PRODUCTION RULES SYSTEMS

1. WHAT IS A PRODUCTION RULE SYSTEM

Production systems are systems based on a widely used knowledge representation technique, namely the notion of condition-action pairs called production rules.

A production system consists of three parts :

- A rule base coposed of a set of production rules.
- A special buffer-like data structure called the context.
- An interpreter, which controls the system's activity (control strategy).

1.1 Production rules

A production rule is a statement of the form

" IF this condition holds THEN tis action is appropriate"

antecedent	consequent
premise	conclusion

IF stoplight is red AND you have stopped THEN right turn OK

A production rule whose condition part is satisfied can fire, i.e can have its action part executed by the interpreter.

1.2 The Context

- Called the data, short term memory buffer, working memory.
- The left hand side of each production in the rule base represents a condition that must be present in the context data structure before the production can fire.
- The actionsof the production rules can change the context so that other rules will have their condition parts satisfied.
- The context shows the current situation at a given time.

1.3 The interpreter

Decide what to do next (which production to fire next).

The interpreter may operate as follows :

- Find all productions whose condition parts are true and make them applicable.
- If more than one production is applicable, deactivate any production whose action adds a duplicate symbol to the context data structure.

- Execute the action of only one applicable production. If no productions are applicable, then quit.

1.4Operation of Production Systems

Production systems operate in cycles. The three phases of each cycle are :

- Matching
- Conflict resolution
- Action

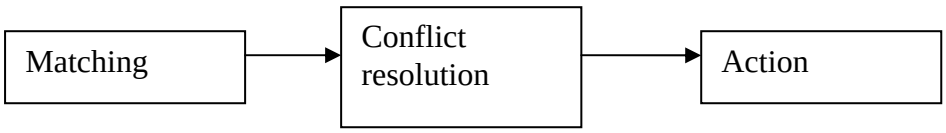


Fig 1. Production system cycle

1.5 Idea of an Expert System

A knowledge-base system is an AI program that incorporates knowledge, obtained from various sources, about a specialized subject area (domain).

An Expert System is a knowledge-base system in which the knowledge comes (almost) entirely from an (group of) expert. The user of an expert system enters facts and questions about this domain, and the program applies them to its knowledge base (KB) in order to produce output. This output is the information that would be provided by an expert consultant. The interaction between the user and the expert system is called a consultation.

An expert system shell is a software system (or tool) that contains an inference engine together with additional code that is designed around particular data structures and enables users easily to incorporate their own expert knowledge in any given domain and to draw on it later.

Production systems have been used as the backbone of expert AI systems.

Fig.2 shows the main components of a rule-based system.

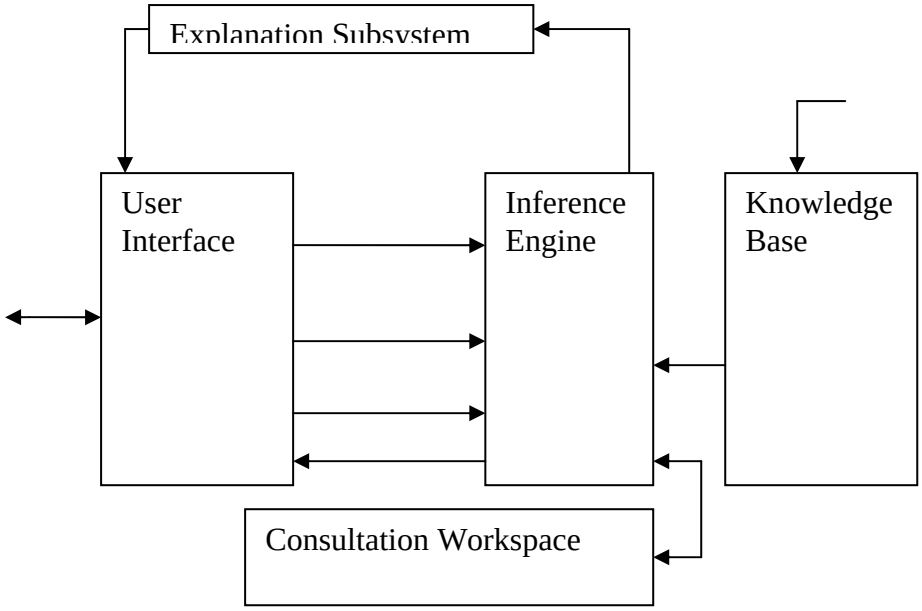


Fig. 2 A rule-base system

Two kinds of Popular Rule Based Systems

1. Synthesis Systems
   R1/XCON
   Tends to be forward chaining (data driven : looks at all facts before proceeding)
   Often make use of breadth first search

2. Analysis & Diagnostic
   Tends to use backward chaining (goal driven)
   MYCIN
   Often make use of depth first search

2. ADVANTAGES AND DISADVANTAGES OF PRODUCTION SYSTEMS

2.1 Advantages

- Modularity
  The ability to add, modify or delete any rule independent of the others. Rules communicate only by the means of the context data structure.

- Uniformity
  All information coded as production rules (can be made understood by another person)

- Naturalness
  Production rules frequently used by human experts to explain how they do their jobs.

2.2 Disadvantages

- Inefficiency
  Time to scan the rules
- Opacity
  Hard to follow the flow of control in problem solving (Non-determinism).
- Not all domain knowledge fits rule format. Algorithmic knowledge can not be expressed naturally (rules do not call each other – no production can be composed of sub-productions).
- Knowledge acquisition is time consuming. Expert consensus must exist. (Knowledge Acquisition bottleneck). XCON 2500 rules , 2 years to create the expert system.

Forward Chaining (Bottom-Up)

Do this until problem is solved or no antecedents match

1. Collect the rules whose antecedents are found in WM.
2. If more than one rule matches use conflict resolution strategy to eliminate all but one
3. Do actions indicated in by rule “fired”

Conflict Resolution Strategies

1. Level of Specificity
   Maximum Specificity (number of antecedents) match antecedents and choose the one with the most matches.
2. Choose a rule at random.
3. Physically order the rules. Assign priorities.
4. Recency Ordering for rules
5. Execution Time

Backward Chaining Algorithm (Top-Down)

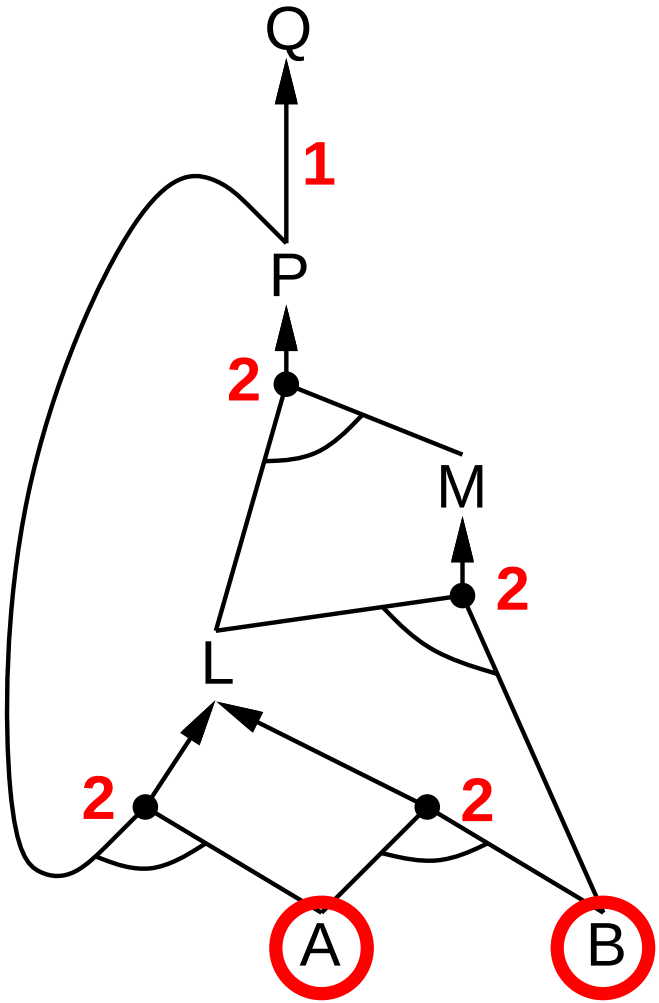
Given goal g

- find the set of rules that determine g
- if a set of rules does not equal empty set then
  loop
- choose rule R
- make R’s antecedent
  new goal

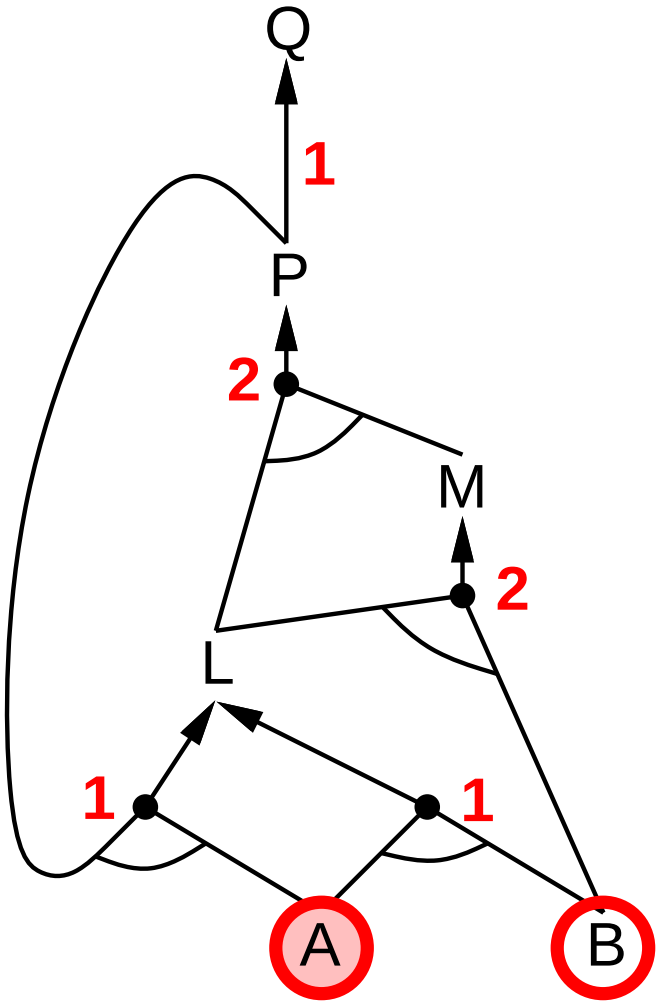
- if new goal is unknown then backchain (ng)
- else
  apply rule R
- until g is solved or S is equal to empty set
- else
  consult user

end

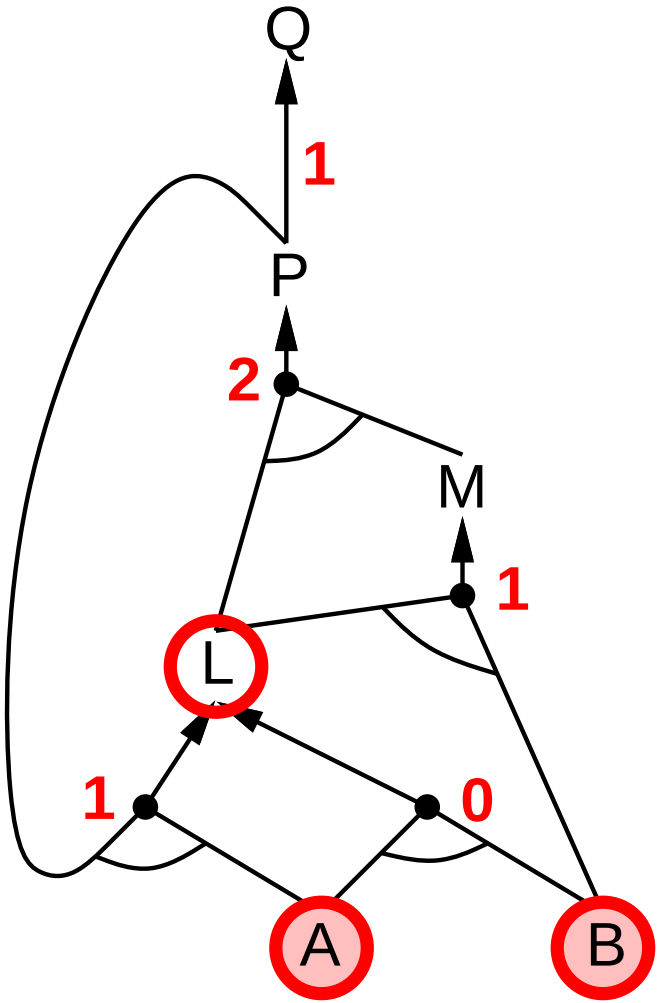
Forward chaining example



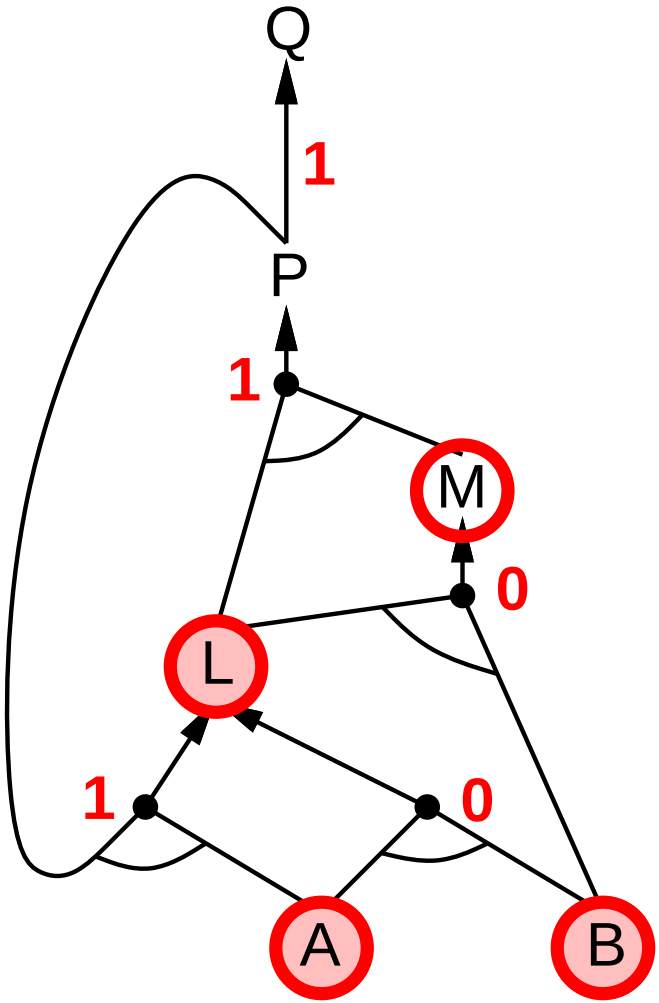
Forward chaining example



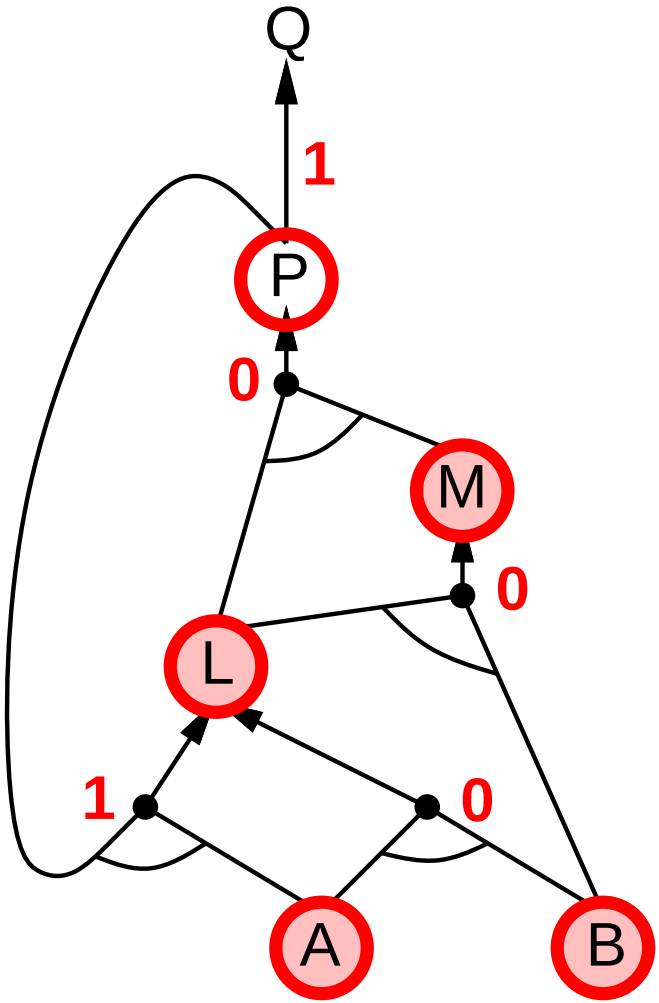
Forward chaining example



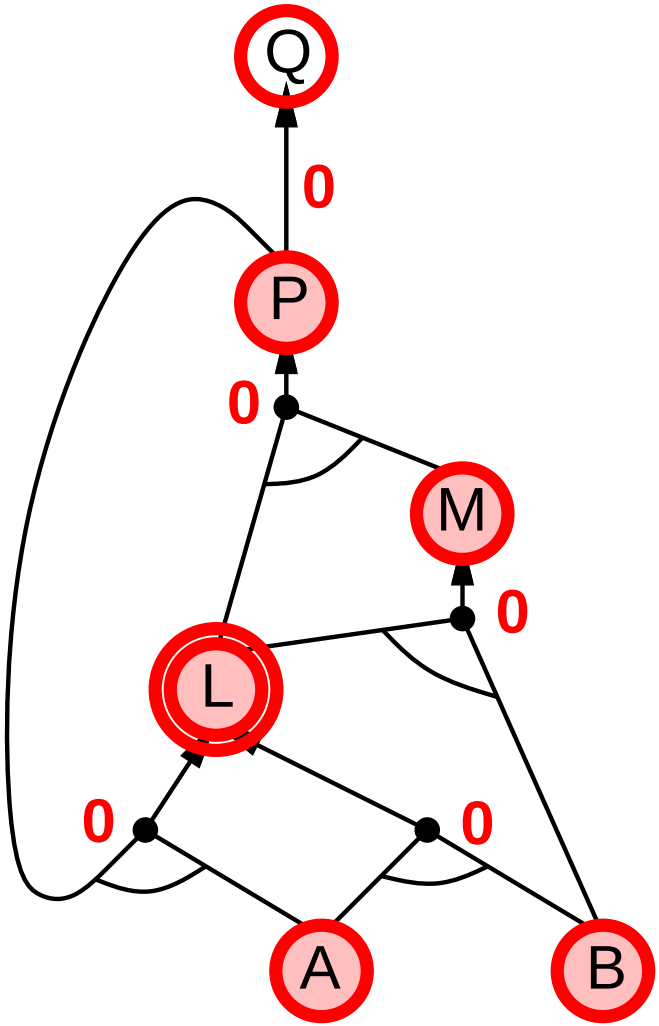
Forward chaining example



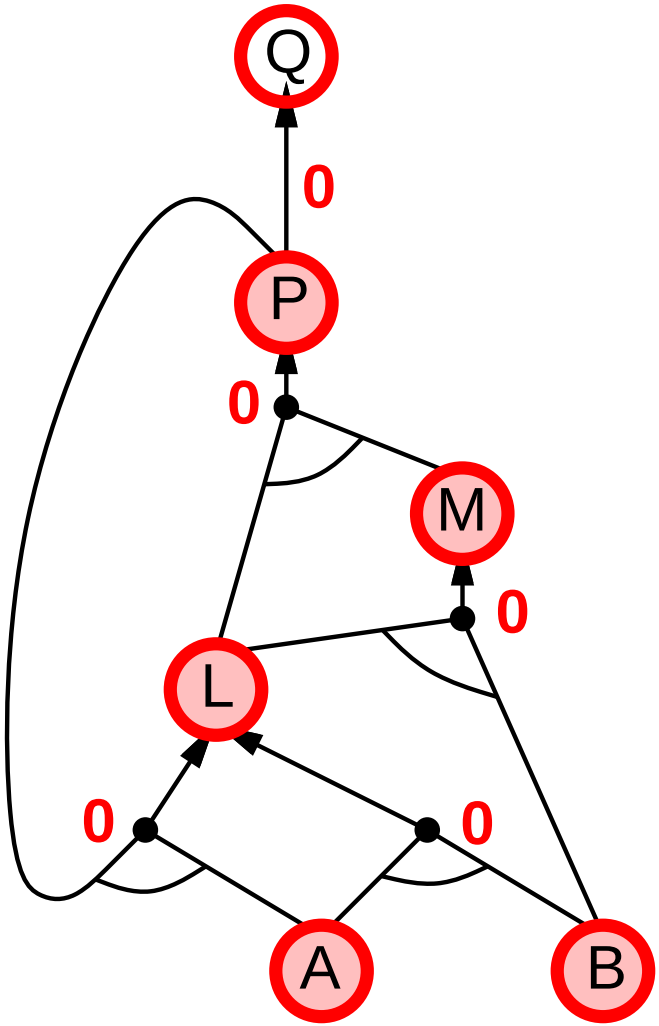
Forward chaining example



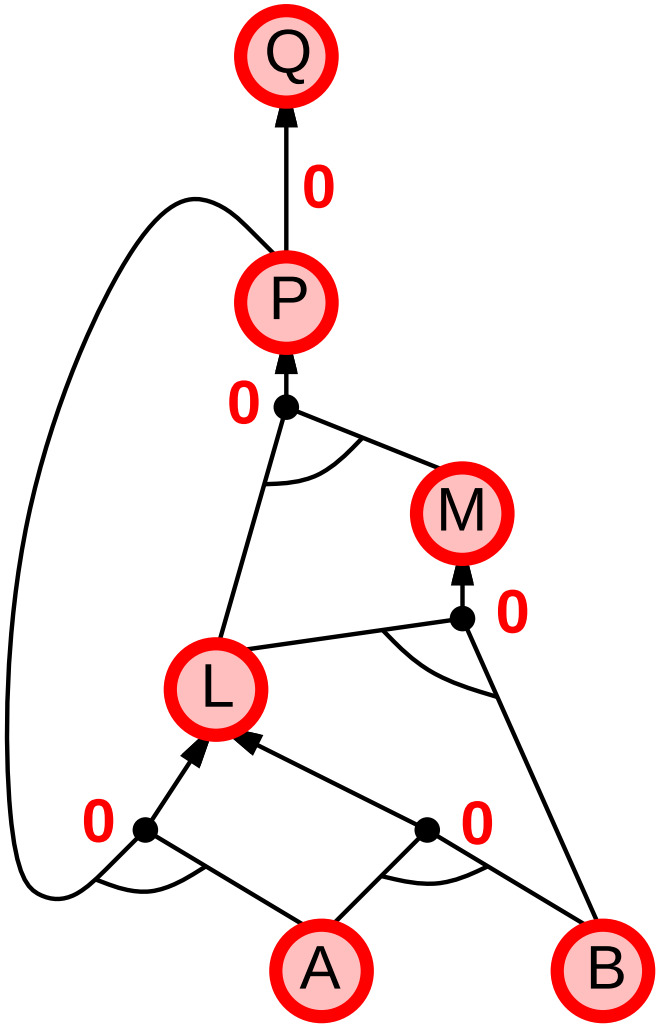
Forward chaining example



Forward chaining example



Forward chaining example



Backward chaining

Idea: work backwards from the query q:

to prove q by BC,

check if q is known already, or

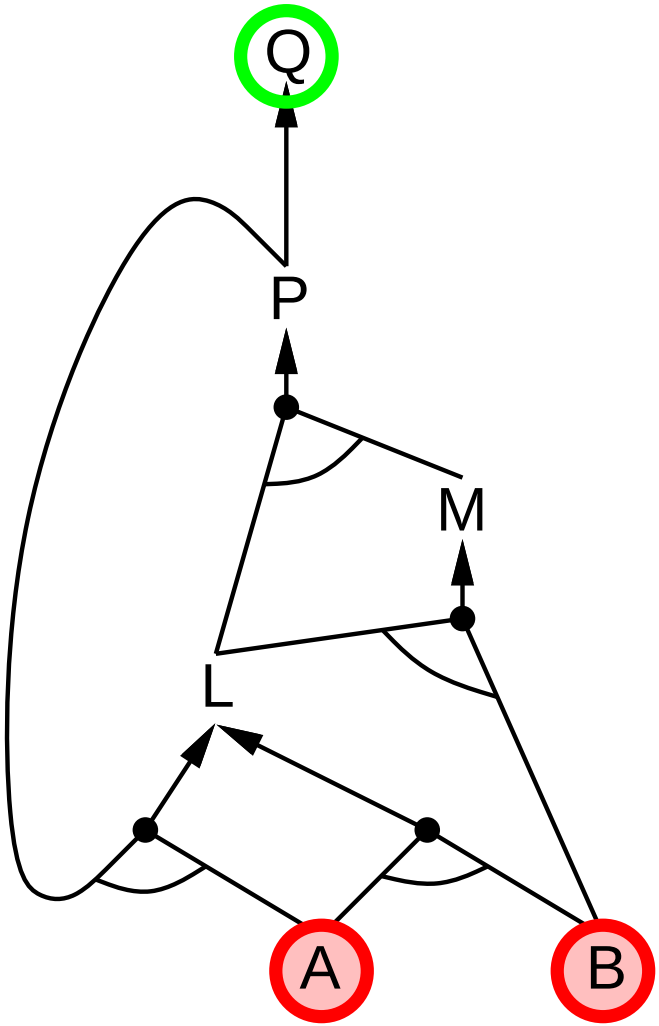
prove by BC all premises of some rule concluding q

Avoid loops: check if new subgoal is already on the goal stack

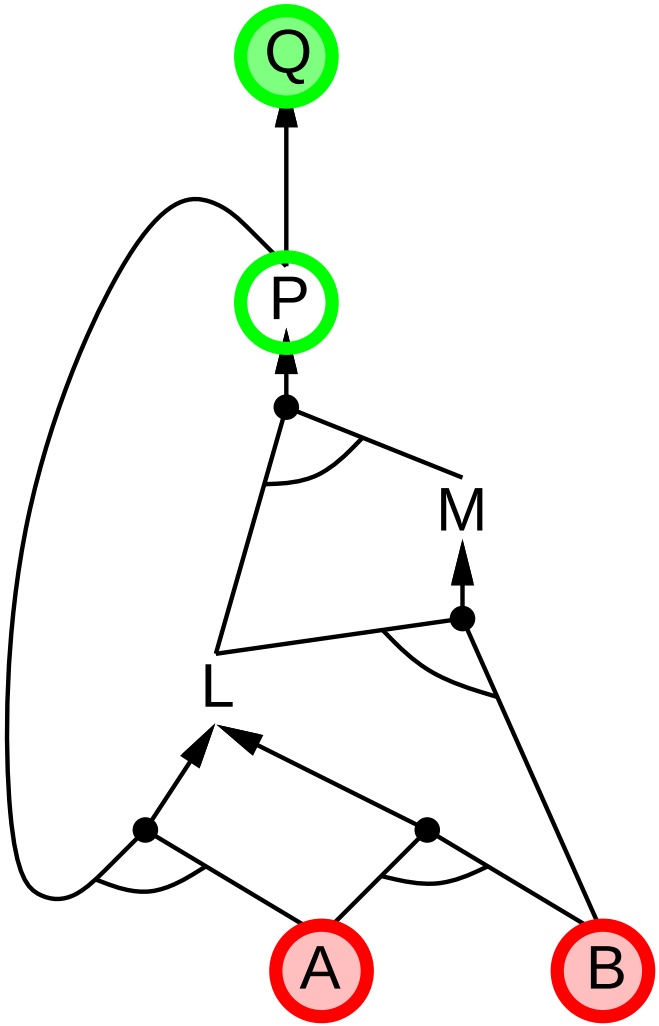
Avoid repeated work: check if new subgoal

1) has already been proved true, or
2) has already failed

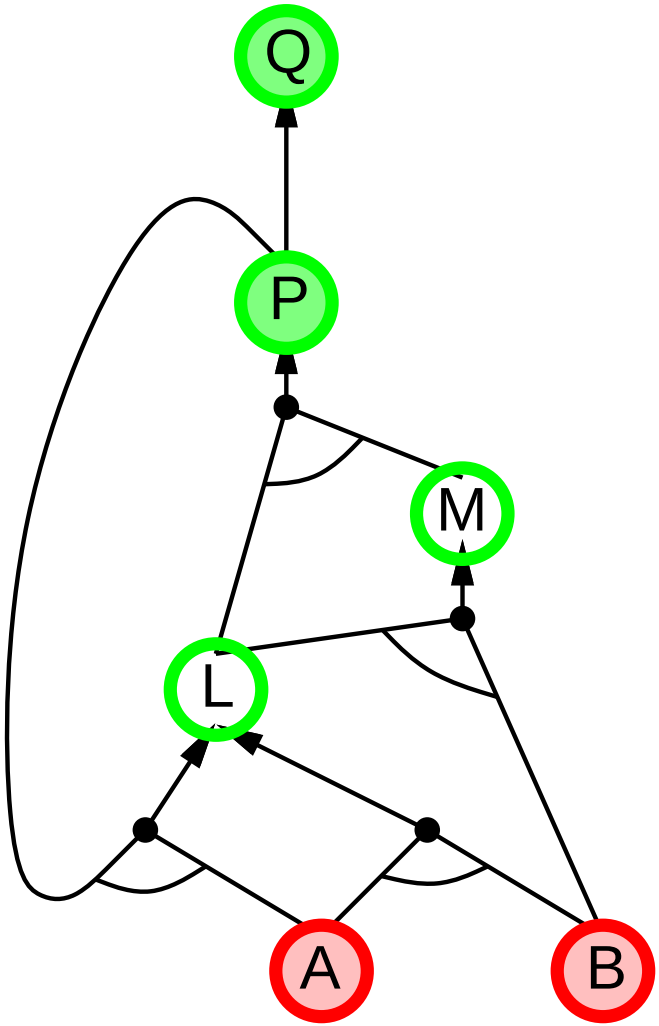
Backward chaining example



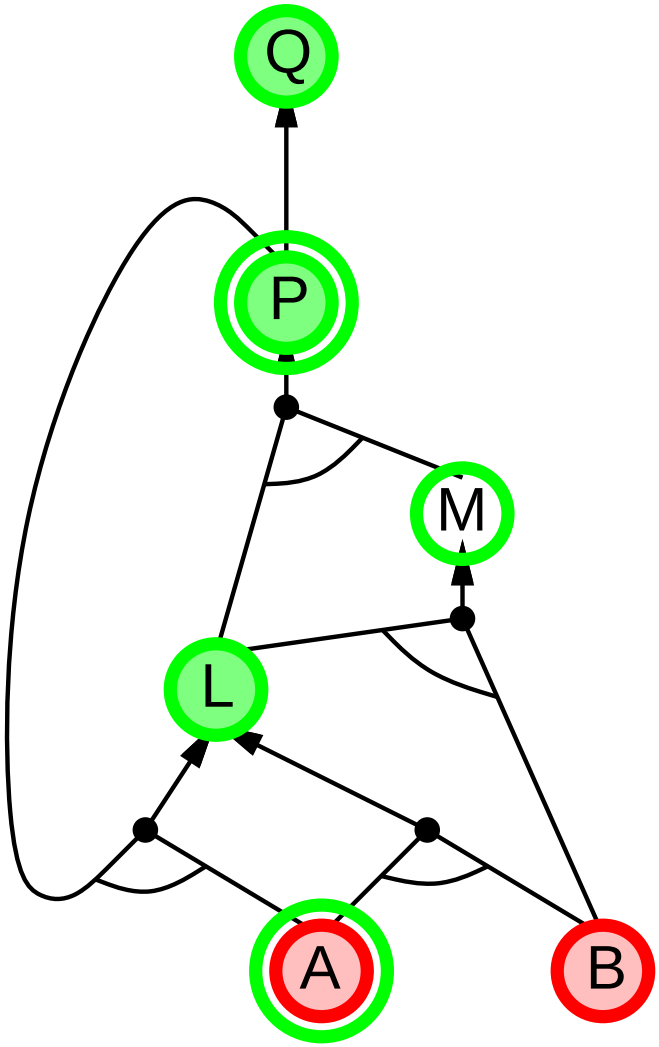
Backward chaining example



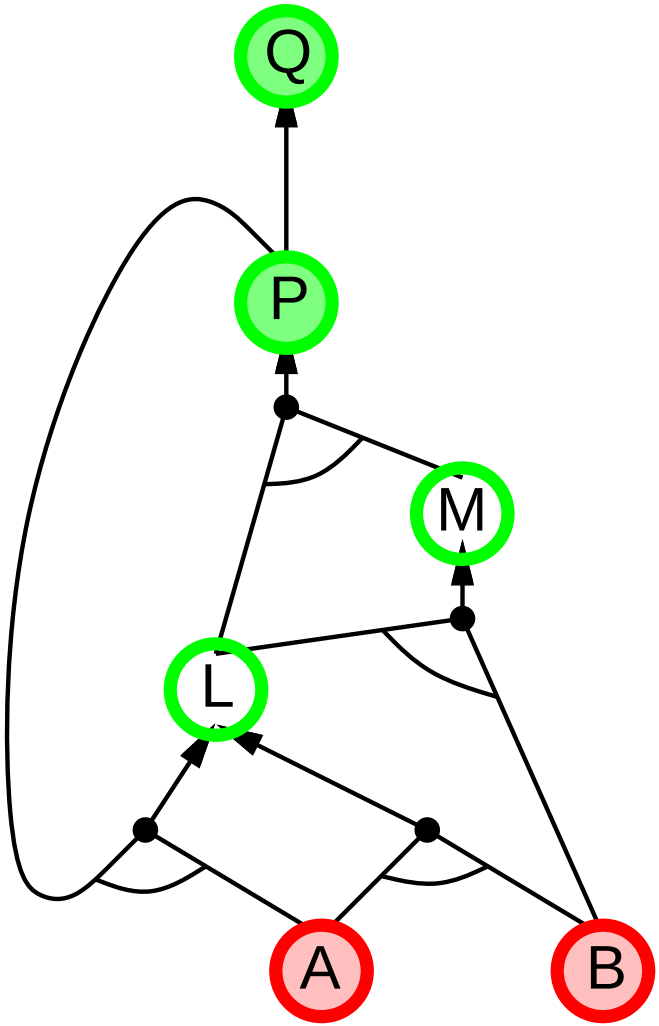
Backward chaining example



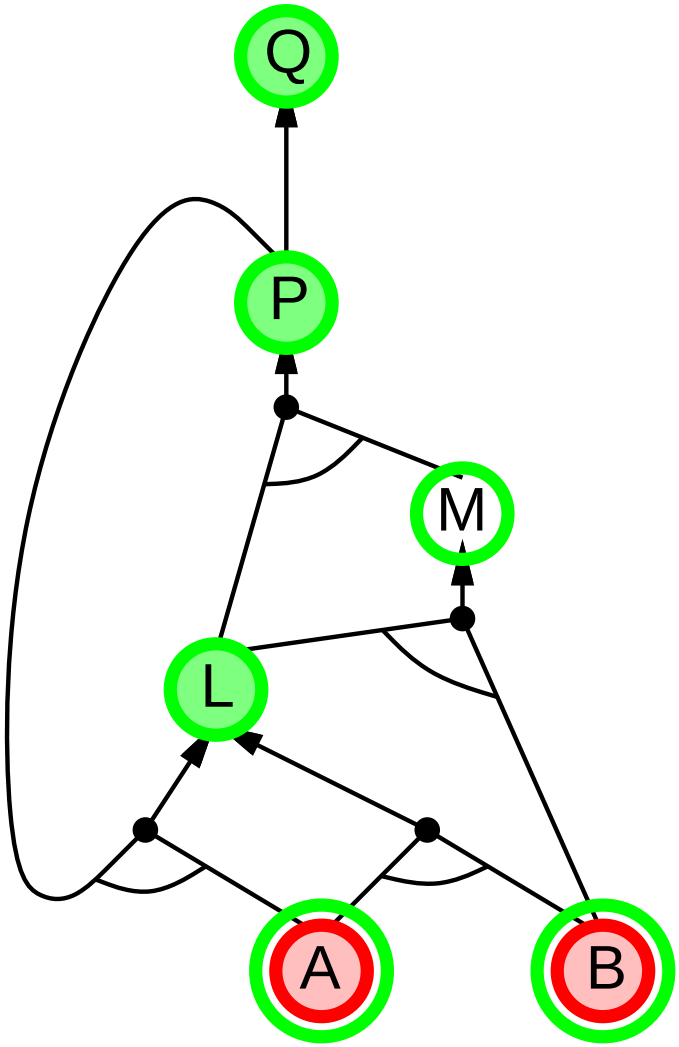
Backward chaining example



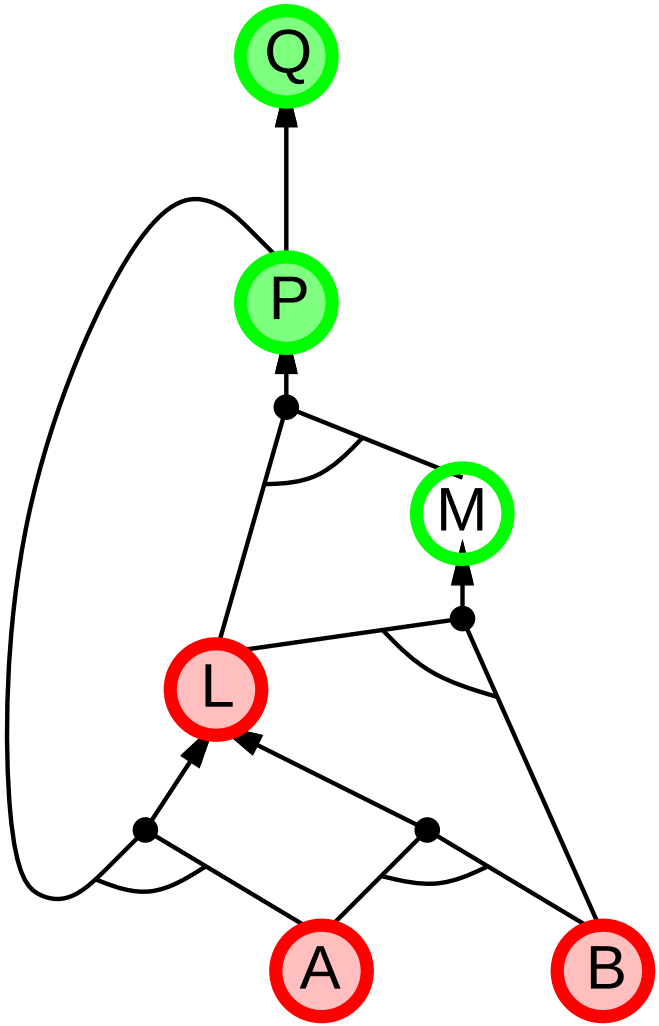
Backward chaining example



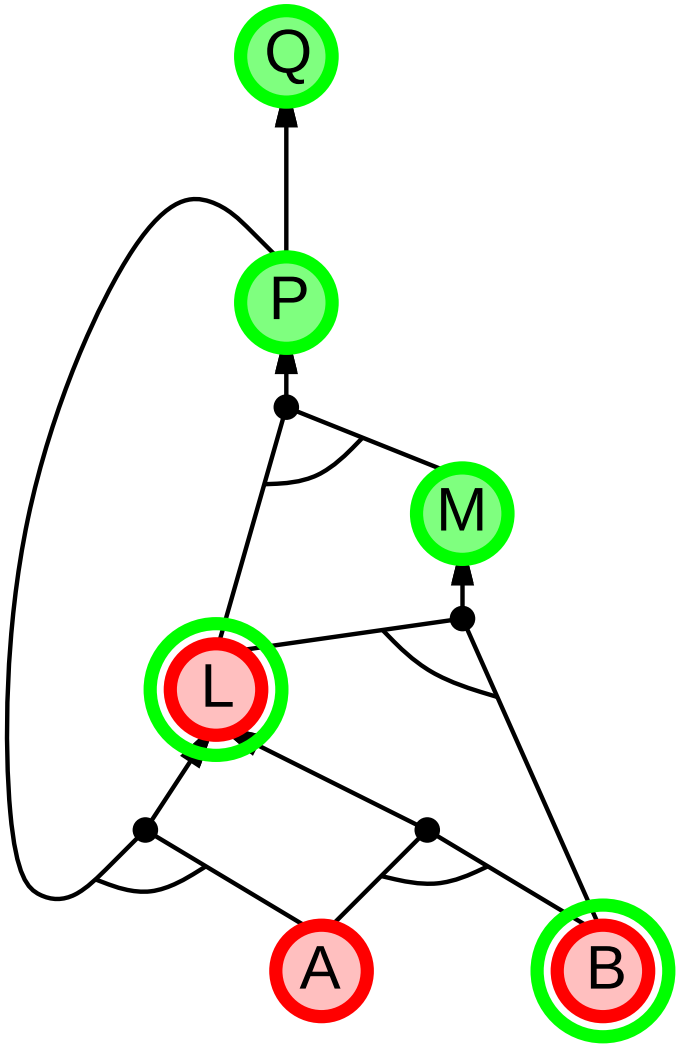
Backward chaining example



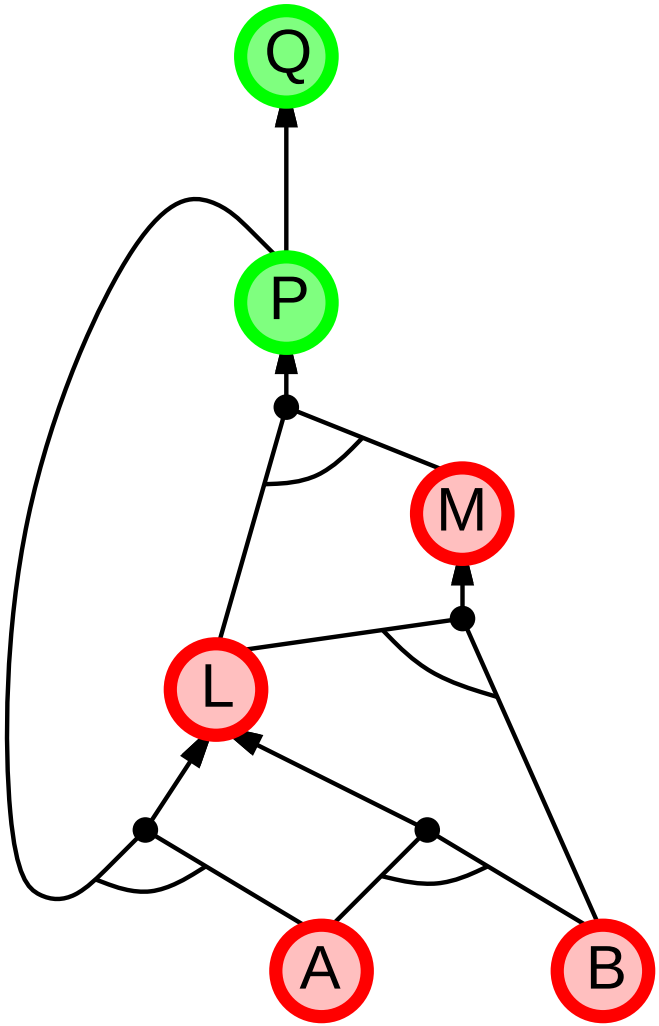
Backward chaining example



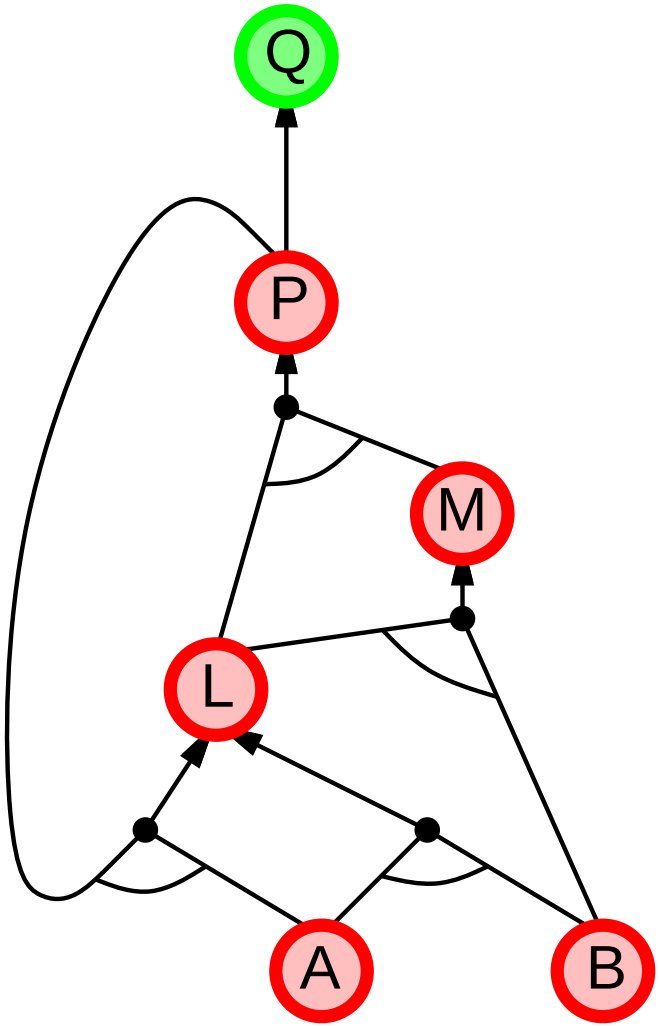
Backward chaining example



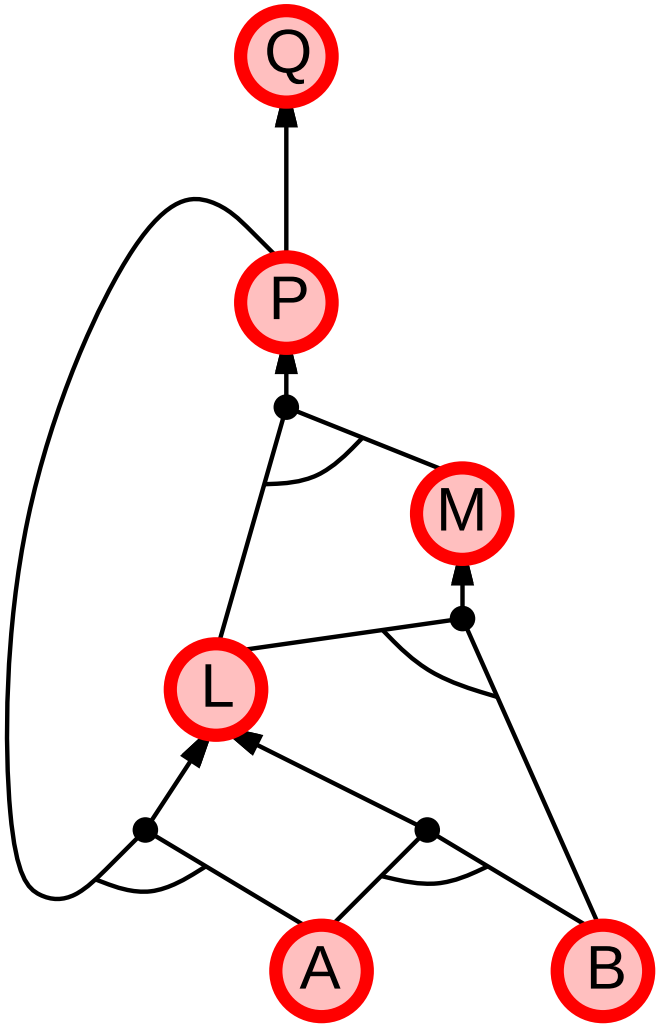
Backward chaining example



Backward chaining example



Backward chaining example



Forward vs. backward chaining

FC is data-driven, cf. automatic, unconscious processing,
e.g., object recognition, routine decisions

May do lots of work that is irrelevant to the goal

BC is goal-driven, appropriate for problem-solving,
e.g., Where are my keys? How do I get into a PhD program?

Complexity of BC can be *much less* than linear in size of KB

ships between the problems and the corresponding diagnoses. Inductive approaches are particularly suited for domains in which principles are hard to formalize (weather prediction, medical diagnosis etc.) or where experts are few (space exploration etc.). Some widely used inductive learning systems include those based on decision trees (Quinlan, 1993) (see section 8), neural networks (Gallant, 1993; Hassoun, 1995) (see section 9), and statistical pattern classification (Duda & Hart, 1973; Fukunaga, 1990; Ripley, 1996) (see section 10).

4 EXPERT SYSTEMS

Expert systems (Durkin, 1994; Puppe, 1993; Stefik, 1995) are programs that model the expertise (knowledge) and reasoning capabilities of qualified specialists within fairly narrow domains (e.g., diagnosis of heart diseases, credit evaluation, etc.). Such systems are typically composed of three essential modules: a *knowledge base* that captures the expertise of the specialist, an *inference engine* that mimics the specialist's reasoning process, and a *working memory* that is used as a scratch pad. In the course of a problem-solving session, the working memory holds the facts provided by the user (e.g., symptoms in a diagnosis task) and intermediate conclusions derived by the inference procedure (Durkin, 1994; Stefik, 1995). A schematic of an expert system is shown in figure 1.

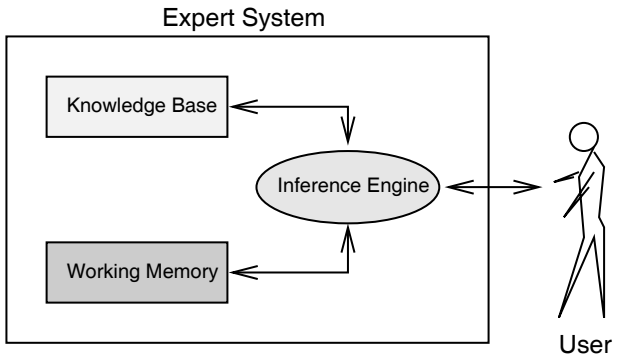


Figure 1: Schematic of an expert system. The knowledge base contains the domain knowledge encoded in some appropriate form (e.g., rules). In a consultation session with the user the inference engine generates inferences using the facts provided by the user and the rules in the knowledge base. The working memory holds user-provided facts and serves as a store for inferences or conclusions drawn based on the facts.

Critical to the process of expert system design is the task of knowledge engineering. A knowledge engineer interviews experts to obtain qualitative knowledge from them

and codes it in the knowledge base using some representation scheme. Typical examples of knowledge representation schemes used in expert systems are: rules, semantic networks, frames, logic etc. For example, rules assert the known relations between *premises* and *conclusions*. In our automobile diagnosis example, we could formulate the domain knowledge about non-functioning Headlights in terms of the following rules:

Rule 1: IF Headlights don't work
THEN faulty Bulbs or/and Battery

Rule 2: IF faulty Bulbs or/and Battery
AND Engine does not start
THEN faulty Battery

Rule 3: IF faulty Bulbs or/and Battery
AND Engine starts
THEN faulty Bulbs

During a consultation with an expert system the user enters facts regarding a current problem into the working memory. The system matches these facts with the knowledge contained in the knowledge base to infer new facts. These new facts are then entered into the working memory and the process continues till the system completes the task presented to it or runs into a dead end. For instance in our automobile example, if the user reports a problem with his headlights while the engine starts just fine, then these two observations become a part of the initial working memory of the expert system for that consultation session.

The inference engine mimics the expert's reasoning process. It works from the facts in the working memory and uses the domain knowledge contained in the knowledge base to derive (or infer) new facts. It achieves this by searching through the knowledge base to find rules whose premises match the facts contained in the working memory. If such a match is found, it simply adds the conclusion of the matching rule to the working memory. This process continues until the inference mechanism is unable to match any rules with the facts in the working memory. In practice, inference process can be further complicated for a number of reasons. For instance, when multiple rules match the data in the working memory, mechanisms have to be provided for conflict resolution. Typical conflict resolution mechanisms are based on specificity of the rules (more specific rules, i.e., those that require more premises to hold, over more general ones); recency of the facts matched by the rules (as determined by the time of their entry into the working memory) etc.

Ideally, the inference procedure used by the system must be both *sound* and *complete*. Soundness and completeness are desirable formal properties of reasoning systems. Loosely speaking, soundness ensures that the conclusions drawn by the system are true

in all the scenarios in which the facts on which they are based are true. Completeness ensures that the inference procedure is able to derive all true conclusions that follow from a set of facts and rules (Ginsberg, 1993; Russell & Norvig, 1995). Sometimes soundness and/or completeness of inference has to be sacrificed for efficiency.

Expert systems can be distinguished based on the inference strategy (forward-chaining from facts to conclusions, backward-chaining from hypotheses to premises, etc.), problem-solving strategy (bottom-up as opposed to top-down), knowledge representation (rule-based, frame-based, case-based etc.), inference techniques (deductive, non-monotonic, probabilistic, fuzzy, etc.), and their problem-solving paradigm (analysis: as in the case of diagnosis systems, versus synthesis: as in the case of design systems), etc. Their detailed treatment is beyond the scope of this paper. The interested reader is referred to (Durkin, 1994; Puppe, 1993; Stefik, 1995) for details.

As an illustration, let us assume that figure 2 constitutes the complete knowledge base for our automobile example (we assume that these rules were constructed somehow through a process of knowledge engineering).

Now, suppose a user reports a problem wherein the Headlights don't work. The expert system adds this fact to its working memory and then looks for a rule in the knowledge base that matches this data. It finds rule 1 and adds the conclusion, namely faulty Bulbs or/and Battery to the working memory. The inference engine cycles through the knowledge base again looking for a match for this newly added data in the working memory. It finds rule 2, but rule 2 requires another premise also to be satisfied, namely Engine does not start. Hence, the expert system queries the user about the status of the engine. Now suppose the user responds with the fact that the Engine does not start, the inference engine uses this new information to match rule 2, thereby adding the conclusion faulty Battery to the working memory. Further attempts to match this newly added information with rules in the knowledge base fail. Hence the inference procedure stops with the conclusion that the user is experiencing problems because of a faulty Battery. If the user had responded with Engine starts instead, rule 3 would have matched, leading to a conclusion of faulty Bulbs. This is essentially how an expert system functions. Notice that in the preceding discussion we have side-stepped issues like conflict resolution.

Our knowledge base does not support diagnoses involving multiple faulty components. Suppose the above diagnostic scenario had access to further information regarding the wipers. For instance, suppose we know in addition that the Wipers work. This changes the diagnosis completely, since working wipers vouch for a healthy battery. Thus in this case failure of headlights and engine must be due to faulty Ignition AND faulty Bulbs rather than a faulty Battery. However, irrespective of the sequence in which the symptoms become available, this diagnosis cannot be derived from our knowledge base. This is a common problem in expert system design and care must be taken to make sure that the rules in the knowledge base adequately cover the domain of interest.

Expert system design is a challenging task for a number of reasons. First, the experts

```
1.  IF      Headlights don't work
    THEN  faulty Bulbs or/and Battery
2.  IF      faulty Bulbs or/and Battery
    AND   Engine does not start
    THEN  faulty Battery
3.  IF      faulty Bulbs or/and Battery
    AND   Engine starts
    THEN  faulty Bulbs
4.  IF      Engine does not start
    THEN  faulty Battery or/and Ignition
5.  IF      faulty Battery or/and Ignition
    AND   Headlights work
    THEN  faulty Ignition
6.  IF      Wipers don't work
    AND   Engine does not start
    THEN  faulty Battery
7.  IF      Wipers don't work
    AND   Engine starts
    THEN  faulty Wiper Motor
8.  IF      Wipers don't work
    AND   Headlights work
```

Figure 2: The knowledge base of an expert system for the automobile diagnosis problem. Domain knowledge is coded in the form of IF-THEN rules. Inferences are drawn by matching the IF part of the rules with the known facts in the working memory and adding the conclusions back to the working memory. These conclusions represent partial inferences. This procedure is repeated until no further rules match the inferences in the working memory.

providing the domain knowledge must be able to articulate their expertise. In our automobile diagnosis example, the expert must be able to put down the relationships between the different automobile components and their dependencies. Second, the knowledge engineer working with the experts should be able to understand the expert's knowledge well enough to choose a good representation scheme to be used in the knowledge base and must determine an appropriate inference mechanism to work with the representation chosen. In our example, *rules* were chosen as the representation scheme. Third, the design of the knowledge base and the inference mechanism, the conflict resolution strategies used, and the order in which facts enter working memory can interact in fairly complex ways making the task of testing and debugging an expert system extremely difficult in practice.