cis20.2
design and implementation of software applications II
spring 2008
session # III.3
knowledge representation

topics:

- knowledge representation
- production systems
- semantic networks
- frames
- scripts
- graph representations
- bayesian networks

knowledge representation

- Marvin Minsky (1963):

  "AI is the science of making machines do things that would require intelligence if done by [humans]"
- today we focus on *knowledge representation (KR)* using networks, including probabilistic networks to represent relationships between pieces of knowledge
- *knowledge representation* focuses on designing specific mechanisms for representing the knowledge in ways that are useful for AI techniques
- Ed Feigenbaum's Knowledge Principle:

  "... power exhibited ... is primarily a consequence of the specialist knowledge employed by the agent and only very secondarily related to ... the power of the [computer]"
  "Our agents must be knowledge rich, even if they are methods poor."

the role of knowledge

- knowledge about a domain allows problem solving to be focused, so that we don't have to perform exhaustive search (i.e., for an agent to decide what to do)
- *explicit* representations of knowledge allow a *domain expert* to understand the knowledge a system has, add to it, edit it, and so on—this is called *knowledge engineering*
- comparatively simple algorithms can be used to *reason* with the knowledge and derive "new" knowledge from it
- so, how do we represent knowledge in a form that a computer can use?
- here's a list of desirable features:
  - *representational adequacy*
    - a KR scheme must be able to actually represent the knowledge appropriate to our problem
    - some KR schemes are better at some sorts of knowledge than others
    - *there is no one ideal KR scheme!*
  - *inferential adequacy*
    - a KR scheme must allow us to make new *inferences* from old knowledge

    - it must make inferences that are:
      - *sound*—the new knowledge actually does follow from the old knowledge
        for example, given two statements:
        (1) Michael is a man.
        (2) All men are mortal.
        the inference "Simon is mortal" is not sound,
        whereas the inference "Michael is mortal" is sound.
      - *complete*—it should make all the right inferences
    - soundness is usually easy to comply with; but completeness is very hard!
  - *inferential efficiency*
    - a KR scheme should be *tractable*—i.e., one should be able to make inferences from it in reasonable (polynomial) time
    - unfortunately, *any* KR scheme with interesting *expressive power* is not going to be efficient
    - often, the more *general* a KR scheme is, the *less efficient* it is...
    - so one ends up using KR schemes that are tailored to problem domain; these are less general, but more efficient
  - *well-defined syntax and semantics*
    - a KR scheme should have *well-defined syntax*—i.e., it should be possible to tell:

· whether any construction is "grammatically correct"

· how to read any particular construction—no *ambiguity*

* a KR scheme should have *well-defined semantics*—it should be possible to precisely determine, for any given construction, exactly what its meaning is
* syntax is easy; semantics is hard!

– *naturalness*

* ideally, a KR scheme should closely correspond to our way of thinking, reading, and writing
* a KR scheme should allow a *knowledge engineer* to read and check the *knowledge base* (i.e., the database that contains the knowledge represented)
* again, the *more general* a KR scheme is, the less likely it is to be readable and understandable...

production systems

- knowledge can be represented as a collection of *production rules*
- each rule has the form:
  $condition \longrightarrow action$
  which may be read
  if $condition$ then $action$
  where the $condition$ (i.e., antecedent) is a *pattern*
  and the $action$ (i.e., consequent) is an operation to be performed if rule *fires* (i.e., when the rule is true)
- a "production system" is essentially a list of rules and it is used by matching a system's state against the conditions in the list
- an action can be something the agent does, or it can be an action that manipulates the agent's memory, such as adding a rule to its database
- the mechanism that fires rules is frequently called an *inference engine*
- example:

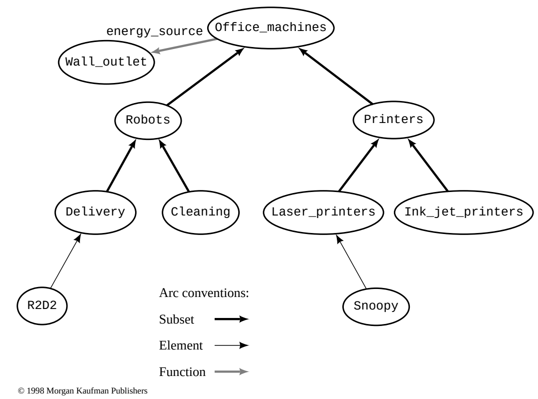

```
R1:  IF user has feathers
     THEN animal is a bird
        R2:  IF animal is a bird
     THEN animal can fly
R3:  IF animal can fly
     THEN animal is not scared of heights
```

- given a set of rules like these, there are essentially two ways we can use them to generate new knowledge:
  - *forward chaining*—data driven: reasoning forward from conditions to actions
  - *backward chaining*—goal driven: reasoning backward from goals back to facts
- sometimes the system has to make choices about which rules should fire, if more than one is applicable at a given time—this is called "conflict resolution", and there are a number of approaches to this situation:
  - most specific rule first (with most matching antecedents)
  - most recent first (i.e., rule that became true most recently)
  - user/programmer specified priorities
  - use "meta-knowledge" (i.e., knowledge about knowledge..., which can be encoded into the system)

semantic networks

- another way of representing knowledge is using *network* structures
- a *semantic network* is a "labelled graph" in which
  - nodes represent objects, concepts, or situations (states)
  - links represent relationships between objects
  - inference occurs by traversing links
- key types of links:
  - $x \xrightarrow{subset} y$ means "x is a kind of y" ($\subset$)
    e.g., $penguin \xrightarrow{subset} bird$
  - $x \xrightarrow{member} y$ means "x is a y"
    e.g., $opus \xrightarrow{member} penguin$
  - $x \xrightarrow{R} y$ means "x is R-related to y"
    e.g., $bill \xrightarrow{friend} opus$

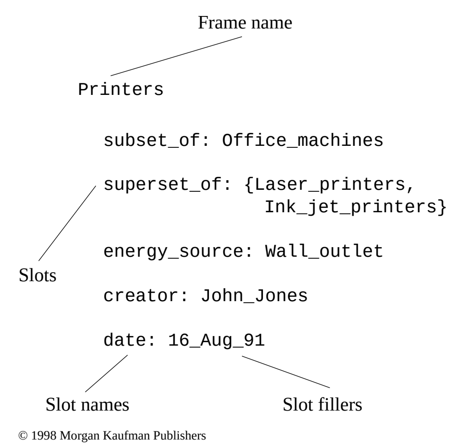
- *binary* relations are easy and natural to represent
- others kinds of relation are harder
- unary relations (properties), e.g., "Opus is small"
- three place relations, e.g., "Opus brings tequila to the party"
- some binary relations are problematic, e.g., "Opus is larger than Bill"
- *quantified* statements are very hard for semantic nets, e.g., "every dog has bitten a postman" or "every dog has bitten every postman" (whereas these types of statements are easy to represent in first-order logic)
- however, *partitioned* semantic nets can represent these
- example semantic net:



frames

- *frames* are a kind of structured knowledge representation mechanism
- all information relevant to a particular concept is stored in a "frame", which resembles something like an object in C++
- each frame has a number of "slots"
- each slot may be "filled" by:
  - a value,
  - a pointer (reference) to another frame,
  - or a procedure (or function or method)
- slots may have default values associated with them
- frames are typically used to represent the properties of objects, and the relationships between them
- frames may represent: generic concepts (e.g., classes) or specific items (e.g., objects)
- the most important kind of link between frames is the `is-a` relationship, which facilitates reasoning about object properties and inheritance

- example:



- a computer can reason with frames simply by following the `is-a` links

scripts

- *scripts* are a variant of frames, for representing stereotypical sequences of events
- a script is thus a frame with a set of prescribed slots, for example:
  - some initial conditions
  - some final conditions
  - some state description
  - some actions
  - some actors
- the structure of the script is heavily domain dependent, so some people think it is not a very useful kind of system
- famous example: the restaurant script

graph representations

- a *directed graph* is a set of variables and a set of directed arcs between them
- a directed graph is *acyclic* if it is not possible to start at a node, follow the arcs in the direction they point, and end up back at the starting node
- we will only talk about directed acyclic graphs
- A is the *parent* of B if there is a directed arc from A to B
- B is the *child* of A if A is the parent of B
- any node with no parents is known as a *root* of the graph
- any node with no children is known as a *leaf* of the graph
- the parents of node A and the parents of those parents, and the parents of those parents, and so on, are the *ancestors* of A
- if A is an ancestor of B, then B is a *descendent* of A
- we say that there is a *link* between two nodes A and B if A is a parent of B or B is a parent of A

- two nodes A and C in a graph have a *path* between them if it is possible to start at A and follow a series of links through the graph to reach C
- note that in defining a path we ignore the direction of the arrows; in other words we are considering the underlying undirected graph
- a graph is said to be *singly-connected* if it includes no pairs of nodes with more than one path between them
- a graph which is not singly-connected is *multiply-connected*
- a singly-connected graph with one root is called a *tree*
- a singly-connected graph with several roots is called a *polytree*
- a polytree is sometimes called a *forest*

Bayesian networks

- a Bayesian network is defined by:

  Definition 1.1 *A* Bayesian network *is a directed acyclic graph where each variable is a random variable with a finite set of mutually exclusive states. For every variable A there is a probability function $\Pr(A \mid B_1, \ldots, B_n)$ where $B_1, \ldots, B_n$ are the parents of A.*

  Note that for root nodes the probability function is just $\Pr(A)$.
- Bayesian networks are also known as probabilistic causal networks.
- we also have:

  Definition 1.2 *Given random variables A, B and C, A and C are* conditionally independent given B*, if:*
  $$\Pr(A \mid B) = \Pr(A \mid B, C)$$
  If B is empty, then A and C are independent in the sense we are used to.
- applying Bayes' rule this means that:
  $$\Pr(C \mid B, A) = \frac{\Pr(A \mid C, B) \Pr(C \mid B)}{\Pr(A \mid B)}$$

$$= \frac{\Pr(A \mid B)\Pr(C \mid B)}{\Pr(A \mid B)}$$
$$= \Pr(C \mid B)$$

- finally, we have:

  Definition 1.3 *Two variables in a causal network are* d-separated *if, for all paths between A and B there is an intermediate variable V such that:*

  1. *The connection is serial or diverging and there is hard evidence for V; or*
  2. *The connection is converging and there is no evidence for either V or any of its descendants.*

- Now the neat thing about Bayesian networks is that:

  Theorem 1.1 *If A and B are d-separated in a Bayesian network G, and evidence e is entered, then:*
  $$\Pr(A \mid B, e) = \Pr(A \mid e)$$

- it is easy to spot conditional independencies from the graphical representation of the Bayesian network
- it also makes it easy to show that:

  Theorem 1.2 *Given a graph which includes two d-separated nodes A and B, then changes in the probability of A have no impact on the change in probability of B.*

  and this, in turn makes it easy to devise algorithms for computing the probabilities of variables as evidence is obtained.

- another consequence of Theorem ?? is that the joint probability over all the variables in a Bayesian network is just the product of all the conditional probabilities in the
- in other words

  Theorem 1.3 *If G is a Bayesian network which includes only the set of variables $U = \{A_1, \ldots, A_m\}$, then the joint probability distribution over U is:*
  $$\Pr(U) = \prod_i \Pr(A_i \mid pa(A_i))$$
  where $pa(A_i)$ is the set of parents of A_i.

- what this means from the computational side is that provided we start at the root nodes, for which we have unconditional probabilities, we can conceive of a message passing algorithm to compute $\Pr(U)$.