CHAPTER-3

KNOWLEDGE REPRESENTATION

- Types of Knowledge
- Knowledge Pyramid
- Knowledge Representation Methods
  - Production Rules
  - Semantic Nets
  - Schemata and Frames
  - Logic
Definitions of Knowledge

a)
(1) the fact or condition of knowing something with familiarity gained through experience or association
(2) acquaintance with or understanding of a science, art, or technique

b)
(1) the fact or condition of being aware of something
(2) the range of one's information or understanding

c) the circumstance or condition of apprehending truth or fact through reasoning : cognition

d) the fact or condition of having information or of being learned

Types of Knowledge

• a priori knowledge (theoretical knowledge)
  – comes before knowledge perceived through senses
  – considered to be universally true
  – (e.g., coin flip will give 50% heads and 50% tails)

• a posteriori knowledge (empirical knowledge)
  – knowledge verifiable through the senses
  – may not always be reliable
  – (e.g., 100 coin flips give only 39 heads - what can you conclude?)

• procedural knowledge
  – knowing how to do something

• declarative knowledge
  – knowing that something is true or false

• tacit knowledge
  – unconscious knowledge not easily expressed by language
Knowledge in Expert Systems

Conventional Programming

Knowledge-Based Systems

Algorithms + Data Structures = Programs

Knowledge + Inference = Expert System

N. Wirth

Knowledge Pyramid

Meta
Knowledge
Information
Data
Noise
Knowledge Representation Methods

1. Production Rules
2. Semantic Nets
3. Schemata and Frames
4. Logic

1. Production Rules

- Frequently used to formulate the knowledge in expert systems.
- A formal variation is Backus-Naur form (BNF).
  - metalanguage for the definition of language syntax
  - a grammar is a complete, unambiguous set of production rules for a specific language
  - a parse tree is a graphic representation of a sentence in that language
  - provides only a syntactic description of the language
    - not all sentences make sense
Example: Production Rules
(for a subset of the English language)

Grammar
<sentence> -> <subject> <verb> <object> <modifier>
<subject> -> <noun>
<object> -> <noun>

Lexicon
<noun> -> man | cat | water
<verb> -> drinks | walks
<modifier> -> always | sometimes

Example: Parse Tree

- Example sentence:
  Man drinks water always.
Example in CLIPS: This program produces all possible sentences according to the production rules (i.e. grammar), although most of them will not be meaningful.

(deffacts lexicon
(noun man)
(noun cat)
(noun water)
(verb drinks)
(verb walks)
(adverb always)
(adverb sometimes))

(defrule sentence-rule
(subj ?s)
(verb ?v)
(obj ?o)
(adverb ?d)
=>
(assert (sentence ?s ?v ?o ?d)))

(defrule subject-rule
(noun ?n)
=>
(assert (subj ?n)))

(defrule object-rule
(noun ?n)
=>
(assert (obj ?n)))

Example in CLIPS:
This program produces all possible sentences according to the production rules (i.e. grammar), although most of them will not be meaningful.

(deffacts lexicon
(noun man)
(noun cat)
(noun water)
(verb drinks)
(verb walks)
(adverb always)
(adverb sometimes))

(defrule sentence-rule
(subj ?s)
(verb ?v)
(obj ?o)
(adverb ?d)
=>
(assert (sentence ?s ?v ?o ?d)))

(defrule subject-rule
(noun ?n)
=>
(assert (subj ?n)))

(defrule object-rule
(noun ?n)
=>
(assert (obj ?n)))
Advantages and Disadvantages of Production Rules

**Advantages:**
- simple and easy to understand
- straightforward implementation
- formal foundations for some variants

**Disadvantages:**
- simple implementations are very inefficient
- some types of knowledge are not easily expressed in such rules
- large sets of rules become difficult to understand and maintain

2. Semantic Nets

- graphical representation for propositional information (shallow knowledge structure)
- originally developed by M. R. Quillian as a model for human memory
- labeled, directed graph
- nodes represent objects, concepts, or situations
  - labels indicate the name
  - nodes can be instances (individual objects) or classes (generic nodes)
- links represent relationships
  - the relationships contain the structural information of the knowledge to be represented
  - the label indicates the type of the relationship
Example in CLIPS

(deffacts initial-facts-of-semantic-net
    (is-a Asterix Gaul)
    (is-a Obelix Gaul)
    (is-a Abraracourcix Gaul)
    (is-a Cetautomatix Gaul)
    (is-a Panoramix Gaul)
    (is-a Idefix Dog)
    (lives-with Dog Human)
    (AKO Gaul Human)
    (is-boss-of Abraracourcix Asterix)
    (is-boss-of Abraracourcix Cetautomatix)
    (is-friend-of Asterix Obelix)
    (buys-from Panoramix Cetautomatix)
    (sells-to Ordralphabetix Panoramix)
    (fights-with Cetautomatix Ordralphabetix)
    (takes-care-of Obelix Idefix)
    (barks-at Idefix Ordralphabetix)
)

Semantic Net Example
(defrule humans
  (is-a ?name Gaul)
  =>
  (assert (ARO ?name Human)))

(defrule mutual-friends
  (is-friend-of ?name1 ?name2)
  =>
  (assert (is-friend-of ?name2 ?name1)))

(defrule mutual-fighting
  (fights-with ?name1 ?name2)
  =>
  (assert (fights-with ?name2 ?name1)))

(defrule shopping1
  (buys-from ?name1 ?name2)
  =>
  (assert (sells-to ?name2 ?name1)))

(defrule shopping2
  (sells-to ?name1 ?name2)
  =>
  (assert (buys-from ?name2 ?name1)))

 Relationships

• without relationships, knowledge is an unrelated collection of facts
  – reasoning about these facts is not very interesting
    • inductive reasoning is possible
• relationships express structure in the collection of facts
  – this allows the generation of meaningful new knowledge
    • generation of new facts
    • generation of new relationships
Types of Relationships

- relationships can be arbitrarily defined by the knowledge engineer
  - allows great flexibility
  - for reasoning, the inference mechanism must know how relationships can be used to generate new knowledge
    - inference methods may have to be specified for every relationship
- frequently used relationships
  - IS-A
    - relates an instance (individual node) to a class (generic node)
  - AKO (a-kind-of)
    - relates one class (subclass) to another class ( superclass)

Objects and Attributes

- attributes provide more detailed information on nodes in a semantic network
  - often expressed as properties
    - combination of attribute and value
  - attributes can be expressed as relationships
    - e.g. has-attribute
Implementation Questions

• simple and efficient representation schemes for semantic nets
  – tables that list all objects and their properties
  – tables or linked lists for relationships
• conversion into different representation methods
  – predicate logic
    • nodes correspond variables or constants
    • links correspond to predicates
  – propositional logic
    • nodes and links have to be translated into propositional variables and properly combined with logical connectives

OAV-Triplets
(Object-Attribute-Value triplets)
– can be used to characterize the knowledge in a semantic net
– quickly leads to huge tables

<table>
<thead>
<tr>
<th>Object</th>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Astérix</td>
<td>profession</td>
<td>warrior</td>
</tr>
<tr>
<td>Obélix</td>
<td>size</td>
<td>extra large</td>
</tr>
<tr>
<td>Idéfix</td>
<td>size</td>
<td>petite</td>
</tr>
<tr>
<td>Panoramix</td>
<td>wisdom</td>
<td>infinite</td>
</tr>
</tbody>
</table>
Method-1:

```lisp
(deftemplate OAV
  (slot Object)
  (slot Attribute)
  (multislot Value))

(deffacts objects
  (OAV (Object Asterix) (Attribute profession) (Value warrior))
  (OAV (Object Obelix) (Attribute size)       (Value extra large))
  (OAV (Object Idefix) (Attribute size)       (Value petite))
  (OAV (Object Panoramix) (Attribute wisdom) (Value infinite)))
```

Method-2:

```lisp
(deffacts objects
  (Asterix  profession warrior)
  (Obelix   size       "extra large")
  (Idefix   size       petite)
  (Panoramix wisdom    infinite))
```

Disadvantages of Semantic Nets

- expressiveness
  - no internal structure of nodes
  - relationships between multiple nodes
  - no easy way to represent heuristic information
  - extensions are possible, but cumbersome
  - best suited for binary relationships

- efficiency
  - may result in large sets of nodes and links
  - search may lead to combinatorial explosion
    - especially for queries with negative results

- usability
  - lack of standards for link types
  - naming of nodes
    - classes, instances
Decision Trees

- A decision tree is a semantic net in which:
  - Each node is connected to a set of possible answers.
  - Each nonleaf node is connected to a test that splits its set of possible answers into subsets corresponding to different test results.
  - Each branch carries a particular test result’s subset to another node.
  - Each leaf represents a possible answer.

Example-1 (Binary decision tree)
SOLUTION-1 (Simple but inefficient)

(defun ask-user (?question)
  (printout t '?question " (y/n) " )
  (bind ?answer (read))
  (while (and (neq ?answer y) (neq ?answer n)) do
    (printout t "(y/n) ? " )
    (bind ?answer (read)))
  (return ?answer))

(defun start
  =>
  (assert (node1))

(defun node1
  ?n <- (node1)
  =>
  (retract ?n)
  (if (eq (ask-user "Is the animal warm blooded?" ) n) then
    (printout t "The animal is a snake" crlf)
    else (assert (node2))) )

(defun node2
  ?n <- (node2)
  =>
  (retract ?n)
  (if (eq (ask-user "Does the animal purr?" ) y) then
    (printout t "The animal is a cat" crlf)
    else
      (assert (node3)) )

(defun node3
  ?n <- (node3)
  =>
  (retract ?n)
  (if (eq (ask-user "Does the animal fly?" ) y) then
    (printout t "The animal is a bird" crlf)
    else
      (printout t "The animal is a dog" crlf) )

SOLUTION-2 (Generalized and better)

(deffunction ask-user (?question)
  (printout t ?question " (y/n) ")
  (bind ?answer (read))
  (while (and (neq ?answer y) (neq ?answer n)) do
    (printout t "(y/n) ? ")
    (bind ?answer (read)))
  (return ?answer))

; "answer" nodes :
;   (node <name> answer <value>)
; "decision" nodes :
;   (node <name> decision <question> <yes-node> <no-node>)
(deffacts decision-tree
  (node root decision "Is the animal warm blooded?" node1 node3)
  (node node1 decision "Does the animal purr?" node4 node2)
  (node node2 decision "Does the animal fly?" node5 node6)
  (node node3 answer "The animal is a snake")
  (node node4 answer "The animal is a cat")
  (node node5 answer "The animal is a bird")
  (node node6 answer "The animal is a dog"))

(defrule start
  =>
  (assert (current-node root)))

(defrule do-decision-node
  ?n <- (current-node ?name)
  (node ?name decision ?question ?yes-branch ?no-branch)
  =>
  (retract ?n)
  (if (eq (ask-user ?question) y)
    then (assert (current-node ?yes-branch))
    else (assert (current-node ?no-branch)) ) )

(defrule do-answer-node
  ?n <- (current-node ?name)
  (node ?name answer ?value)
  =>
  (retract ?n)
  (printout t ?value crlf))
Example-2

3. Schemata

- suitable for the representation of more complex knowledge (deep knowledge structure)
  - causal relationships between a percept or action and its outcome
  - “deeper” knowledge than semantic networks
    - nodes can have an internal structure
  - related to the notion of *records* in *databases*
Concept Schema

- abstraction that captures general/typical properties of objects
  - has the most important properties that one usually associates with an object of that type
    - may be dependent on task, context, background and capabilities of the user, …
  - similar to stereotypes
- makes reasoning simpler by concentrating on the essential aspects
- may still require relationship-specific inference methods

Schema Examples

- the most frequently used instances of schemata are
  - frames [Minsky 1975]
  - scripts [Schank 1977]
- frames consist of a group of slots and fillers to define stereotypical objects
- scripts are time-ordered sequences of frames
Frames

- represents related knowledge about a subject
  - provides default values for most slots
- frames are organized hierarchically
  - allows the use of inheritance
- knowledge is usually organized according to cause and effect relationships
  - slots can contain all kinds of items
    - rules, facts, images, video, comments, debugging info, questions, hypotheses, other frames
  - slots can also have *procedural attachments*
    - procedures that are invoked in specific situations involving a particular slot
      - on creation, modification, removal of the slot value

Simple Frame Example

<table>
<thead>
<tr>
<th>Slot Name</th>
<th>Filler</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Astérix</td>
</tr>
<tr>
<td>height</td>
<td>small</td>
</tr>
<tr>
<td>weight</td>
<td>low</td>
</tr>
<tr>
<td>profession</td>
<td>warrior</td>
</tr>
<tr>
<td>armor</td>
<td>helmet</td>
</tr>
<tr>
<td>intelligence</td>
<td>very high</td>
</tr>
<tr>
<td>marital status</td>
<td>presumed single</td>
</tr>
</tbody>
</table>
Overview of Frame Structure

- two basic elements: *slots* and *facets* (fillers, values, etc.);
- typically have parent and offspring slots
  - used to establish a property inheritance hierarchy
    (e.g., specialization-of)
- descriptive slots
  - contain declarative information or data (static knowledge)
- procedural attachments
  - contain functions which can direct the reasoning process (dynamic knowledge)
    (e.g., "activate a certain rule if a value exceeds a given level")
- data-driven, event-driven (bottom-up reasoning)
- expectation-drive or top-down reasoning
- pointers to related frames/scripts - can be used to transfer control to a more appropriate frame

Slots

- each slot contains one or more facets
- facets may take the following forms:
  - values
  - default
    - used if there is not other value present
  - range
    - what kind of information can appear in the slot
  - if-added
    - procedural attachment which specifies an action to be taken when a value in the slot is added or modified (data-driven, event-driven or bottom-up reasoning)
  - if-needed
    - procedural attachment which triggers a procedure which goes out to get information which the slot doesn't have (expectation-driven; top-down reasoning)
  - other
    - may contain frames, rules, semantic networks, or other types of knowledge
Usage of Frames

- filling slots in frames
  - can inherit the value directly
  - can get a default value
  (these two are relatively inexpensive)
  - can derive information through the attached procedures
    (or methods) that also take advantage of current context
    (slot-specific heuristics)
  - filling in slots also confirms that frame or script is
    appropriate for this particular situation

Restaurant Frame Example

- generic template for restaurants
  - different types
  - default values
- script for a typical sequence of activities at a restaurant
Restaurant Script

EAT-AT-RESTAURANT Script

Props: (Restaurant, Money, Food, Menu, Tables, Chairs)
Roles: (Hungry-People, Wait-People, Chef-People)
Point-of-View: Hungry-People
Time-of-Occurrence: (Times-of-Operation of Restaurant)
Place-of-Occurrence: (Location of Restaurant)
Event-Sequenced:
  first: Enter-Restaurant-Script
  then: if (Wait-To-Be-Seated-Sign or Reservations) then Get-Attention-Script
  then: Please-Be-Seated-Script
  then: Order-Food-Script
  then: Eat-Food-Script unless (Long-Wait) when Exit-Restaurant-Angry-Script
  then: if (Food-Quality was better than Palatable) then Compliments-To-The-Chef-Script
  then: Pay-For-It-Script.
finally: Leave-Restaurant-Script
Advantages of Frames

- fairly intuitive for many applications
  - similar to human knowledge organization
  - suitable for causal knowledge
  - easier to understand than logic or rules
- very flexible

Disadvantages of Frames

- it is tempting to use frames as definitions of concepts
  - not appropriate because there may be valid instances of a concept that do not fit the stereotype
  - exceptions can be used to overcome this
    - can get very messy
- inheritance
  - not all properties of a class stereotype should be propagated to subclasses
  - alteration of slots can have unintended consequences in subclasses
KR Languages and Programming Languages

- how is a knowledge representation language different from a programming language (e.g. Java, C++)?
  - programming languages can be used to express facts and states
- what about "there is a pit in [2,2] or [3,1] (but we don't know for sure)" or "there is a wumpus in some square"
- programming languages are not expressive enough for situations with incomplete information
  - we only know some possibilities which exist

KR Languages and Natural Languages

- how is a knowledge representation language different from natural language
  - e.g. English, Spanish, German, …
- natural languages are expressive, but have evolved to meet the needs of communication, rather than representation
- the meaning of a sentence depends on the sentence itself and on the context in which the sentence was spoken
  - e.g. “Look!”
- sharing of knowledge is done without explicit representation of the knowledge itself
- ambiguous (e.g. small dogs and cats)
Good Knowledge Representation Languages

- combines the best of natural and formal languages:
  - expressive
  - concise
  - unambiguous
  - independent of context
    - what you say today will still be interpretable tomorrow
  - efficient
    - the knowledge can be represented in a format that is suitable for computers
    - practical inference procedures exist for the chosen format
  - effective
    - there is an inference procedure which can act on it to make new sentences