Knowledge-Based Systems

- Expert Systems
- Knowledge Representation
- Production Systems
- CLIPS - the C Language Integrated Production System
- Reasoning under Uncertainty
- Homework: Download CLIPS from the Web
What Is an Expert System?

• An expert system deals with *ill-structured problems* with *expert-level performance*.

• Expert systems are also called knowledge based systems, because people have found that those systems need general domain knowledge as well as expertise.

• Expert systems were the main research stream of computer science as well as AI for the 1980’s. Many people still think AI is expert systems, although this is not accurate. In some institutions, AI courses are entitled “AI and Expert Systems”.

• Expert system shells: expert system building tools with ‘empty’ knowledge bases.
The Structure of an Expert System

- an interface to facilitate human-computer communication,
- a working memory/data base, which stores the evidence and intermediate results of a specific problem to be solved,
- a knowledge base, KB, acquired from a domain expert or a group of experts,
- an inference engine to solve users’ problems by applying the knowledge base,
- an explanation/tracing engine to tell the user how the solution has been achieved (hence answers why questions),
- a knowledge acquisition engine, K.A. Engine, to acquire and modify the knowledge in the knowledge base when necessary, and
- a knowledge base management subsystem, KBMS, to maintain the knowledge (e.g., detecting inconsistency such as repetitions, redundancy and contradictions) in the knowledge base.
Knowledge Processing in AI Systems

• Knowledge is divided into 3 levels:
  1. data: evidence of a specific problem,
  2. knowledge: applicable to all relevant problems, and
  3. knowledge manipulation, to acquire a KB and apply it to solve users’ problems.

• Three fundamental techniques in AI (including KBS):
  – Knowledge representation
  – Knowledge acquisition
  – Knowledge inference
Knowledge Representation

• What is knowledge? Knowledge contains significant objects and relations in a domain and is applicable to various problems in the domain. E.g., ‘People have 2 arms’.

• Data are evidence/descriptions about specific problems in the domain. E.g., ‘My height and weight are 1.72m and 68kg respectively’.

• A knowledge base (KB) stores knowledge, and a database (DB) stores data related to particular problems.

• A DB enumerates objects and relations of a domain (its extension); A KB includes the intentional knowledge used to reason about these objects.
Knowledge Representation (2)

- KR concerns with the design of data structures and their interpretation.

- Logic and state space are 2 representation schemes.
  In logic, a rule says that its head holds only when its body can be satisfied. In a state space, states can be transformed by applying a set of moves/procedures.

- Scheme and Medium:
  - Scheme is related to the conceptual design of data structures.
    Medium concerns with the implementation of a representation scheme, often programming languages.
  - A scheme can be implemented with different mediums; A programming language can be used to implement different representation schemes.
  - E.g., you have been asked to use Prolog to implement a state space search. You can also use C to do another implementation.
Knowledge Representation (3)

Classification of knowledge representation schemes


- **Procedural representation**: Knowledge is represented as a set of instructions for solving problems. Typical structure: *If P₁, ..., Pₘ then G₁, ..., Gₙ* in *production systems*. Interpretation: To achieve the goals G₁, ..., Gₙ, the premises P₁, ..., Pₘ are solved in order. Medium: CLIPS, ...

- **Network representation**: Knowledge is expressed as a graph in which the nodes represent objects or concepts in the problem domain and the arcs represent relations or associations between them. Examples: Semantic networks, conceptual graphs, ...

- **Structured representation**: Extend networks by allowing each node to be a complex data structure consisting of names slots with attached values. These values may be simple numeric or symbolic data, pointers to other complex data structures or even procedures for performing a particular task. Examples: Frames, objects, ...
Knowledge Representation (4)

Issues in designing/choosing a KR scheme

1. **Expressiveness/granularity.** How to express John’s car is redder than Mary’s?
   Logic and production systems: superficial representations. Frames and objects: deep/causal representations of all the information about an object in a place, making it easier to access, update or delete.

2. **Extensibility/modularity:** Adding/removing a piece of knowledge does not affect the whole knowledge base.

3. **Clarity:** Concise and simple structures to implement efficient retrieve and inconsistency checking.

4. **Naturalness:** Easily understandable.
A production system is defined by 3 basic components:

- **A rule base** that consists of a collection of If LHS (left-hand side) then RHS (right-hand side) statements called productions, production rules, or simply rules. The LHS determines when the rule may be applied. The RHS defines the associated action.

  E.g.,
  
  If A=0, B>5 then C=max{B,10}.
  If P(a),Q(b) then test(R(a,b,c) :- P(a),Q(b),T(c)).

- **A working memory** which holds facts including the data, goal statement, and intermediate results that make up the current state of the problem in solving.

- **An inference engine** that decides when to apply which rules.
Production Systems (2)

The inference engine typically operates according to the following “recognize–act” algorithm:

1. **Match.** Find the rules in the rule base whose LHSs are satisfied from the existing contents of the working memory.

2. **Conflict resolution.** Select one rule with a satisfied LHS by applying one or more conflict resolution strategies (e.g., selecting the first rule whose LHS matches the working memory or the first satisfied rule whose RHS can produce new items); if no rules are available in the rule base, stop.

3. **Act.** Adapt the working memory according to the RHS of the selected rule, perhaps adding a new item or deleting an old one.


Data-driven search or forward chaining in the “recognize–act” cycles. Production systems can also be implemented in a goal-driven or backward chaining manner (as in Prolog systems).
Production Systems (3): An Example

Initial working memory: \( \text{wm}=(\text{green}, \text{weighs } 15 \text{ lbs}) \).

The rule base consists of 6 rules:

- **P1**: If on-wm green then put-on-wm produce.
- **P2**: If on-wm packed in small container
  then put-on-wm delicacy.
- **P3**: If on-wm refrigerated OR on-wm produce
  then put-on-wm perishable.
- **P4**: If on-wm weighs 15 lbs AND on-wm inexpensive
  AND NOT on-wm perishable then put-on-wm staple.
- **P5**: If on-wm perishable AND on-wm weighs 15 lbs
  then put-on-wm turkey.
- **P6**: If on-wm weighs 15 lbs AND on-wm produce
  then put-on-wm watermelon.

where on-wm \( X \) indicates that \( X \) exists in the working memory and put-on-wm Y means put symbol Y in the working memory.

The inference procedure: In each ‘recognize-act’ cycle,

i) Label all the rules whose LHSs are satisfied from the working memory,
ii) Delabel those rules whose RHSs are just repetition of the working memory.
iii) If no labeled rules, exit. Otherwise, choose the rule with the smallest ordinal.
iv) Clear all the labels, goto (i)

Generate the stages of the working memory.
Production Systems (4)

Cycle I:
(1) Conflict set = {P1}
(2) After delabeling {P1}
(3) Rule chosen: P1
WM=(produce,green,weighs 15 lbs)

Cycle II:
(1) Conflict set = {P1,P3,P6}
(2) After delabeling {P3,P6}
(3) Rule chosen: P3
WM=(perishable,produce,green,weighs 15 lbs)

Cycle III:
(1) Conflict set = {P1,P3,P5,P6}
(2) After delabeling {P5,P6}
(3) Rule chosen: P5
WM=(turkey,perishable,produce,green,weighs 15 lbs)
/* We get a wrong solution here: ‘‘green,produce -> turkey’’, due to inexactness in P5 */

Cycle IV:
(1) Conflict set = {P1,P3,P5,P6}
(2) After delabeling {P6}
(3) Rule chosen: P6
WM = (watermelon,turkey,perishable,produce,green, weighs 15 lbs)

Cycle V:
(1) Conflict set = {P1,P3,P5,P6}
(2) After delabeling {}
(3) Exit.
WM = (watermelon,turkey,perishable,produce,green, weighs 15 lbs)

Question: Why (green, weighs 15 lbs) ⇒ turkey?
Production Systems (5): Conflict Resolution

1. Refraction: Once a rule has fired, it may not fire again until the working memory elements that match its LHS have been modified. This discourages looping.

2. Recency: Rules whose LHSs match with the most recently added working memory items are preferred. This focuses the search on a single line of reasoning.

3. Specificity: A more specific rule (with more conjunctive conditions, for example) is preferred to a general rule. This allows these rules which match fewer potential working memory situations a better chance.

4. Heuristic control: Providing some heuristic function to evaluate the strength of each rule.
• Working memory elements: (object attribute value). E.g.,
  (Expression ^Name Exprs17 ^Arg1 2 ^Op * ^Arg2 X)
  ^ is used to distinguish attributes from values.
This WM element says that: the object of class Expression is named Exprs17, has 2 as its first argument, * as its operator, X as its second argument.

• Production rules

    (P production-name
     LHS (one or more condition patterns/elements
          (first not negated),
          each in LISP list format)
     -->
     RHS (one or more actions, each in LISP list format))

An LHS is satisfied when

1. Every pattern that is not preceded by - (negation) matches a WM element, and
2. No pattern that is preceded by - matches a WM memory.
OPS5: The Official Production System Version 5 (2)

- Patterns in LHS are partial descriptions, so
  \[(Expression \land Op \ast \land Arg2 0)\]
  matches \[(Expression \land Name Exprs86 \land Arg1 X \land Op \ast \land Arg2 0)\].

- Variables begin with ‘<’ and end with ‘>’, e.g., <X>. Variable bindings in OPS5 take the same way as in Prolog: They match any values, but all occurrences of the same variable in a LHS must match the same value.

- Predicates: =, <>, <, >, <=, >= – They are placed between an attribute and a value/variable.
  E.g., \[(Expression \land Arg1 <LEFT> \land Op <> \land Arg2 <LEFT>)\].
  The ‘=’ predicate is normally omitted.

- The RHS consists of an unconditioned sequence of actions.
  **MAKE**: builds a new WM element and adds it to WM. E.g.,
  \[(MAKE Expression \land Name Exprs1 \land Arg1 1)\].
  Other attributes not given here all have the value NIL.
OPS5: The Official Production System Version 5 (3)

- Interpreter: The RETE Match Algorithm
- The conflict set: The output of the match process and the input to conflict resolution.
  
  In OPS5:

  \[
  \text{Conflict set = } \{< \text{Production}, \text{list of WM elements matched by its LHS} >\}
  \]

- Problems with the basic ‘recognize-act’ cycles:
  
  1. The successful match of a rule with the working memory does not always mean its immediate act. Some rules may be successful in matching with the working memory from the very beginning of a problem-solving process, but always fails to get the priority of act in each conflict resolution phase. When there are changes in the working memory, it also needs to be tested again and again.
  
  2. A rule may fail to match with the working memory in an overall problem-solving process but it probably needs to be tested in each 3-phase cycle when the working memory is changed.

Conclusion: Efficiency has always been a major issue for large production systems.
RETE: An Algorithm for Computing the Conflict Set

• How to avoid iterating over the WM elements? – Store information between cycles.
  – A condition/pattern can only match a number of WM elements, e.g., only those with the same object name.
  – The iteration can be avoided by storing, with each pattern, a list of the elements that it matches.

• The lists are updated when the WM changes:
  1. When an element enters the WM, the interpreter finds all the patterns that match it and adds it to their lists.
  2. When an element leaves the WM, the interpreter again finds all the patterns that match it and deletes it from their lists.

• Descriptions of the WM changes: tokens. A token is a tag (+|-) plus a list of data elements. E.g., 2 tokens
  
  ( - (Expression ^Name Exprs41 ^Arg1 Y ^Op + ^Arg2 Y))
  ( + (Expression ^Name Exprs41 ^Arg1 2 ^Op * ^Arg2 Y))

changed the first element to the second.
RETE: An Algorithm for Computing the Conflict Set (2)

• How to avoid iterating over the set of rules in a rule base? Using a tree-structured sorting network for the rules.

• Consider a rule:

(P Time0X
  (Goal ^Type Simplify ^Object <X>)
  (Expression ^Name <X> ^Arg1 0 ^Op * ^Arg2 <Y>)
  -->
  (MODIFY 2 ^Op NIL ^Arg2 NIL))

• Intra-element features: ones that involve only one WM element.
  Intra-element features in the 2nd pattern:
  - The class of the element is Expression
  - ^Arg1 0
  - ^Op *

• Inter-element features: a variable can occur in more than one pattern. E.g.,
  ^Name <X>
  or a logical OR on 2 intra-element features.
RETE (3): The Sorting Network

- Nodes: tests for the presence of both kinds of features.
- For each LHS, the compiler (from rules to a network) begins with the intra-element features and builds a linear sequence of nodes for the pattern.
- After intra-element features, it builds nodes to test for inter-element features: each of these nodes has 2 inputs. The first of the 2-input nodes joins the linear sequences for the first two patterns, the 2nd joins the output of the first with the sequence for the 3rd pattern, and so on.
- After the 2-input nodes, the compiler builds a special terminal node to represent the production rule.
RETE (4): Processing in the Network

- The nodes to perform the intra-element tests have one input and one or more outputs, each testing one feature and sending out tokens that pass the test to its successors.
- The inter-element tests compare tokens from different paths and join them into bigger tokens if they satisfy the inter-element constraints of the LHS.
- The terminal node sends out of the black box the information that the conflict set must be changed.
The Rete Matching Algorithm

- Only the nodes affected by a newly inserted or modified fact are checked.
- For example, consider the rules:
  - IF $a(X,1)$ and $b(X,Z)$ THEN $g1(X,Z)$
  - IF $a(X,2)$ and $b(X,Z)$ THEN $g2(X,Z)$
The Rete Matching Algorithm

Initially the working memory is empty

- There is a starting node and a node for each of the rule conditions and conjunctions of conditions.
- Arcs are labeled with variable bindings.
The Rete Matching Algorithm

Fact $a(3,1)$ is added to the working memory

$a(3,1)$ is deposited in the node labeled $a(X,Y)$ and will propagate through the arc labeled $Y=1$

Rule doesn’t match
The Rete Matching Algorithm

Fact b(3,4) is added to the working memory

b(3,4) is deposited in the node labeled b(Y,Z) and will propagate through the arcs labeled Y=1 and Y=2

Rule matches

Rule doesn’t match
The Rete Matching Algorithm

Fact \( a(3,2) \) is added to the working memory

\( a(3,2) \) is deposited in the node labeled \( a(X,Y) \) and will propagate through the arc labeled \( Y=2 \)

Rule matches

\( a(X,1),b(X,Z) \)

Rule matches

\( a(X,2),b(X,Z) \)
The Rete Matching Algorithm

- The Rete algorithm (and extensions) are widely used in rule-based systems.
- It allows for an efficient matching process (on average).
- A naïve algorithm that tries all combinations of rules and facts has exponential complexity.
RETE (5): Advantages

- Avoids iterating over the elements in the WM.
- Avoids iterating over the set of rules in a rule base.
- The subparts of the sorting network for the similar patterns are sharable.
Features of Production Systems

1. **Modularity of rules.** No direct interactions between rules, but via the working memory. This supports the incremental development of production systems by successively adding, deleting, or changing the knowledge (rules) of the system.

2. **Efficient expression of human problem solving.** The “If ... then” structure is natural and concise.

3. **Separation of knowledge and control.** Rules and the “recognize-act” algorithm are independent. This makes programming much easier.

4. **Tracing and explanation.** Selected rules can be displayed to show how a result has been achieved.
CLIPS - the C Language Integrated Production System

Quiz: A Rete Network for Watermelon-Vs-Turkey

- The network is independent of a specific WM state.
- Every rule has a place in the network.
- WM element testing can be shared by alpha memories.
**LFA: Linear Forward-chaining Algorithm**

A Reasoning Network for Watermelon vs. Turkey

LFA:
Linear Forward-Chaining Algorithm
to avoid these problems & achieve linear time complexity

Wu, Expert Systems, 1993
Wu et al., Informatica, 1998
Sorting Rules with LFA

1. Find and remove dead cycles.
   - A cycle like *if A then B, if B then C and if C then A* is a dead cycle if none of *A, B* and *C* is a leaf node.

2. Renumber all rules which have all of their premise factors being leaf nodes.
   - For any factor *F*, if all rules with it as their conclusion factor have been renumbered, then it is treated as leaf factor for further renumbering.

3. If all rules have been renumbered, goto Step 4; otherwise, resolve a live cycle and goto Step 2.
   - A live cycle is not a dead cycle.

4. Stop.
Assignment: Production Systems

Given the following rule base,

P1: savings(poor) -> invest(savings).
P2: savings(good) & income(good) -> invest(stocks).
P3: savings(good) & income(poor) -> invest(savings) & invest(stocks).
P4: saved(X) & dependents(Y) & (X > minsav(Y)) -> savings(good).
P5: saved(X) & dependents(Y) & not(X > minsav(Y)) -> savings(poor).
P6: earnings(X, steady) & dependents(Y) & (X > mininc(Y)) -> income(good).
P7: earnings(X, steady) & dependents(Y) & not(X > mininc(Y)) -> income(poor).
P8: earnings(X, unsteady) -> income(poor).
P9: dependents(X) -> minsav(X) = 5000 * X.
P10: dependents(X) -> mininc(X) = 1500 + (4000 * X).

Write up a production system to implement the following functions:

1. Read facts from a data file, facts.db, in forms of saved(real), earnings(real,symbol), and dependents(integer).
2. Display on the terminal the rules being fired one by one and the results produced by each of them.
Example Rete Network for Investment

WM: 5 saved (22,000),
WM2: earnings (25,000, steady),
WM3: dependents (3)

X = poor
X = good

Y = bad
Y = good

Savings (X)
Income (Y)
Saved (S)
Dependents (S)
Earnings (T, Q)

Q = steady
Q = unsteady

P1
P2
P3

P4
P5
P6
P7
P8

Conflict Set
Example LFA Network for Investment.

- **Savings (L)**
  - P4, P5
  - min sav (X)
  - P9 #1

- **Invest (K)**
  - P1, P2, P3
  - #8, #9, #10

- **Income (S)**
  - P6, P7, P8
  - #5, #6, #7

- **Dependents (X)**
  - P10 #2

- **Earnings (Y, Z)**
Reasoning under Uncertainty – Sources of Noise

- Predicate calculus: from correct premises, sound inference rules produce new, guaranteed correct conclusions. This is not always the case with realistic applications.

- In knowledge-based systems (expert systems), we often attempt to draw correct conclusions from poorly formed and uncertain evidence using unsound inference rules.

- Examples of noise sources:
  - Ambiguous symptoms/evidence: The system structure is good.
  - Inexact knowledge: An army puffed up with pride is bound to lose.
  - Inexact (abductive) reasoning: $P \implies Q$ and $Q$ it is possible to infer $P$.

- Uncertainty processing in AI: Certainty Factor (CF).
Key Components in Inexact Reasoning

1. A measure (e.g., a probability or a fuzzy degree) to describe imperfect data.

2. A measure (e.g., a conditional probability or a rule strength) to represent imperfect rules.

3. An inexact model which contains a set of computing formulae to

   (a) Evaluate the certainty factor of each conclusion in the RHS of a rule according to the certainty factors of all the conditions (and their conjunction, disjunction and negation) in the LHS of the rule

   (b) Compute the certainty factor of a conclusion which is supported by a set of rules.
Typical Models for Inexact Reasoning (1)

1. **Bayesian Probability Theory.**
   - Description of evidence and knowledge: probability.
   - Computing the CF value of a conclusion: the Bayes’ theorem about conditional probability.

\[
P(H_j|E) = \frac{P(E|H_j) \times P(H_j)}{\sum_{k=1}^{n} (P(E|H_k) \times P(H_k))}
\]

(see Chapter 13 of AIME2e and the book authors’ notes on the course home page)

2. **Certainty Factor Algebra.**
   - Description of evidence: probability.
   - Description of knowledge: $\text{CF}(H|E) = \text{MB}(H|E) - \text{MD}(H|E)$ where $\text{MB}(H|E)$ is the measure of belief of a hypothesis $H$ given evidence $E$, and $\text{MD}(H|E)$ is the measure of disbelief of a hypothesis $H$ given evidence $E$.
   - Computing the CF value of a conclusion from 2 premises (P1 and P2):
     - $\text{CF}(\text{P1 and P2}) = \text{MIN}($CF(P1), CF(P2)$)$
     - $\text{CF}(\text{P1 or P2}) = \text{MAX}($CF(P1), CF(P2)$)$
   - Combine multiple CFs when 2 or more rules support the same result $R$:
     - $\text{CF}(R1) + \text{CF}(R2) - \text{CF}(R1) \times \text{CF}(R2)$ when $\text{CF}(R1) > 0, \text{CF}(R2) > 0$;
     - $\text{CF}(R1) + \text{CF}(R2) + \text{CF}(R1) \times \text{CF}(R2)$ when $\text{CF}(R1) < 0, \text{CF}(R2) < 0$;
     - $\frac{\text{CF}(R1) + \text{CF}(R2)}{1 - \text{MIN}(|\text{CF}(R1)|, |\text{CF}(R2)|)}$ otherwise.
Typical Models for Inexact Reasoning (2)

3. Fuzzy Logic.

- Description of evidence: a set membership function that takes on real values between 0 and 1 to describe the possibility of vagueness.
- Fuzziness vs probability: vague vs undeterministic
- Linguistic variables and degrees of truth
- Computing the CF value of a conclusion from 2 premises (P1 and P2):
  \[ CF(P_1 \text{ and } P_2) = \text{MIN}(CF(P_1), CF(P_2)) \]
  \[ CF(P_1 \text{ or } P_2) = \text{MAX}(CF(P_1), CF(P_2)) \]
Typical Models for Inexact Reasoning (3)


- Fundamental distinction between uncertainty and ignorance.
- Create “belief functions” which satisfy axioms that are weaker than those of probability theory. – Probability theory is a subclass of belief functions.
- Belief functions allow us to use our knowledge to bound the assignment of probabilities to events without having to come up with exact probabilities, when these maybe unavailable.
Typical Models for Inexact Reasoning (4)

5. Nonmonotonic Logic and Reasoning with Beliefs. A qualitative approach to modeling inference based on beliefs and assumptions.

- A nonmonotonic system addresses the problem of changing beliefs. It handles uncertainty by making the most reasonable assumptions in light of uncertain information.
- It proceeds with its reasoning as if these assumptions were true.
- At a later time, a belief may change, necessitating a reexamination of conclusions derived from that belief.
- Truth maintenance systems: attempt to reduce the complexity of revising the conclusions of the reasoner in light of changing beliefs, by storing the justifications for each inference.

* Monotonic reasoning: begins with a set of axioms, assumed to be true, and infers their consequences. If we add a new axiom or fact to such a system, it may cause the set of true statements to increase; however, adding knowledge will never make the set of true statements decrease.
Logic vs Associationist Theories

- Associationist theories: They define the meaning of an object in terms of a network of associations with other objects in a mind or a knowledge base. Basic components: nodes and associations.

- Logical representations: emphasize truth-preserving operations on well-formed expressions.

- \((2+2=5) \rightarrow \text{color(elephants, green)}\) is true in logic, but meaningless in associationist theories.

- Logic is preferred by philosophers and mathematicians while associationist theories are preferred by psychologists.
**Semantic Networks**

- A semantic network represents knowledge as a graph, with the nodes corresponding to facts or concepts and the arcs to relations or associations between concepts. Both nodes and arcs are generally labeled.

- Network representation of properties of `robin` and `bird` and `clyde is a robin`: nodes (such as `clyde`, `robin`, `bird`, `ownership` and `time`) and arcs with labels such as `instance-of/is-a`, `owner` and `start-time`.

- Inheritance in semantic networks. E.g., `frosty is a snowman` so inherits all the properties of snowman. We don’t need to store these properties specifying at node `frosty`.

- Link types in logical terms: $\land$ (`and`), $\lor$ (`or`), $\neg$ (`not`), $\Rightarrow$, $=$

- The power of network representation: (a) spreading activation: definitions of links and associated inference rules, and (b) inheritance.

- How to implement a semantic network in Prolog?

- Advantages and disadvantages?
Semantic Networks: Examples

State: I own a tan leather chair.

Event: John gives the book to Mary.

http://www.cse.unsw.edu.au/~billw/cs9414/notes/kr/frames/frames.html
Conceptual Graphs

• Nodes in conceptual graphs are either concepts (represented as boxes) or conceptual relations (represented as ellipses). Labeled arcs are not used; instead the conceptual relation nodes represent relations between concepts.
• Concept nodes represent either concrete or abstract objects.
• Graphs may be arbitrarily complex but must be finite.
Frames

• A frame is a static data structure used to represent well-understood, stereotyped situations. It organizes our knowledge of the world based on past experiences. We can revise the details of these past experiences to represent the individual differences for new situations.

• A Backus-Naur-Form description for a frame (e.g., a hotel frame).

• A frame consists of a number of slots with attached values; pointers to other frames; or attached procedures for performing some function to get some values.

• Default information: More information: call the receptionist.

• Filling a slot in a frame: (a) query, (b) default, (c) inheritance, or (d) attached procedures.

• A comparison between frames in AI and O-O in software engineering.
A Comparison between Frames and Semantic Networks

- Frames make it easier to organize knowledge hierarchically. We can describe an object with its various attributes and think of the object as a single entity for some purposes and only consider details of its internal structure for other purposes.
- **Procedural attachment** is a particularly important feature. We use procedural attachment to create *demons*, which are procedures that are invoked as a side effect of some other action in the knowledge base.
- Frame systems support **class inheritance** explicitly via default values and slots.
- **Semantic Networks:** See Overhead 35.