today's topics:
  • knowledge representation
Introduction

- Using logic is one approach to knowledge representation.
- Another possibility is to design specific mechanisms for representing the kind of knowledge we need in AI.
- Leads to an area of AI called *knowledge representation*.
- This lecture will look at some general aspects of knowledge representation, and also the specific example of production rules.
The Knowledge Principle

- Ed Feigenbaum:

  “... 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 *focussed* — not necessary to exhaustively search.
- *Explicit* representations of knowledge allow a *domain expert* to understand the knowledge a system has, add to it, edit it, and so on.
  
  *Knowledge engineering.*

- Comparatively *simple* algorithms can be used to *reason* with the knowledge and derive *new* knowledge.
Knowledge Representation

• Question: How do we represent knowledge in a form amenable to computer manipulation?

• Desirable features of KR scheme:
  – representational adequacy;
  – inferential adequacy;
  – inferential efficiency;
  – well-defined syntax & semantics;
  – naturalness.
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

- 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;
  - *complete* — it should make all the right inferences.
- Soundness usually easy; completeness very hard!
Example. Given knowledge...

* Michael is a man.
* All men are mortal.

the inference

* Simon is mortal.

is not sound, whereas

* Michael is mortal.

is sound.
Inferential Efficiency

• A KR scheme should be *tractable* — make inferences 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.
• Use KR schemes tailored to problem domain — less general, but more efficient.
• (Any KR scheme with expressive power = first-order logic is *undecidable*.)
Syntax and Semantics

• It should be possible to tell:
  – whether any construction is “grammatically correct”.
  – how to read any particular construction — no ambiguity.
Thus KR scheme should have well defined syntax.

• It should be possible to precisely determine, for any given construction, exactly what its meaning is.
  Thus KR scheme should have well defined semantics.

• Syntax is easy; semantics is hard!
Naturalness

• Ideally, KR scheme should closely correspond to our way of thinking, reading, and writing.
• Allow knowledge engineer to read & check knowledge base.
• Again, more general a KR scheme is, less likely it is to be readable & understandable.
Rules

- Knowledge is specified as a collection of *production rules*.
- Each rule has the form
  
  \[ \text{condition} \rightarrow \text{action} \]

  which may be read
  
  if \text{condition} then \text{action}.

- The *condition* (antecedent) is a *pattern*.
- The *action* (consequent) is an operation to be performed if rule *fires*. 
• A rule-based (production) system has a *working memory* of *facts* against which *condition* is matched.

• Action is often a *fact* to be added to working memory.

• Rule fires if match is successful; Mechanism that fires rules is *inference engine*.

![Diagram of a rule-based system](cis32-spring2006-sklar-lec14)
• Example rule base:

R3: IF animal has feathers
    THEN animal is a bird

R4: IF animal is a bird
    THEN animal can fly

R5: IF animal can fly
    THEN animal is not scared of heights
Relation to search

- Using rules can be thought of as just another form of search.
- Facts are states.
- Working memory is the agenda.
- Rules are the operations on states.
- This suggests that there are schemes for applying rules which are similar to breadth-first search etc.
- We will look at these next.
• Another example:

R1: IF animal has hair
    THEN animal is a mammal

R2: IF animal gives milk
    THEN animal is mammal

R3: IF animal has feathers
    THEN animal is a bird

R4: IF animal can fly
    AND animal lays eggs
    THEN animal is bird
R5: IF animal eats meat
    THEN animal is carnivore

R6: IF animal has pointed teeth
    AND animal has claws
    THEN animal is carnivore

R7: IF animal is mammal
    AND animal has hoofs
    THEN animal is ungulate

R8: IF animal is mammal
    AND animal chews cud
    THEN animal is ungulate
R9: IF animal is mammal
   AND animal is carnivore
   AND animal has tawny colour
   AND animal has dark spots
   THEN animal is cheetah

R10: IF animal is mammal
     AND animal is carnivore
     AND animal has tawny colour
     AND animal has black stripes
     THEN animal is tiger
R11: IF animal is ungulate
   AND animal has long legs
   AND animal has dark spots
   THEN animal is giraffe

R12: IF animal is ungulate
   AND animal has black stripes
   THEN animal is zebra

R14: IF animal is bird
   AND animal does not fly
   AND animal has long legs
   AND animal has long neck
   THEN animal is ostrich
R14: IF animal is bird
  AND animal does not fly
  AND animal can swim
  AND animal is black and white
  THEN animal is penguin

R15: IF animal is bird
  AND animal is good flyer
  THEN animal is albatross
Forward Chaining

- Given a set of rules like these, there are essentially two ways we can use them to generate new knowledge:
  - *forward chaining* — data driven;
  - *backward chaining* — goal driven.

- In what follows...
  let \((c, a)\) be a rule.
  let \(\text{fires}(c, WM)\) be true if condition \(c\) fires against working memory \(WM\).

- Forward chaining algorithm is as follows.
var WM : set of facts
var goal : goal we are searching for
var RuleBase : set of rules
var firedFlag : BOOLEAN
repeat
    firedFlag = FALSE
    for each (c,a) in RuleBase do
        if fires(c,WM) then
            if a == goal then return success
            end-if
            add a to WM
            set firedFlag to TRUE
        end-if
    end-for
until firedFlag = FALSE
return failure
• Example. Suppose

\[
WM = \{ \text{animal has hair,} \\
\text{animal eats meat,} \\
\text{animal has tawny colour,} \\
\text{animal has dark spots}\}
\]

and goal is

\[
\text{animal is cheetah}
\]
• Note that *all rules which can fire do fire.*
• Can be inefficient — lead to spurious rules firing, unfocussed problem solving (cf. breadth-first search).
• Set of rules that can fire known as *conflict set.*
• Decision about which rule to fire — *conflict resolution.*
• Number of strategies possible (cf. heuristic search):
  - *most specific rule first* (with most antecedents).
  - *most recent first;*
  - *user specified priorities.*
Meta Knowledge

• Another solution: meta-knowledge, (i.e., knowledge about knowledge) to guide search.

IF
   conflict set contains any rule (c,a) such that
   a = ‘‘animal is mammal’’
THEN
   fire (c,a)

• So meta-knowledge encodes knowledge about how to guide search for solution.

• Explicitly coded in the form of rules, as with “object level” knowledge.
Backward Chaining

- Backward chaining means reasoning from \textit{goals} back to \textit{facts}.
- The idea is that this focusses the search.
- Thinking of the rules as building a tree connecting facts, . . .
- . . . in backward chaining, every path ends with the goal.
- Since, in general, there are more initial facts that goals, . . .
- . . . more of the paths built will be solutions than in forward chaining (we hope :-).
var WM : set of facts
var RuleBase : set of rules
var firedFlag : BOOLEAN
function prove(g : goal)
    if g in WM then
        return TRUE
    if there is some (c,a) in WM 
        such that a == g then 
            for each precondition p in c do
                if not prove(p,WM) then return FALSE
            return TRUE
    else
        return FALSE
end-function
Example. Suppose

\[ WM = \{ \text{animal has hair,} \]
\[ \quad \text{animal eats meat,} \]
\[ \quad \text{animal has tawny colour,} \]
\[ \quad \text{animal has dark spots} \} \]

\[ \text{and goal is} \]

\[ \text{animal is cheetah} \]
Semantic Networks

- Taxonomic reasoning can be more efficient not in logic.
- Developed by Quillian in 1968, for semantic memory.
- Models the “associations” between ideas that people maintain.
- Semantic net is a labelled graph.
  - nodes in graph represent objects, concepts, or situations;
  - arcs in graph represent relationships between objects.
Key types of arc:

- $x \xrightarrow{\text{subset}} y$
  
  “$x$ is a kind of $y$” ($\subset$)

  Example: $penguin \xrightarrow{\text{subset}} bird$

- $x \xrightarrow{\text{member}} y$
  
  “$x$ is a $y$”

  Example: $opus \xrightarrow{\text{member}} penguin$

- $x \xrightarrow{R} y$
  
  “$x$ is $R$-related to $y$”

  Example: $bill \xrightarrow{\text{friend}} opus$

- Inference is then by traversing arcs.
Got yer WE-DOH filled out?

ACK!

Here, maym.

How many?

ACK!

73. He's had a BAD month.

Are you pleding a joint claim?

AS HIS GUARDIAN, I'M ENTITLED TO FUNDS.

Say / there's that word again.

We're entitled to that...

PROBABLY why there are in such a haggledy-piggledy!

Take! Take! Take! Take! Take!

We've got to break the cycle!!

You don't want your check? Gonna have to Gonna have to Gonna have to

FEDERAL
HAIRBALL SUPPORT
AND ASSISTANCE CENTER
• *Binary* relations are easy and natural to represent.
• Others kinds of relation are harder.
• Unary relations (properties).
  Example: “Opus is small”.
• Three place relations.
  Example: “Opus brings tequila to the party.”
• Some binary relations are problematic . . .
  “Opus is larger than Bill.”
• *Quantified* statements are very hard for semantic nets.

Examples:

– “every dog has bitten a postman”
– “every dog has bitten every postman”

• *Partitioned* semantic nets can represent these.

• Of course, expressions like this are very easy to represent in first order logic.
Example semantic net:

- **Office_machines**
  - **energy_source**
    - **Wall_outlet**
  - **Robots**
    - **Delivery**
    - **Cleaning**
  - **Printers**
    - **Laser_printers**
    - **Ink_jet_printers**
  - **R2D2**
  - **Snoopy**

Arc conventions:
- Subset
- Element
- Function

© 1998 Morgan Kaufman Publishers
Frames

- Frames are a kind of \textit{structured} knowledge representation mechanism.
- All information relevant to a particular concept is stored in \textit{frame} which resembles C \texttt{struct}, PASCAL record, Java object...
- Each frame has a number of \textit{slots}.
- Each slot may be \textit{filled} by:
  - a value;
  - a pointer to another frame;
  - a procedure.
- Slots may have \textit{default values} associated with them.
- Frames = OO!
• Frames are typically used to represent the *properties* of objects, and the relationships between them.

• Frames may represent:
  – *generic concepts* (cf classes) or
  – *specific items* (cf objects).

• Most important kind of link between frames:

  \[\text{is-a}\]

• Facilitates reasoning about object properties.

• Allows *default values* to be *inherited*. 
- Example frame system:

```
Frame name

Printers

subset_of: Office_machines

superset_of: {Laser_printers, Ink_jet_printers}

energy_source: Wall_outlet

creator: John_Jones

date: 16_Aug_91
```

© 1998 Morgan Kaufman Publishers
• How to reason with frame systems?
• Easy to answer questions such as
  \[ is \ x \ a \ y? \]
  Simply follow the is-a links.
• Example: Is snoopy a laser printer.
• (Problem of multiple inheritance — Nixon diamond.)
• Also useful for default reasoning.
  Simply inherit all default values that are not explicitly provided.
• Example: Does snoopy the printer have a wall outlet?
• *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; and
  – Some actors

• The structure of the script is heavily domain dependent.
Example:

SCRIPT
Name: RESTAURANT
Roles: Customer, Waiter, Cook, Cashier
Entry condition: Customer is hungry
Props: Food, table, money, menu, tip
Events:
  1/ Customer enters restaurant
  2/ Customer goes to table
  3/ Waiter brings menu
  4/ Customer orders food
  5/ Waiter brings food
  6/ Customer eats food
  ...

... 

10/ Customer leaves restaurant

Main concept: 6

Results: Customer not hungry,  
Customer has less money,  
Restaurant has more money,  
Waiter gets tip
• Scripts developed by Roger Schank for *understanding stories*.
• Used to help *understand language*.
• Scripts provide *context* information without which sentences cannot be understood:
  – sentences are not unconstrained sequences of words;
  – stories are not unconstrained sequences of sentences.
• Schank developed SAM (Script Applier Mechanism) that could *fill in gaps* in stories.
• Also able to “explain” elements of stories, e.g., people get upset or angry when story deviates from script.
Problems with Frames & Semantic Nets

- Both frames and semantic nets are essentially *arbitrary*.
- Both are useful for representing certain sorts of knowledge.
- But both are essentially *ad hoc* — lack precise meaning, or *semantics*.
- Inference procedures poorly defined & justified.
- The *syntax* of KR scheme is *irrelevant*.
- *Logic* generalises these schemes... and that is both an advantage and a disadvantage.
Summary

- This lecture has introduced the idea of knowledge representation, and some of the requirements of a knowledge representation scheme.
- We also looked at several knowledge representation schemes:
  - production rules
  - semantic nets
  - frames
  - scripts