Overview

- Search: clarifications and tips
- Knowledge representation
- Knowledge bases
- Logic and Frames
- Propositional Logic
- Inference rules
- Normal forms

Tips for Project 1 (I)

```lisp
(function General-Search (problem, Que-Fn)
  node-list := initial-state
  loop begin
    if Empty(node-list) then return FAIL
    node := Get-First-Node(node-list)
    if (node == goal) then return node as SOLUTION
    node-list := Que-Fn(node-list, Expand(node))
  loop end

  Utilize the pseudo code in the text book
```

Tips for Project 1 (II)

```
’((1 3 4 8 6 2 7 0 5);blank is stored as 0
  h ;heuristic function value
  depth ;depth from the root
  path)) ;list of moves from
; the start

Sorting a node list, e.g. according to the heuristic:

(sort <node-list>
#’(lambda (x y) (< (second x) (second y)) )
)

lambda : read define-anonymous function
#’something = (function something)
cf. ’something = (quote something)

DFS Visit Order Revisited

The DFS (and its variants) visit order in lecture 6 is either correct or incorrect depending on how we define visit:

- node visit as initial creation (used in old slide06):
  → 1 2 3 4 5 8 9 10 11 6 7 12 13 14 15
- node visit as goal test and potential expansion:
  → 1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
**Sorting: Alternatives**

(defun sort-node-list (node-list)
  (sort node-list
     #'(lambda (x y) (< (second x) (second y)) ))
)

; the above is equivalent to :
(defun sort-node-list (node-list)
  (sort node-list
     (function (lambda (x y) (< (second x) (second y)) )))
)

; the above is equivalent to :
(defun compare-h ( x y )
  (< (second x) (second y)))

(defun sort-node-list (node-list)
  (sort node-list #'compare-h))

**Lambda Expression**

Lambda expression can basically replace any occurrences of function names, i.e. it works like an anonymous function:

(defun mysqr (x) (* x x))
(mysqr ’11)

; the above is the same as
((lambda (x) (* x x)) ’11)

; some more examples
(defun myop (x op)
  (eval (list op (first x) (second x))))

(myop ’(2 3) ’*)

(myop ’(2 3) ’(lambda (x y) (* x y)))

**Sorting: Example**

(setq test-node-list
  '((list1 10 0 0) (list2 87 0 0)
   (list 100 0 0) (list 5 1 0 0)))

(defun sort-node-list (node-list)
  (sort node-list #'compare-h))

(defun sort-node-list (node-list)
  (sort node-list
     #'(lambda (x y) (< (second x) (second y)) )
))

(sor

* You can use any combination of values to sort, and do ascending or descending sorts by changing the lambda function.

**Knowledge Representation**

There are basically two classes of representations in traditional AI:

- **Logic**: methods based on first-order predicate calculus (mathematical logic)
- **Frames**: methods based on networks of nodes representing objects or concepts, and labeled arcs representing relations among nodes

These two are competitive, but complementary.
Representation Hypothesis

A central tenet in AI is the representation hypothesis, which states that intelligent behavior is based on

- **representation** of input and output data as symbols
- **reasoning** by processing symbol structures, resulting in new symbol structures

The problem then is what the representations and reasoning process should be.

Strengths and Weaknesses of Logic and Frames

Logic (predicate calculus)

- Strengths: (1) logical power, (2) rigorous mathematical foundation
- Weaknesses: (1) slow (search), (2) rigidity (T/F)

Frames

- Strengths: (1) supports defaults (data is seldom complete), (2) procedural attachment(*)
- Weaknesses: weak logical power

(*) Procedural attachment: pullers (if needed), pusher (if added), if referenced, if deleted, if changed, etc.

Alternatives to the Representation Hypothesis

There are several alternatives:

- analog information: continuous values
- special-purpose hardware: domain specific functions such as vision
- neural networks: subsymbolic approaches
- holographic memories
- etc.

Knowledge Base Systems

Domain-independent algorithms: **Inference engine**
Domain-specific content: **Knowledge base**

- KB: set of sentences in a formal language
- KB is declarative: tell what we want, not how we want it done (i.e. procedural)
KB Constructs

- Knowledge base: how to represent knowledge with sentences or formulas → various forms of logic
- Inference engine: how to generate new sentences or formulas given old ones in the KB → various forms of inference procedures

Logic: Language for KBs

Logic is the representational language for KBs:
- logic consists of syntax (sentence structure) and semantics (how sentences relate to the real world; T/F values)
- interpretation: fact to which a sentence refers (T/F assignment)
- inference: deriving new sentences from old ones

Inference procedure
- sound: no false sentences can be derived from the KB using the inference procedure
- complete: inference procedure can derive all true conclusions from a set of premises

Types of Logic

Ontological: what exists in the world?
Epistemological: what can we know?

<table>
<thead>
<tr>
<th>Language</th>
<th>Ontological</th>
<th>Epistemological</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propositional Logic</td>
<td>facts</td>
<td>T/F/?</td>
</tr>
<tr>
<td>First-order Logic</td>
<td>facts, objects, relations</td>
<td>T/F/?</td>
</tr>
<tr>
<td>Temporal Logic</td>
<td>facts, objects, relations, times</td>
<td>T/F/?</td>
</tr>
<tr>
<td>Probability Theory</td>
<td>facts</td>
<td>degree of belief 0..1</td>
</tr>
<tr>
<td>Probability Theory</td>
<td>facts</td>
<td>degree of belief 0..1</td>
</tr>
<tr>
<td>Fuzzy Logic</td>
<td>degree of truth</td>
<td>degree of belief 0..1</td>
</tr>
</tbody>
</table>

* first-order logic == predicate calculus

Let's begin with propositional logic.
Well-Formed Formulas (Cont’d)

Well-Formed Formulas (wff): Syntax

\[ wff \Rightarrow \text{atom} | \text{constant} \]

\[ wff \Rightarrow (\neg wff) \]

\[ wff \Rightarrow (wff \lor wff) \]
\[ | (wff \land wff) \]
\[ | (wff \rightarrow wff) \]
\[ | (wff \leftrightarrow wff) \]

Operator precedence: \( \neg, \land, \lor, \rightarrow, \leftrightarrow \) (decreasing order)

Propositional Logic: Semantics

- atoms can take on \( T \) or \( F \) values.
- an interpretation assigns specific truth values to the atoms.
- for a formula with \( n \) atoms, there are \( 2^n \) possible truth assignments.
- a formula is true under an interpretation iff the formula evaluates to \( T \) with the assignment of truth values within the interpretation.
- a formula is valid if it is \( T \) under all interpretations.
- a formula is inconsistent (unsatisfiable) if it is \( F \) under all interpretations.
- a formula \( G \) is valid if \( \neg G \) is inconsistent

Propositional Logic: Semantics (cont’d)

- if a formula \( F \) is \( T \) under an interpretation \( I \), then we say \( I \) satisfies \( F \). We also say \( I \) is a model for \( F \)
- if formula \( F \) is \( F \) under interpretation \( I \), then we say \( I \) falsifies \( F \)
- two formulas \( F \) and \( G \) are equivalent iff \( F \) and \( G \) have the same truth values under every interpretation \( I \):
\[ F \leftrightarrow G \]
- there can be many models (at least one) of a formula \( F \) if \( F \) is satisfiable.

Basic Laws of Propositional Logic

- \( F \lor G = G \lor F \), \( F \land G = G \land F \) (commutative)
- \( (F \lor G) \lor H = F \lor (G \lor H) \), \( (F \land G) \land H = F \land (G \land H) \), (associative)
- \( F \lor (G \land H) = (F \lor G) \land (F \lor H) \), \( F \land (G \lor H) = (F \land G) \lor (F \land H) \) (distributive)
- \( F \lor F = F \), \( F \land F = F \) (F: False)
- \( F \lor T = T \)
\[ F \lor \neg F = T \]
\[ F \land \neg F = F \]

17 18 19 20
Basic Formulas (cont’d)

- \( \neg(\neg F) = F \)
- \( \neg(F \lor G) = \neg F \land \neg G \)
- \( \neg(F \land G) = \neg F \lor \neg G \) (De Morgan’s Law)
- \( F \leftrightarrow G = (F \rightarrow G) \land (G \rightarrow F) \)
- \( F \rightarrow G = \neg F \lor G \)
- \( F \land F = F \)
- \( F \lor F = F \)

Normal Forms (I)

- **literals**: atom | \( \neg \)atom
- **clauses**: disjunction of 1 or more **literals**
  
  \( \text{literal} \lor \text{literal} \lor \ldots \)
- **terms**: conjunction of 1 or more **literals**
  
  \( \text{literal} \land \text{literal} \land \ldots \)

Inference Rules

- **Modus Ponens**: 
  
  \[ \frac{F \rightarrow G, F}{G} \]
- **Unit Resolution**: 
  
  \[ \frac{F \lor G, \neg G}{F} \]
- **Resolution**: 
  
  \[ \frac{F \lor G, \neg G \lor H}{F \lor H} \text{ or equivalently } \frac{\neg F \rightarrow G, G \rightarrow H}{\neg F \rightarrow H} \]

Normal Forms (II)

- **Conjunctive Normal Form**: conjunction of **clauses**
  
  \( C_1 \land C_2 \land C_3 \ldots \)
  
  *e.g.* \( (\neg F \lor G \lor H) \land (\neg G) \land (K \lor L) \)
- **Disjunctive Normal Form**: disjunction of **terms**
  
  \( T_1 \lor T_2 \lor T_3 \ldots \)
  
  *e.g.* \( (\neg F \land G \land H) \lor (\neg G) \lor (K \land L) \)
Key Points

- Knowledge representation: logic and frames, pros and cons
- Knowledge bases: the basic components
- Propositional Logic: basic laws
- Inference rules: what is inference, basic inference rules
- Normal forms: definitions

Next Time

Theorem proving:

- Horn clauses, and horn normal form
- forward chaining
- backward chaining
- resolution