MODULE 6

SYLLABUS

- Expert Systems: rule based expert systems.
- Natural language processing
  - natural language understanding problem,
  - deconstructing language.
- Syntax stochastic tools for language analysis, natural language applications

Expert Systems

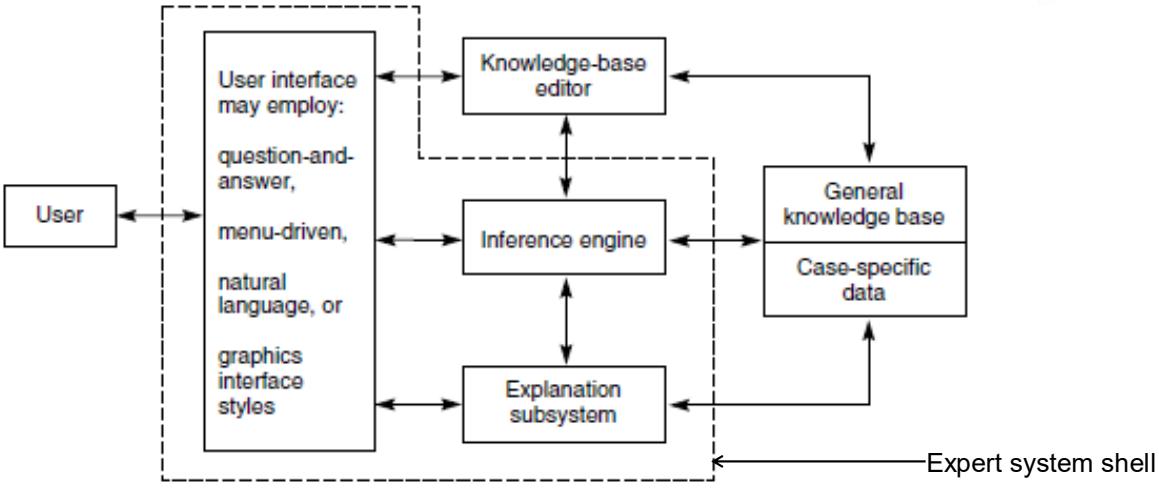


Figure 8.1 Architecture of a typical expert system for a particular problem domain.

User interface

- The user interacts with the system through a user interface it can be
  - question-and-answer,
  - menu-driven, or
  - graphical interfaces.
- final decision on the interface type is a compromise between user needs and the requirements of the knowledge base and inferencing system

Knowledge base

- The heart of the expert system is the knowledge base, which contains the knowledge of a particular application domain.
- The knowledge base contains both general knowledge as well as case-specific information.

Inference engine

- The inference engine applies the knowledge to the solution of actual problems.
- It is essentially an interpreter for the knowledge base. In the production system, the inference engine performs the recognize-act control cycle.
- there is separation of knowledge and control which allows changes to be made in one part of the knowledge base without creating side effects in others and allows the same control and interface software to be used in a variety of systems

Explanation subsystem

- The explanation subsystem allows the program to explain its reasoning to the user.
- These explanations include justifications for the system's conclusions, in response to how queries, explanations of why the system needs a particular piece of data, why queries, and, where useful, tutorial explanations or deeper theoretical justifications of the program's actions.

Knowledge-base editors

- Knowledge-base editors help the programmer locate and correct bugs in the program's performance, often accessing the information provided by the explanation subsystem.
- They also may assist in the addition of new knowledge, help maintain correct rule syntax, and perform consistency checks on any updated knowledge base.

Selecting a Problem and the Knowledge Engineering Process

- The need for the solution justifies the cost and effort of building an expert system.
- Human expertise is not available in all situations where it is needed.
- The problem may be solved using symbolic reasoning.
- The problem domain is well structured and does not require common sense reasoning.
- The problem may not be solved using traditional computing methods.
- Cooperative and articulate experts exist.
- The problem is of proper size and scope.

Primary people involved in building an expert system

- primary people involved in building an expert system are the
  - Knowledge engineer,
  - Domain expert, and
  - End user.
- Expert systems are built by progressive approximations, with the program's mistakes leading to corrections or additions to the knowledge base.
- In a sense, the knowledge base is "grown" rather than constructed.

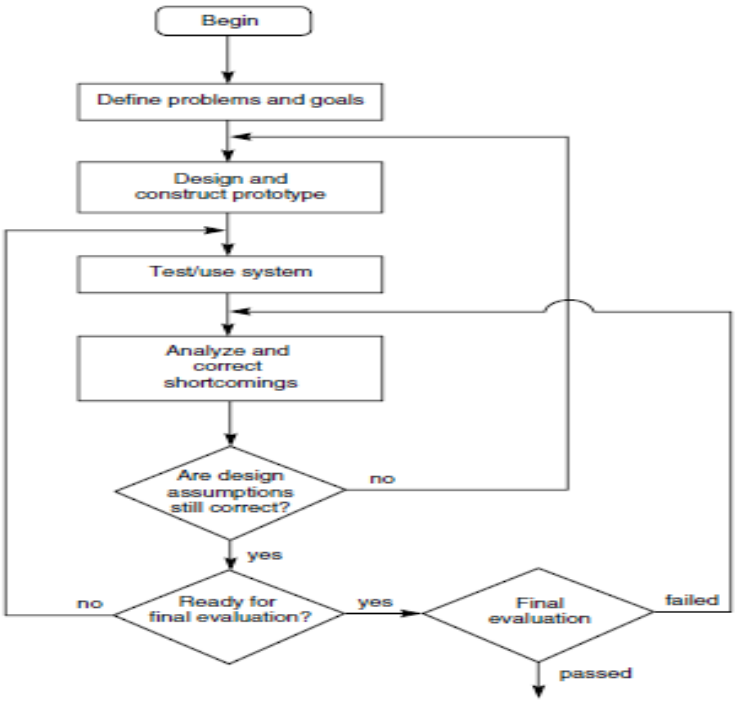


Figure 8.2 Exploratory development cycle.

Conceptual Models and Their Role in Knowledge Acquisition

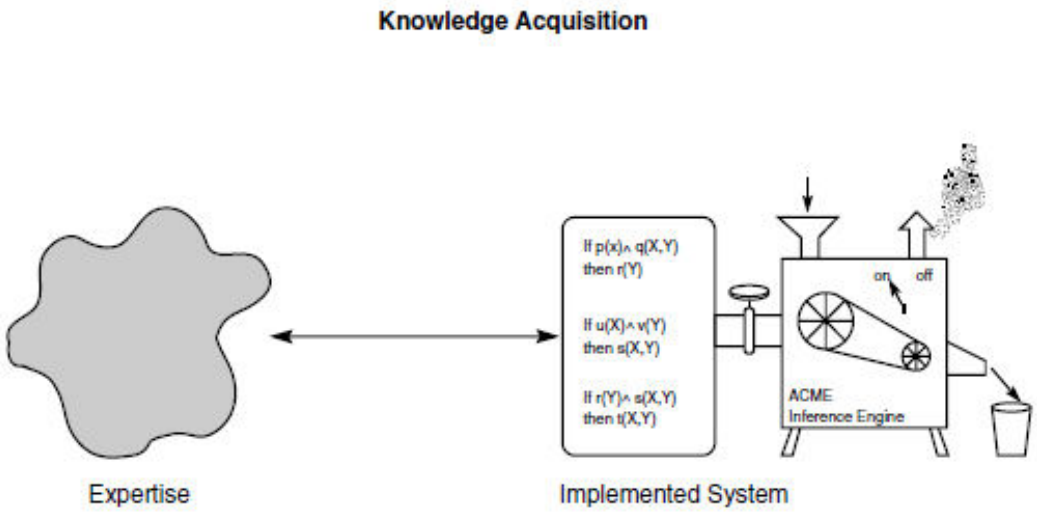


Figure 8.3 The standard view of building an expert system.

- The knowledge engineer must translate this informal expertise into a formal language suited to a computational system.
- A number of important issues arises in the process of formalizing human skilled performance:
  - Human skill is often inaccessible to the conscious mind.
  - Human expertise often takes the form of knowing how to cope in a situation rather than knowing what a rational characterization of the situation might be
  - Expertise changes

- By a conceptual model, we mean the knowledge engineer's evolving conception of the domain knowledge.
- Although this is undoubtedly different from the domain expert's, it is this model that actually determines the construction of the formal knowledge base.
- An expert system should include a requirements document; however, because of the constraints of exploratory programming, expert system requirements should be treated as coevolving with the prototype.

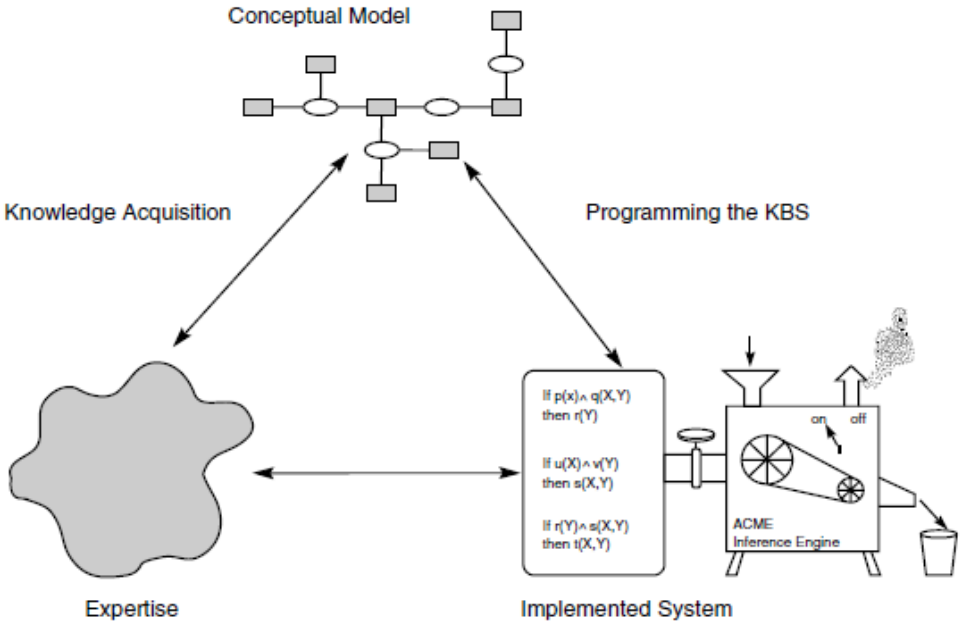


Figure 8.4 The role of mental or conceptual models in problem solving.

Rule-Based Expert Systems

- Rule-based expert systems represent problem-solving knowledge as if... then... rules.
- This approach is one of the oldest techniques for representing domain knowledge in an expert system.
- It is also one of the most natural, and remains widely used in practical and experimental expert systems.

- In a rule-based system, condition action pairs are represented as if... then... rules, with the premises of the rules,
- the if portion, corresponding to the condition, and the conclusion,
- the then portion, corresponding to the action: when the condition is satisfied, the expert system takes the action of asserting the conclusion as true.
- The inference engine implements the recognize-act cycle of the production system; this control may be either data-driven or goal-driven

goal-driven expert system

- The goal expression is initially placed in working memory.
- The system matches rule conclusions with the goal, selecting one rule and placing its premises in the working memory.
- This corresponds to a decomposition of the problem's goal into simpler subgoals.
- The process continues in the next iteration of the production system, with these premises becoming the new goals to match against rule conclusions.
- The system thus works back from the original goal until all the subgoals in working memory are known to be true, indicating that the hypothesis has been verified.

Example

1. Rule 1: if
   the engine is getting gas, and
   the engine will turn over,
   then
   the problem is spark plugs.

2. Rule 2: if
   the engine does not turn over, and
   the lights do not come on
   then
   the problem is battery or cables.

3. Rule 3: if
   the engine does not turn over, and
   the lights do come on
   then
   the problem is the starter motor.

4. Rule 4: if
   there is gas in the fuel tank, and
   there is gas in the carburetor
   then
   the engine is getting gas.

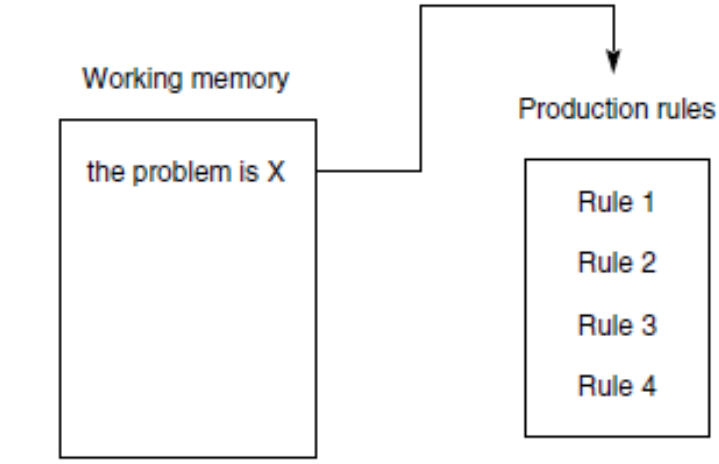


Figure 8.5 The production system at the start of a consultation in the car diagnostic example.

- To run this knowledge base under a goal-directed control regime, place the top-level goal, the problem is X, in working memory
- X is a variable that can match with any phrase, as an example, the problem is battery or cables; it will become bound to the solution when the problem is solved.

- Three rules match with this expression in working memory: rule 1, rule 2, and rule 3.
- If we resolve conflicts in favor of the lowest-numbered rule, then rule 1 will fire.
- This causes X to be bound to the value spark plugs and the premises of rule 1 to be placed in the working memory.
- The system has thus chosen to explore the possible hypothesis that the spark plugs are bad.

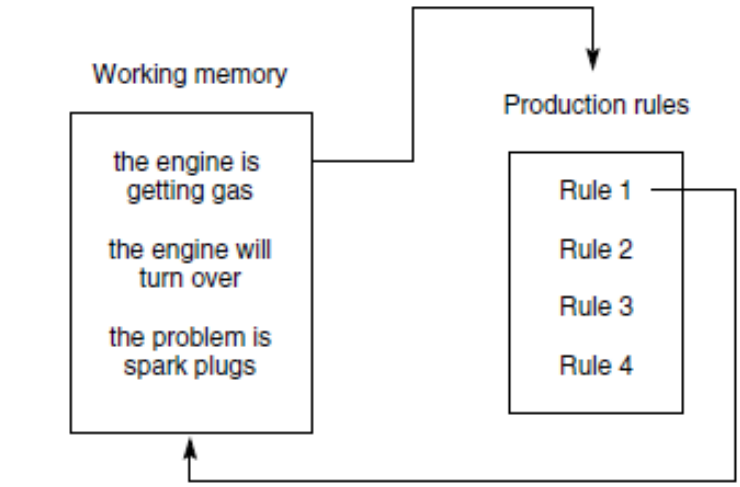


Figure 8.6 The production system after Rule 1 has fired.

- Note that there are two premises to rule 1, both of which must be satisfied to prove the conclusion true.
- These are and branches of the search graph representing a decomposition of the problem (finding whether the problem is spark plugs) into two subproblems (finding whether the engine is getting gas and whether the engine will turn over).
- We may then fire rule 4, whose conclusion matches with the engine is getting gas, causing its premises to be placed in working memory

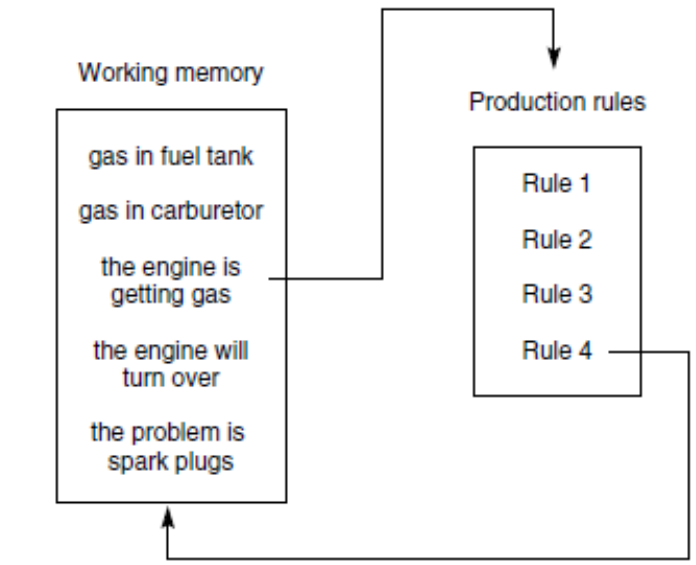


Figure 8.7 The system after Rule 4 has fired. Note the stack-based approach to goal reduction.

- At this point, there are three entries in working memory that do not match with any rule conclusions.
- Our expert system will, in this situation, query the user directly about these subgoals.
- If the user confirms all three of these as true, the expert system will have successfully determined that the car will not start because the spark plugs are bad.
- In finding this solution, the system has searched the leftmost branch of the and/or graph

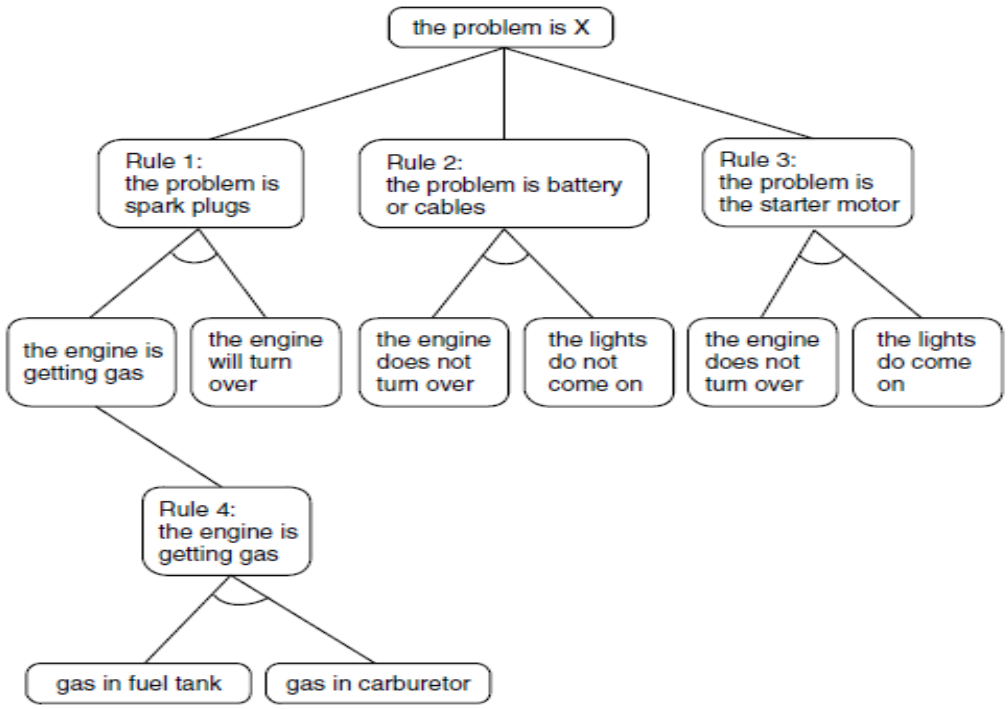
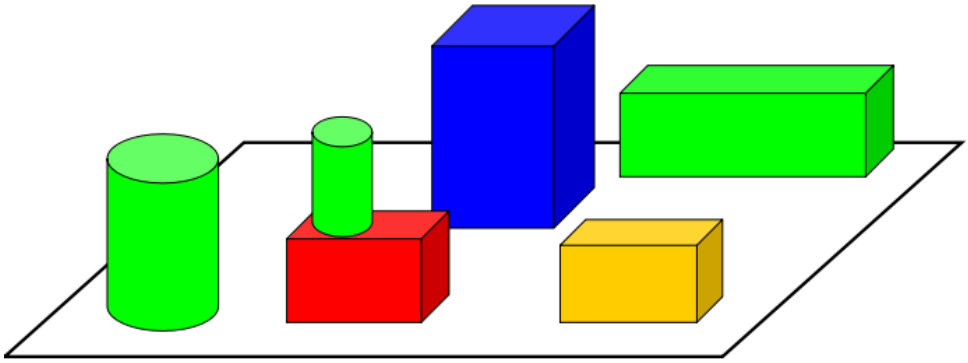


Figure 8.8 The and/or graph searched in the car diagnosis example, with the conclusion of Rule 4 matching the first premise of Rule 1.

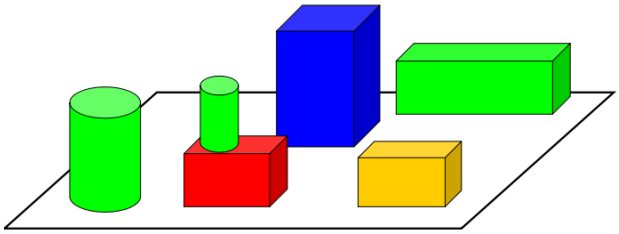
Natural language processing

- An early natural language understanding system: SHRDLU (Winograd, 1972)



It could converse about a blocks world

- What is sitting on the red block?
- What shape is the blue block on the table?
- Place the green cylinder on the red brick.
- What color is the block on the red block? Shape?



The problems

- Understanding language is not merely understanding the words: it requires inference about the speaker's goals, knowledge, assumptions. The context of interaction is also important.
  - Do you know where Rekhi 309 is?
  - Yes.
  - Good, then please go there and pick up the documents.
  - Do you know where Rekhi 309 is?
  - Yes, go up the stairs and enter the semi-circular section.
  - Thank you.

- Implementing a natural language understanding program requires that we represent knowledge and expectations of the domain and reason effectively about them.
  - nonmonotonicity
  - belief revision
  - metaphor
  - planning
  - learning
  - …

Shall I compare thee to a summer's day?
Thou art more lovely and more temperate:
Rough winds do not shake the darling buds of May,
And summer's lease hath all too short a date:

Shakespeare's Sonnet XVIII

- There are three major issues involved in understanding natural language:
  - A large amount of human knowledge is assumed.
  - Language is pattern based. Phoneme, word, and sentence orders are not random.
  - Language acts are products of agents embedded in complex environments.

SHRDLU's solution

- Restrict focus to a *microworld* : blocks world
- Constrain the language: use templates
- Do not deal with problems involving commonsense reasoning: still can communicate meaningfully

Linguists' approach

- *Prosody*: rhythm and intonation of language
- *Phonology*: sounds that are combined
- *Morphology*: morphemes that make up words
- *Syntax*: rules for legal phrases and sentences
- *Semantics*: meaning of words, phrases, sentences
- *Pragmatics*: effects on the listener
- *World knowledge*: background knowledge of the physical world

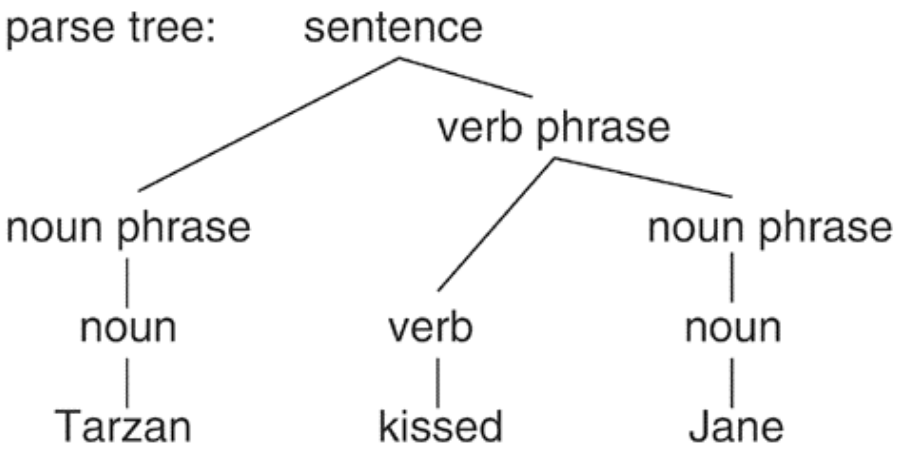
Stages of language analysis

1. *Parsing*: analyze the syntactic structure of a sentence
2. *Semantic interpretation*: analyze the meaning of a sentence
3. *Contextual/world knowledge representation*: Analyze the expanded meaning of a sentence

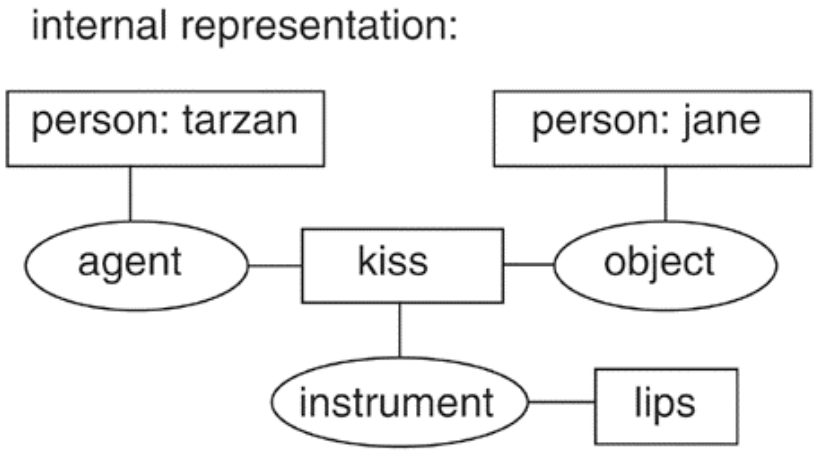
For instance, consider the sentence:

- Tarzan kissed Jane.

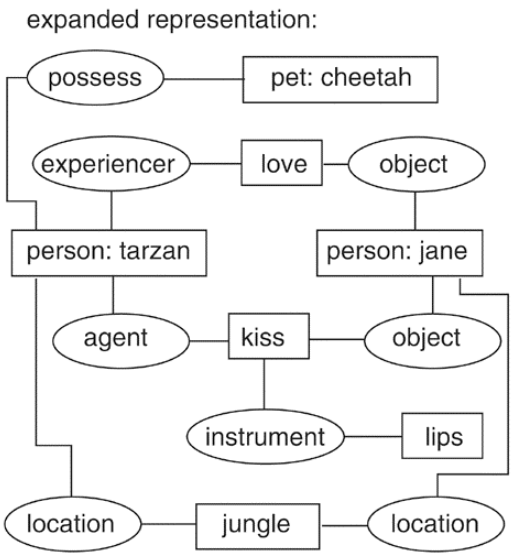
The result of parsing would be:



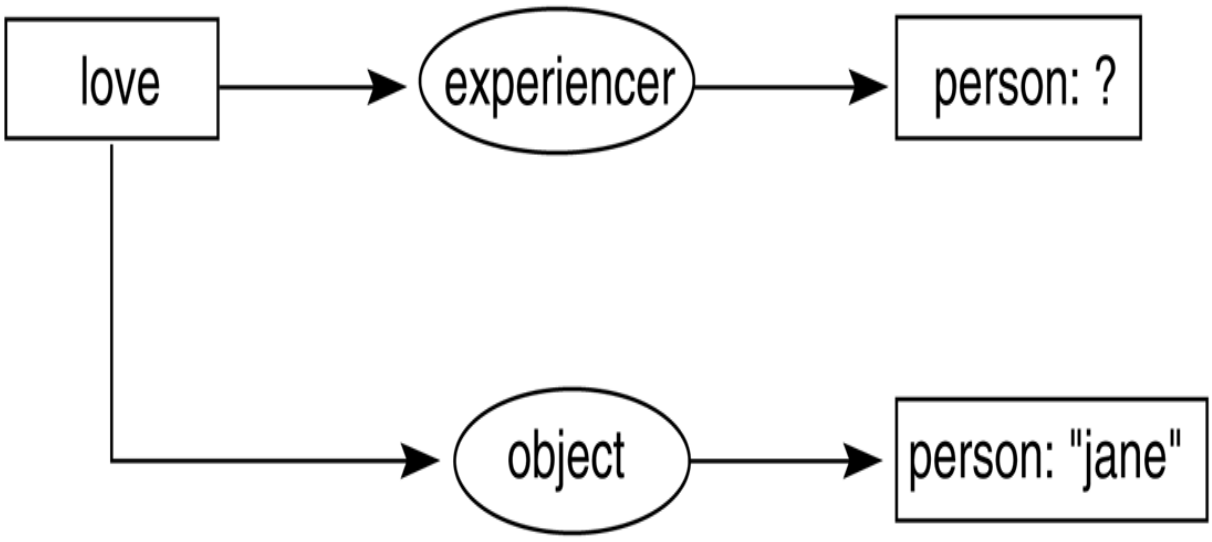
The result of semantic interpretation



The result of contextual/world knowledge interpretation

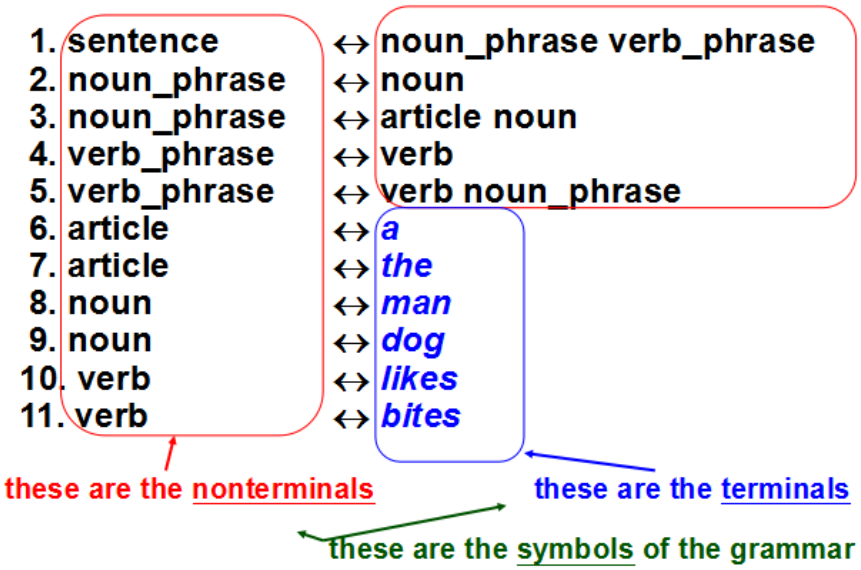


Can also represent questions: Who loves Jane?



Parsing using Context-Free Grammars

- A bunch of *rewrite rules*:

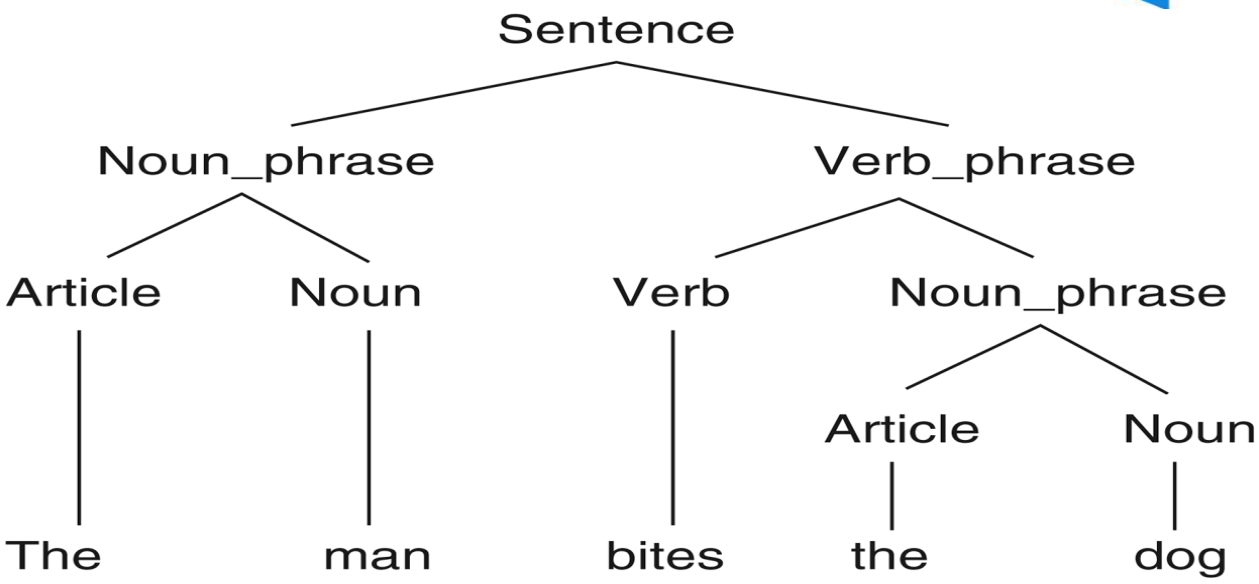


Parsing

- It is the search for a legal *derivation* of the sentence.
- sentence → noun_phrase verb_phrase
  → article noun verb_phrase
  → The noun verb_phrase
  → The man verb_phrase
  → The man verb noun_phrase
  → The man bites noun_phrase
  → The man bites article noun
  → The man bites the noun
  → The man bites the dog
- Each intermediate form is a *sentential form* .

- The result is a *parse tree*. A parse tree is a structure where each node is a symbol from the grammar. The root node is the *starting nonterminal*, the intermediate nodes are nonterminals, the leaf nodes are terminals.
- "Sentence" is the *starting nonterminal*.
- There are two classes of parsing algorithms
  - *top-down parsers*: start with the starting symbol and try to derive the complete sentence
  - *bottom-up parsers*: start with the complete sentence and attempt to find a series of reductions to derive the start symbol

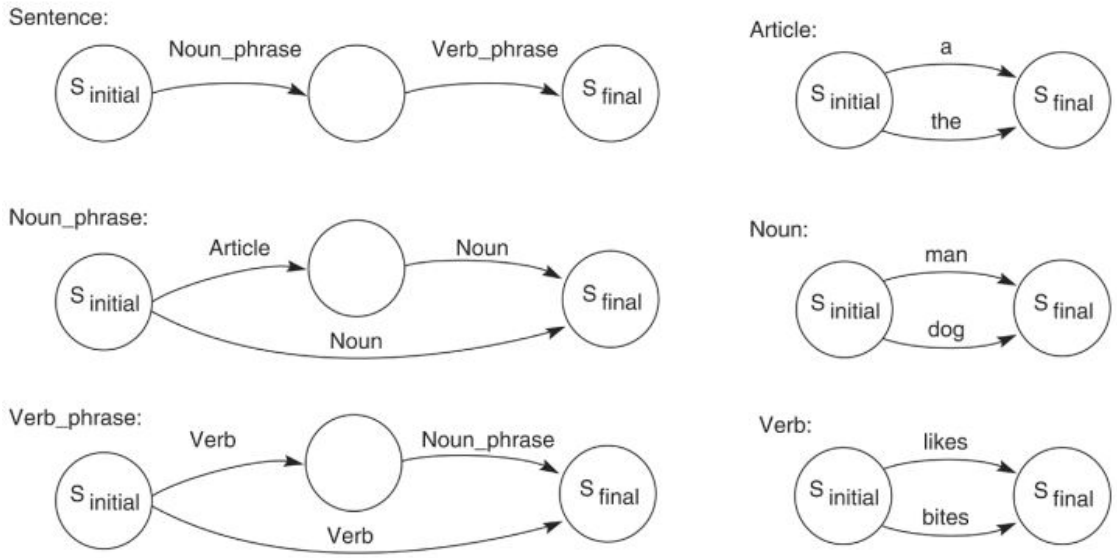
The parse tree



Parsing is a search problem

- Search for the correct derivation
- If a wrong choice is made, the parser needs to backtrack
- *Recursive descent parsers* maintain backtrack pointers
- *Look-ahead techniques* help determine the proper rule to apply

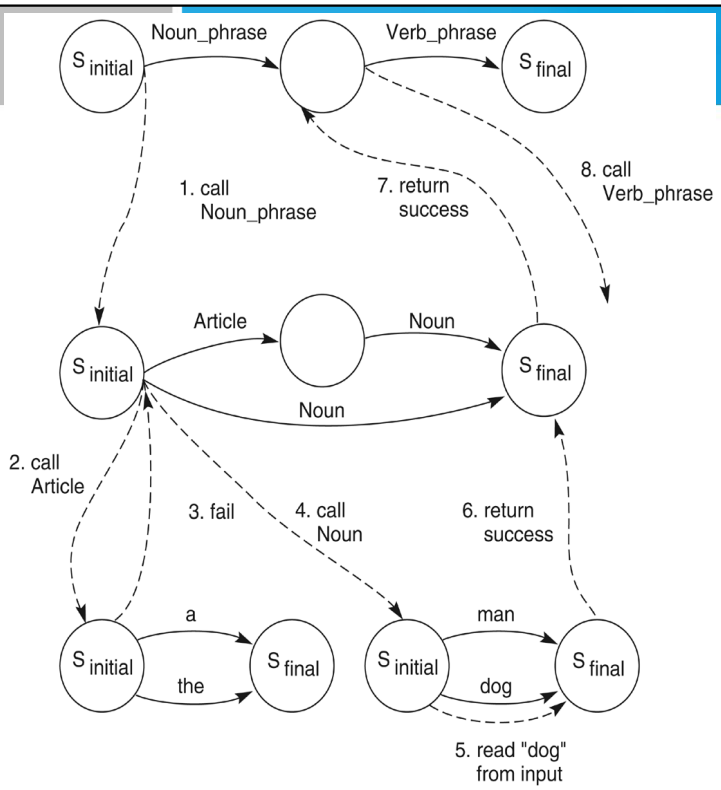
Transition networks



- It is a set of finite-state machines representing the rules in the grammar
- Each network corresponds to a single nonterminal
- Arcs are labeled with either terminal or nonterminal symbols
- Each path from the start state to the final state corresponds to a rule for that nonterminal
- If there is more than one rule for a nonterminal there are multiple paths from the start to the goal (e.g., noun_phrase)

- Finding a successful transition through the network corresponds to replacement of the nonterminal with the RHS
- Parsing a sentence is a matter of traversing the network:
  - If the label of the transition (arc) is a terminal, it must match the input, and the input pointer advances
  - If the label of the transition (arc) is a nonterminal, the corresponding transition network is invoked recursively
  - If several alternative paths are possible, each must be tried (backtracking)---very much like nondeterministic finite automaton---until a successful path is found

Parsing the sentence “Dog bites.”



- A “successful parse” is the complete traversal of the net for the starting nonterminal from $s_{initial}$ to s_{final} .
- If no path works, the parse “fails.” It is not a valid sentence (or part of sentence).
- The following algorithm would be called using parse($s_{initial}$)
- It would start with the net for “sentence.”