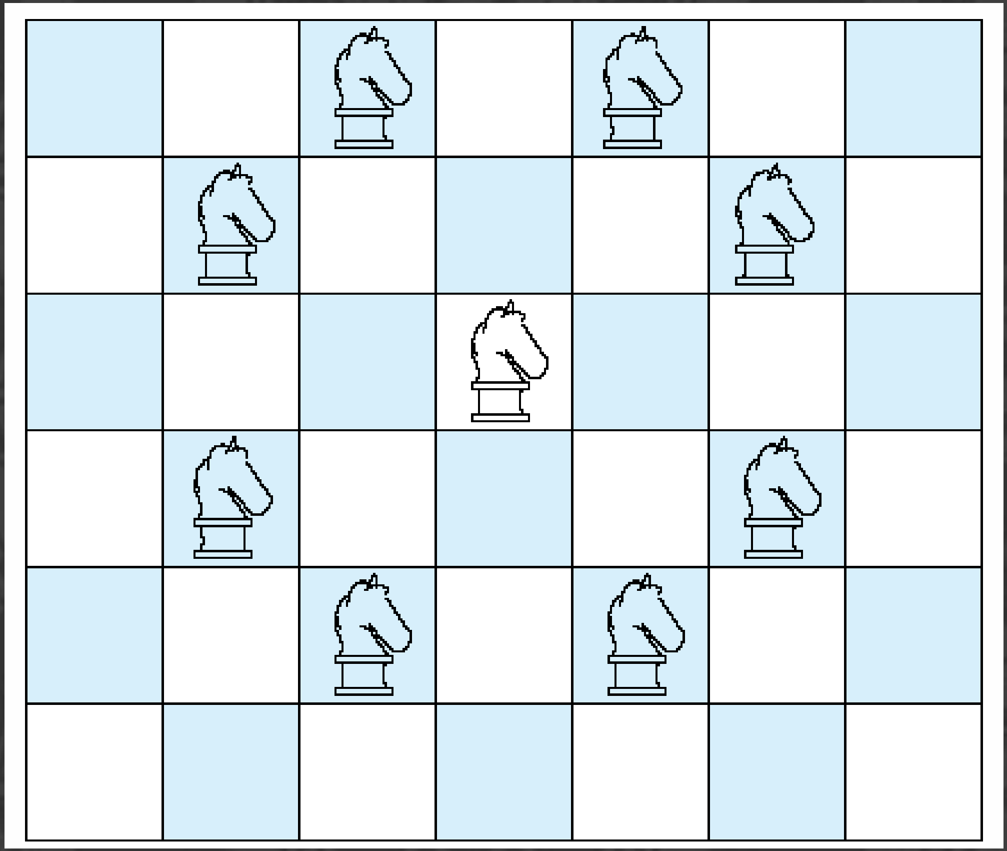
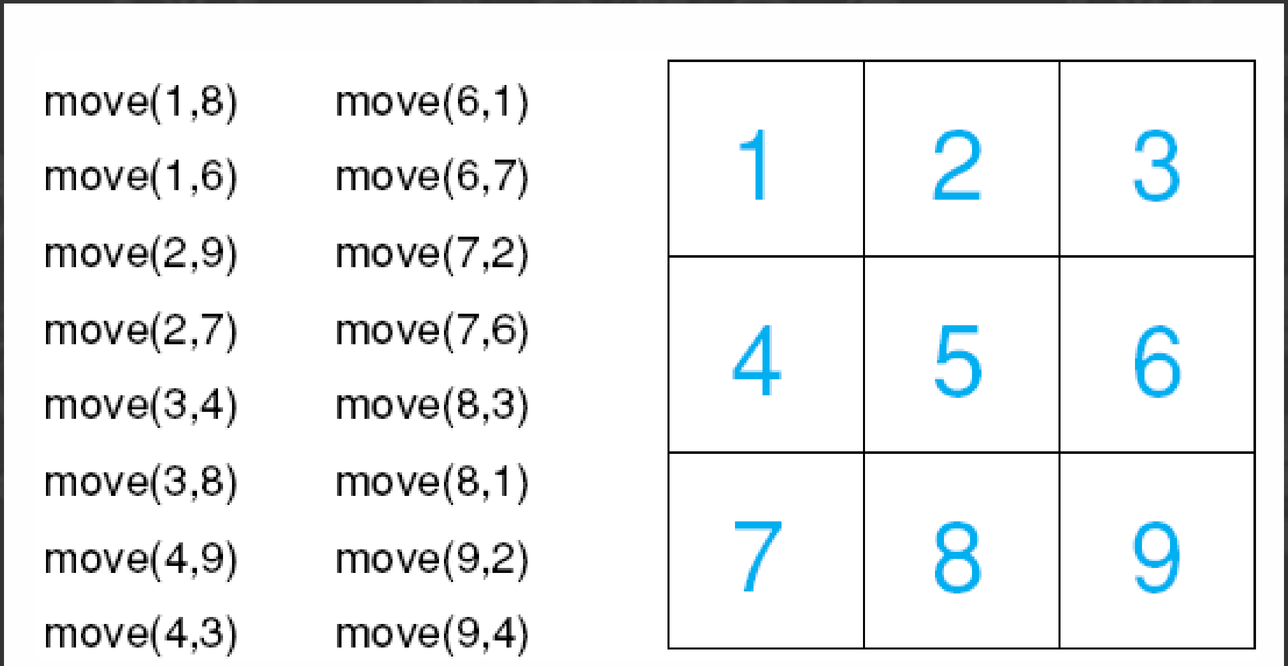


Fig 6.6 a 3 x 3 chessboard with move rules for the simplified knight tour problem.

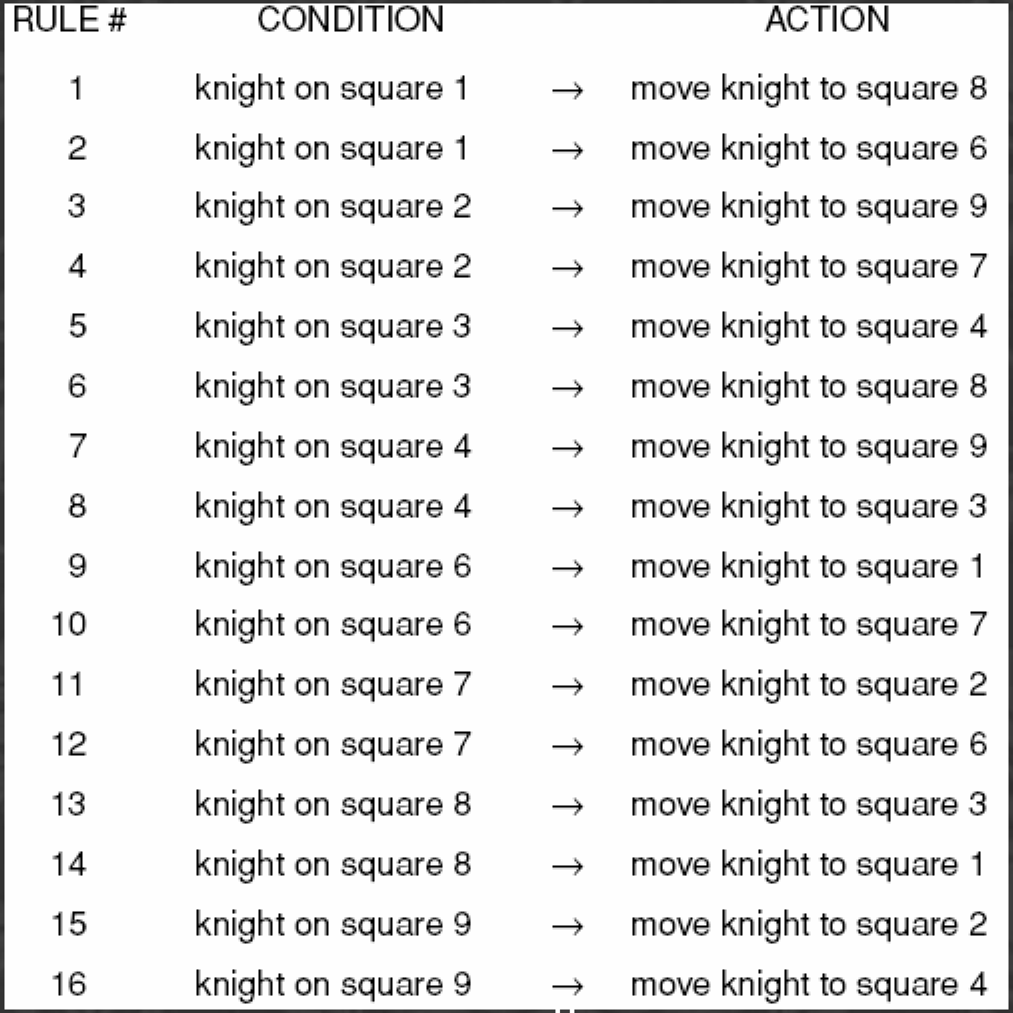


move(1,8)
move(1,6)
move(2,9)
move(2,7)
move(3,4)
move(3,8)
move(4,9)
move(4,3)

move(6,1)
move(6,7)
move(7,2)
move(7,6)
move(8,3)
move(8,1)
move(9,2)
move(9,4)

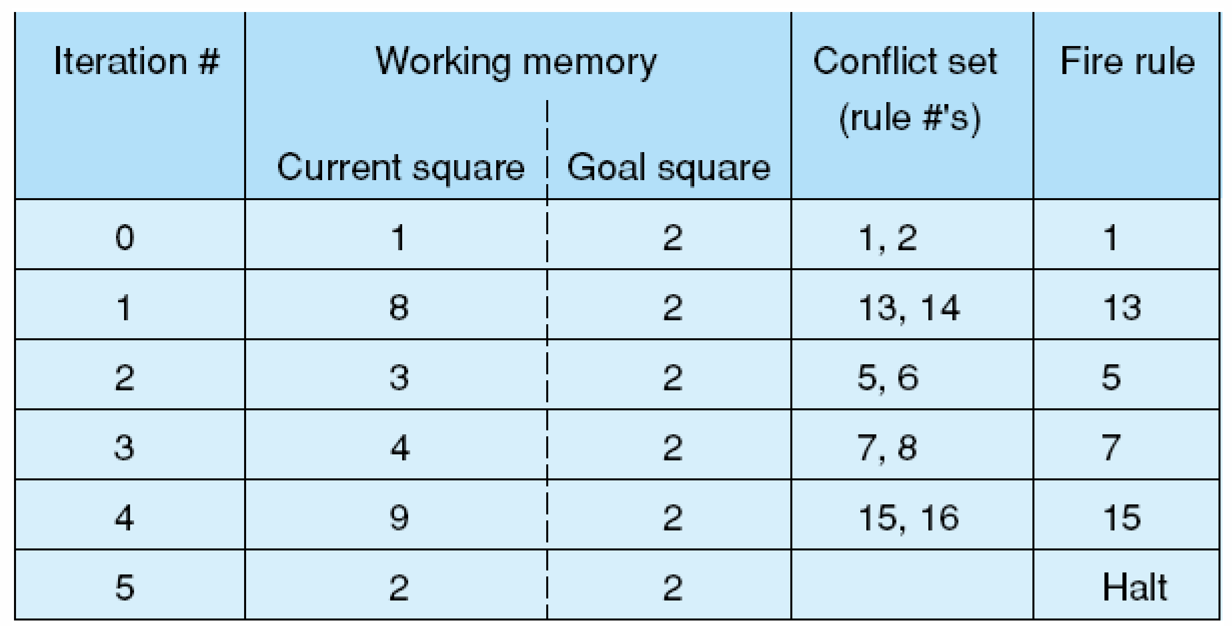
1	2	3
4	5	6
7	8	9

Table 6.1 Production rules for the 3 x 3 knight problem.



RULE #	CONDITION		ACTION
1	knight on square 1	→	move knight to square 8
2	knight on square 1	→	move knight to square 6
3	knight on square 2	→	move knight to square 9
4	knight on square 2	→	move knight to square 7
5	knight on square 3	→	move knight to square 4
6	knight on square 3	→	move knight to square 8
7	knight on square 4	→	move knight to square 9
8	knight on square 4	→	move knight to square 3
9	knight on square 6	→	move knight to square 1
10	knight on square 6	→	move knight to square 7
11	knight on square 7	→	move knight to square 2
12	knight on square 7	→	move knight to square 6
13	knight on square 8	→	move knight to square 3
14	knight on square 8	→	move knight to square 1
15	knight on square 9	→	move knight to square 2
16	knight on square 9	→	move knight to square 4

Fig6.7 A production system solution to the 3 x 3 knight’s tour problem.



Iteration #	Working memory		Conflict set (rule #'s)	Fire rule
	Current square	Goal square		
0	1	2	1, 2	1
1	8	2	13, 14	13
2	3	2	5, 6	5
3	4	2	7, 8	7
4	9	2	15, 16	15
5	2	2		Halt

Fig 6.8 The recursive path algorithm as production system.

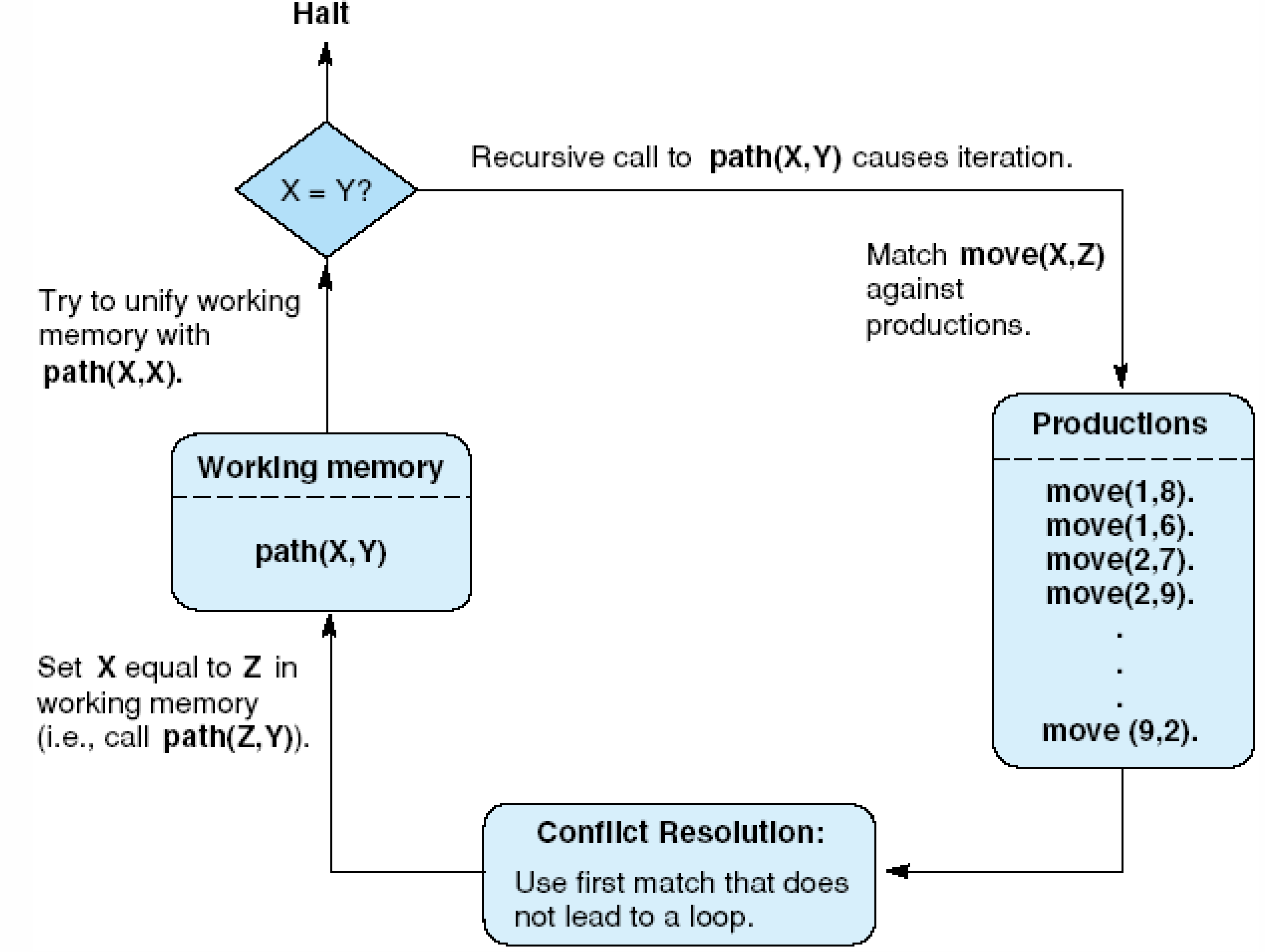
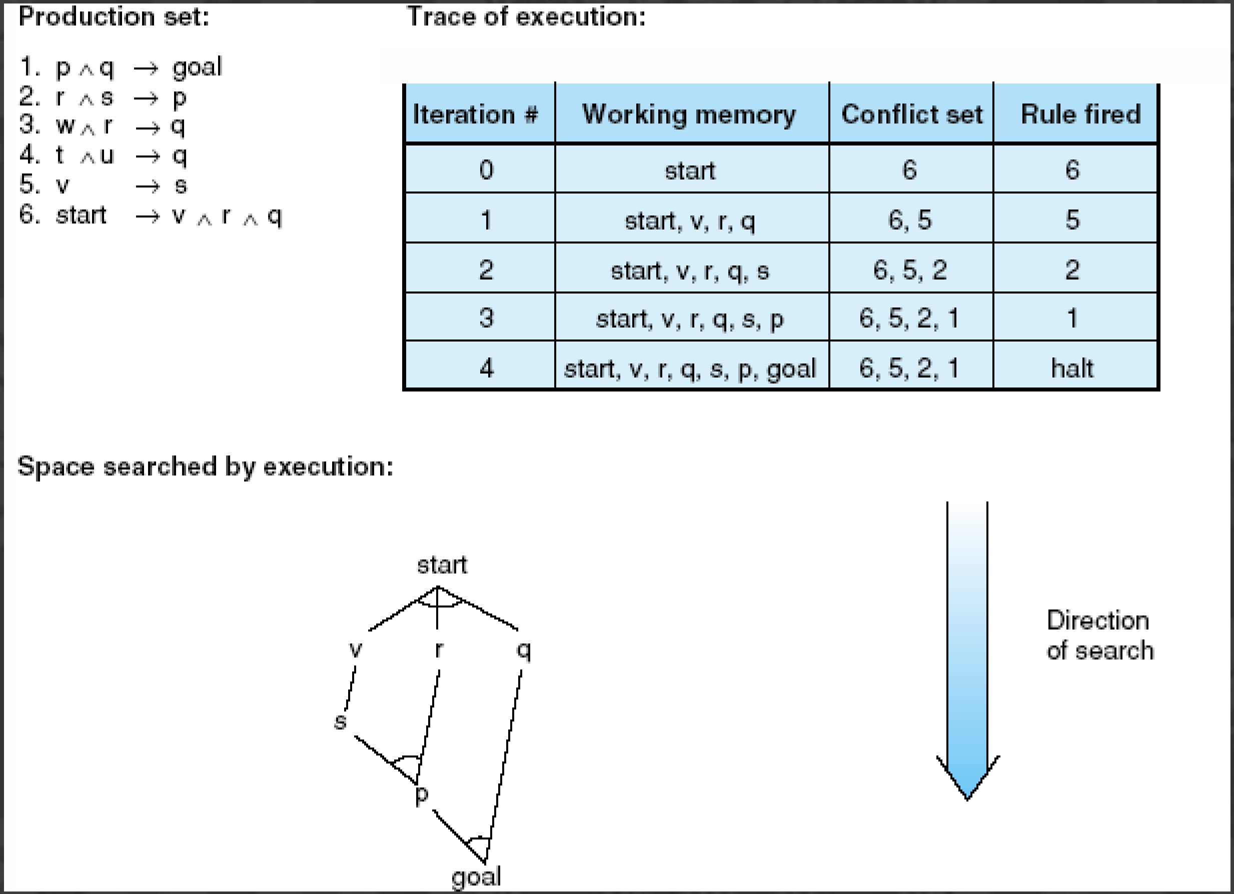

Halt
X = Y?
Recursive call to path(X,Y) causes iteration.
Match move(X,Z) against productions.
Try to unify working memory with path(X,X).
Working memory
path(X,Y)
Productions
move(1,8).
move(1,6).
move(2,7).
move(2,9).
.
.
.
move (9,2).
Set X equal to Z in working memory (i.e., call path(Z,Y)).
Conflict Resolution:
Use first match that does not lead to a loop.

Fig 6.9 Data-driven search in a production system.



Production set:

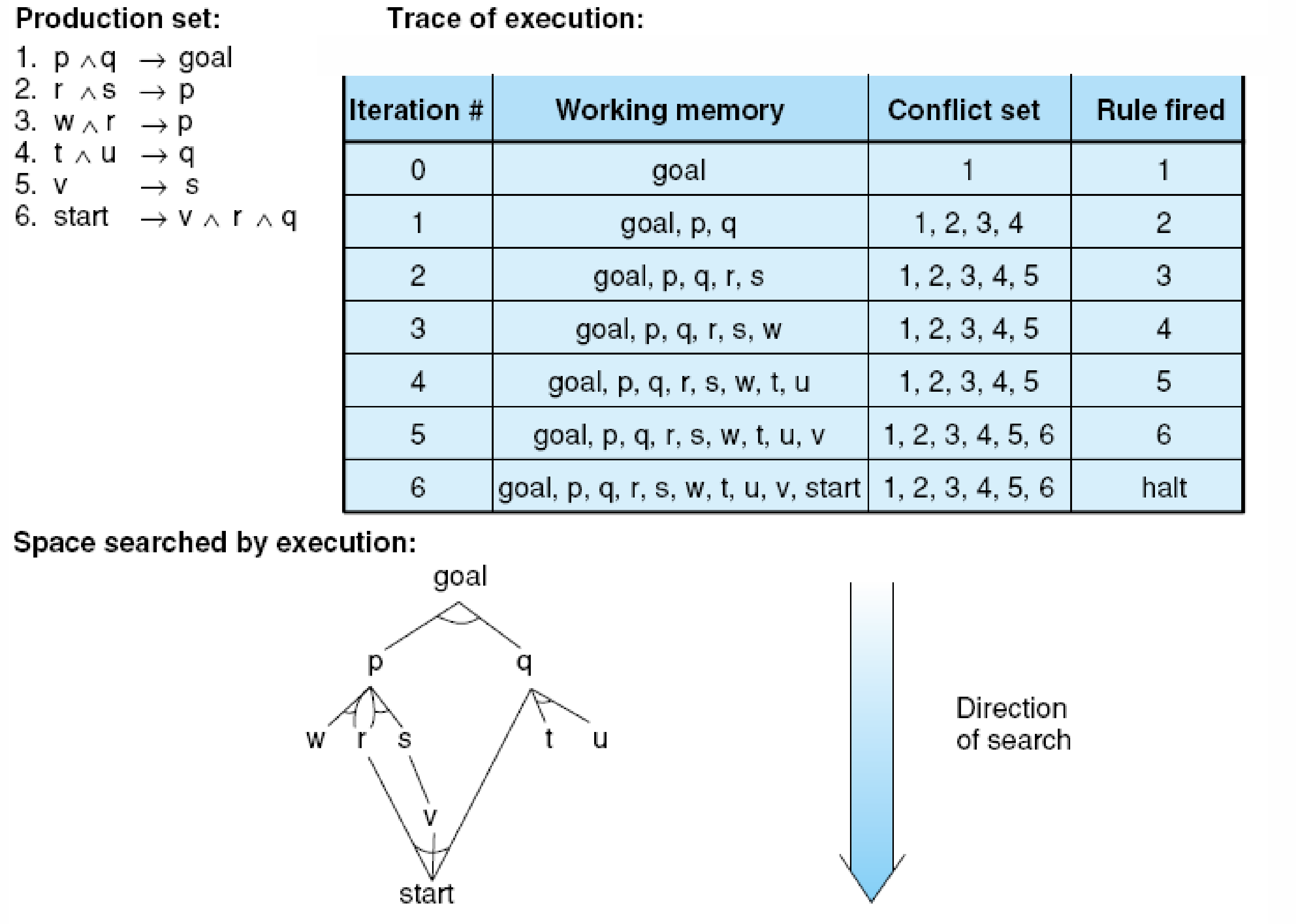
1. $p \wedge q \rightarrow goal$
2. $r \wedge s \rightarrow p$
3. $w \wedge r \rightarrow q$
4. $t \wedge u \rightarrow q$
5. $v \rightarrow s$
6. $start \rightarrow v \wedge r \wedge q$

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:

Fig 6.10 Goal-driven search in a production system.



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:

Fig 6.11 Bidirectional search missing in both directions, resulting in excessive search.

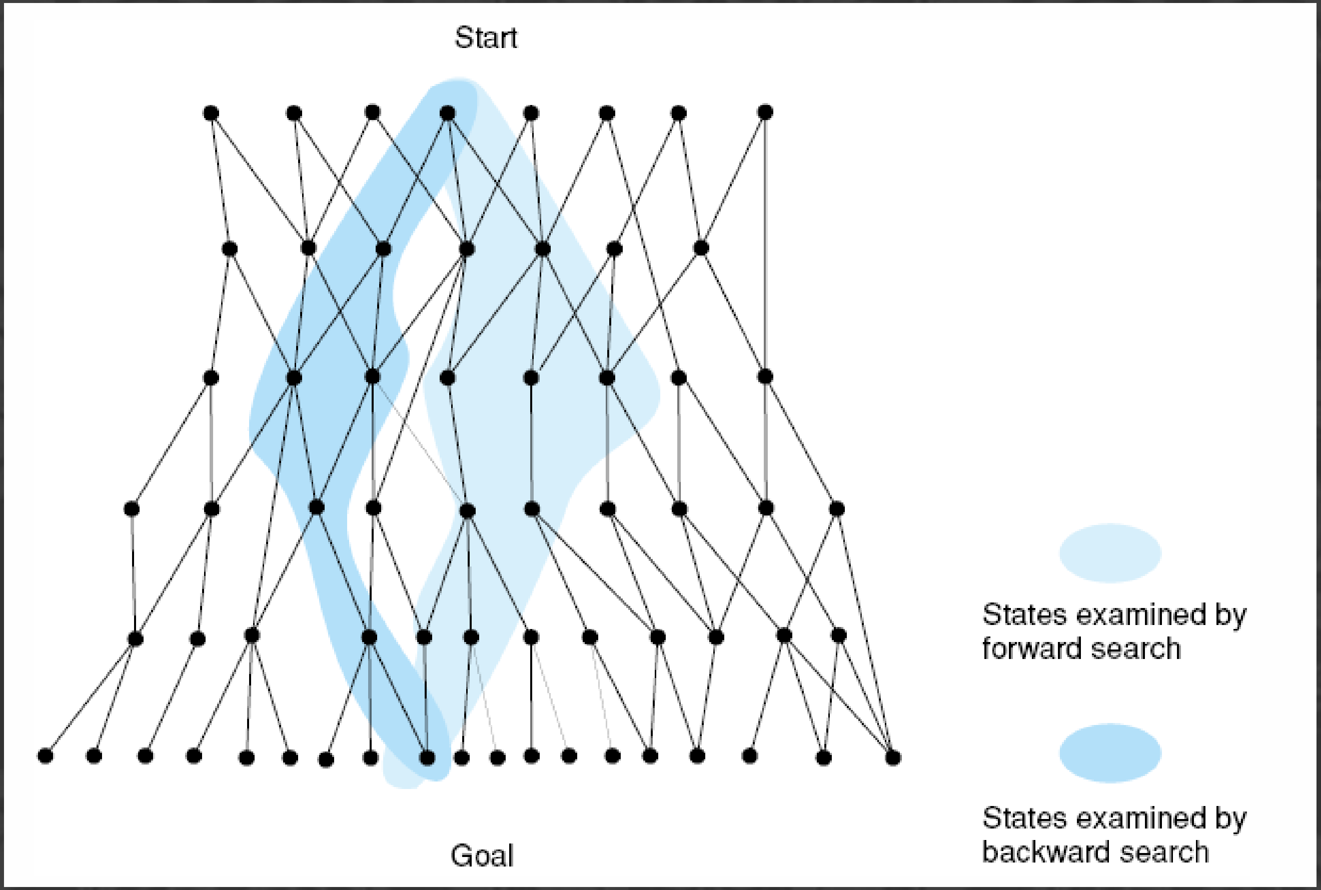
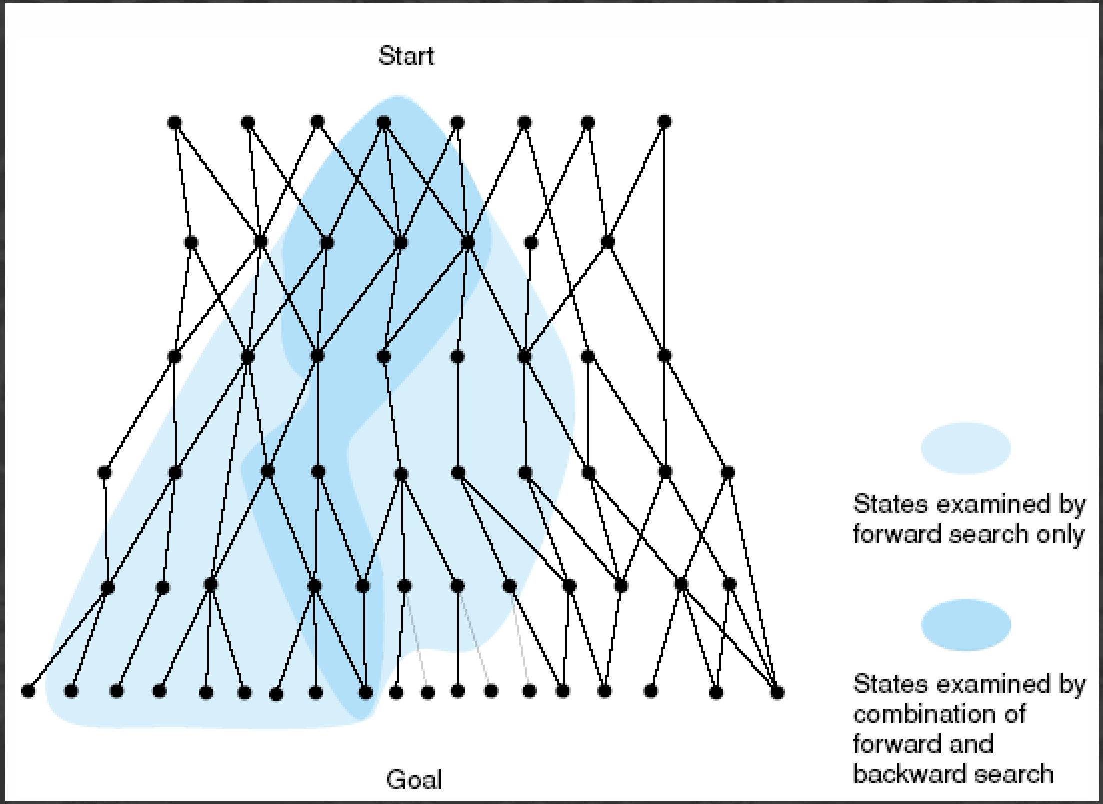

Start
States examined by forward search
States examined by backward search
Goal

Fig 6.12 Bidirectional search meeting in the middle, eliminating much of the space examined by unidirectional search.


Start
Goal
States examined by forward search only
States examined by combination of forward and backward search

Major advantages of production systems for artificial intelligence

Separation of Knowledge and Control

A Natural Mapping onto State Space Search

Modularity of Production Rules

Pattern-Directed Control

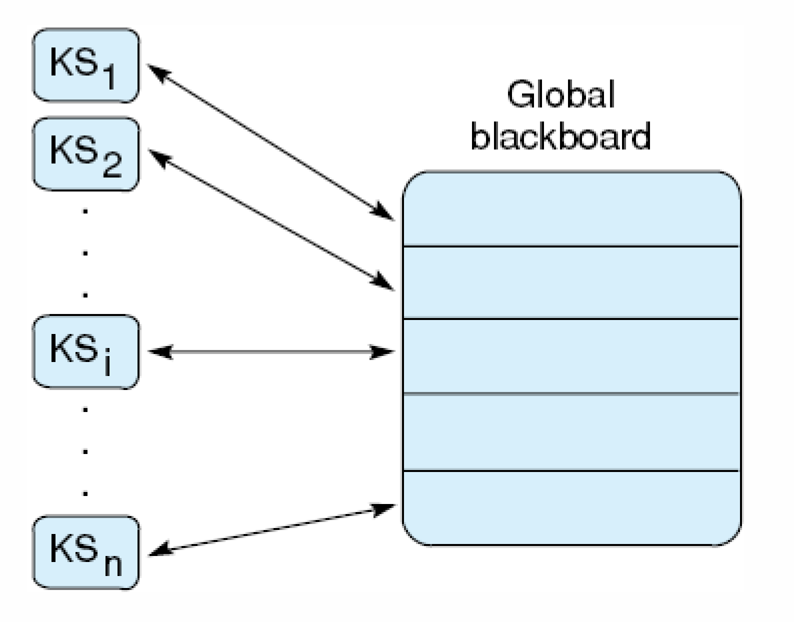
Opportunities for Heuristic Control of Search

Tracing and Explanation

Language Independence

A Plausible Model of Human Problem-Solving

Fig 6.13 Blackboard architecture


KS_1
KS_2
KS_i
KS_n
Global
blackboard