Knowledge Representation and Search

- If a computer only has the right knowledge, representation of that knowledge, and way of indexing that knowledge then it can be intelligent

Physical Symbol Hypothesis

- Intelligent activity in either human or machine requires [Newell & Simon] :
  - Symbol patterns to represent significant aspects of a problem domain
  - Operations on these patterns to generate potential solutions to problems
  - Search to select a solution from among these possibilities

Knowledge Bases (KBs)

- Inference Engine
  - Contains domain-independent algorithms
- Knowledge Base
  - A set of sentences in a formal language
  - The declarative approach to building a system: tell it what it needs to know
  - Contains domain-specific content
- Agents can be viewed from a couple of different perspectives
  - The knowledge level: what they know
  - Implementation level: data structures in KB and algorithms that manipulate them

Representation Schemes

- Schemes should be:
  - Expressive: The scheme must be adequate to express all necessary information
  - Efficient: Support efficient execution of the resulting code
  - Natural: Provide a natural scheme for expressing the required knowledge

An example

- Task Description
  - To write a program that finds, for a given phone number, all possible encodings by words, and prints them. A phone number is an arbitrary(?) string of dashes -, slashes / and digits. The dashes and slashes will not be encoded. The words are taken from a dictionary which is given as an alphabetically sorted ASCII file (one word per line).
- Participants: 14 programmers (ave. experience: ~ 7 yr)
- Biggest experimental flaw: subjects self selected
Mapping

- The following mapping from letters to digits is given:

```
E | J N Q | R W X | D S Y | F T | A M |
e | j n q | r w x | d s y | f t | a m |
0 |   1   |   2   |   3   |  4  |  5  |
```

```
C I V | B K U | L O P | G H Z
c i v | b k u | l o p | g h z
6 |   7   |   8   |   9
```

The Results

- **Using Lisp**
  - Time (hr): 2 to 8.5, ave: 5
  - Lines of Code: 51 to 182
  - Run Time (median): 30 seconds

- **Using C/C++**
  - Time (hr): 3 to 25, ave: 11
  - Lines of Code: 107 to 614, ave: 277
  - Run Time (median): 54 seconds

- **Using Java**
  - Time (hr): 4 to 63, ave: 9
  - Lines of Code: 107 to 614, ave: 277

Quickest C/C++ program ran faster than the quickest Lisp program

A Good Representational Scheme

- Inferential efficiency: It should facilitate efficiency in accessing specific knowledge for a particular problem
- Acquisitional adequacy: It should support the addition of new knowledge to the database as the program is running

Key Issues in Knowledge Representation

- Representing basic factual knowledge
  - At the lowest level, we must represent basic facts – e.g. “snoopy is a bird”.
  - We need some sort of system for recording such factual knowledge in a way that’s easy to access.

Why?

- We want agents to accomplish a task in the world
- Agents should be able to:
  - Represent states, actions, etc.
  - Incorporate new perceptions about the world
  - Update internal representations of the world
  - Deduce hidden properties of the world
  - Deduce appropriate actions

Pico’s World

- Pico is a depressed pirate
  - Perceptions: smoke, glitter, smell
  - Actions: left turn, right turn, forward, grab, release, shoot
  - Goals: get gold without entering fire or snoopy square

Pico’s World is derived from Russell & Norvig’s Wumpus World
Snoopy

- Everybody should fear the snoopy bird!
- www.snoopybird.com

Example World

Historically

- The search to bottle reason
  - Aristotle (384-322 B.C.): codified styles of deductive reasoning
  - Gottfried Leibniz (1646-1716): dreamed of a calculus philosophicus
  - George Boole (in 1854): developed the foundations of propositional logic.
  - Gottlieb Frege (1879): developed predicate calculus

Types of Logic

<table>
<thead>
<tr>
<th>Language</th>
<th>What exists?</th>
<th>What states of knowledge?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propositional</td>
<td>Facts</td>
<td>True/False/Unknown</td>
</tr>
<tr>
<td>First-order logic</td>
<td>Facts, objects, relations</td>
<td>True/False/Unknown</td>
</tr>
<tr>
<td>Temporal logic</td>
<td>Facts, objects, relations, times</td>
<td>True/False/Unknown</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>

Propositional Calculus

- “Using words, phrases, and sentences, we can represent and reason about properties and relationships in the world.” [Luger p. 47]

Entailment

- KB |= \( \alpha \)
  - Knowledge Base KB entails sentence \( \alpha \) if and only if \( \alpha \) is true in all worlds where KB is true
  - Examples:
    - the KB containing “The mushroom is purple” and “The mushroom tastes good” entails “The mushroom is purple or the mushroom tastes good”
    - \( x + y = 4 \) entails \( 4 = x + y \)
  - One can think of the truth of \( \alpha \) as being contained in truth of KB
A Good Representational Scheme

- Represenational adequacy: Should be able to represent all knowledge needed
- Inferential adequacy
  - Soundness: something can be inferred only if it’s entailed by the knowledge base
  - Completeness: anything that’s entailed by the knowledge base can be inferred
  - Decidability: if a statement’s true/false, it should be possible to infer its truth/falseness in finite time

[From Elaine Rich & Kevin Knight’s AI book]

Models

- Formally structured worlds with respect to which truth can be evaluated
- \( m \) is a model of a sentence \( \alpha \) if \( \alpha \) is true in \( m \)
- \( M(\alpha) \) is the set of all models of \( \alpha \)
- Then \( KB \vdash \alpha \iff M(KB) \subseteq M(\alpha) \)

Example World

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Smoke</td>
<td>Smoke</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Smell</td>
<td>Smoke</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Smell Smoke</td>
<td>Smoke</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Smoke Smoke</td>
<td>Smoke</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Inference

- \( KB \vdash \alpha = \) sentence \( \alpha \) can be derived from the KB by procedure \( i \)
- Soundness: \( i \) is sound if whenever \( KB \vdash \alpha \), it is also true that \( KB \models \alpha \)
- Completeness: \( i \) is complete if whenever \( KB \models \alpha \), it is also true that \( KB \vdash \alpha \)
- Note: there is a sound and complete inference procedure for the logic we’re defining

Propositional Logic Semantics

- Each model specifies true/false for each proposition symbol
- Rules for evaluation truth with respect to a model \( m \):
  - \( S \) is true iff \( S \) is false
  - \( S_1 \land S_2 \) is true iff \( S_1 \) is true AND \( S_2 \) is true
  - \( S_1 \lor S_2 \) is true iff \( S_1 \) is true OR \( S_2 \) is true
  - \( S_1 \implies S_2 \) is true iff \( S_1 \) is false OR \( S_2 \) is true
  - \( S_1 \iff S_2 \) is true iff \( S_1 \implies S_2 \) is true AND \( S_2 \implies S_1 \) is true

Reminder

- What are these English sentences in propositional logic?
  - There is a smell in 2,1 if and only if there is a snoopy in either 2,2 or 3,1.
  - There is no snoopy in 1,1
  - There is smoke in 2,1 if and only if there is a fire in either 2,2 or 3,1
Validity and Satisfiability

- A sentence is valid if it’s true in all models
- A sentence is satisfiable if it is true in some model
- A sentence is unsatisfiable if it is true in no models
- Satisfiability is connected to inference via the following:
  - $\mathbf{K}\mathbf{B} \vdash \alpha$ if and only if $(\mathbf{K}\mathbf{B} \land \lnot \alpha)$ is unsatisfiable

Normal Forms

- Other approaches to inference use syntactic operations on sentences, often expressed in standardized forms
  - Conjunctive Normal Form (CNF)
    - Conjunction of disjunctions of literals
    - $(A \lor B) \land (B \lor C \lor D)$
  - Disjunctive Normal Form (DNF)
    - Disjunction of conjunctions of literals
    - $((A \land B) \lor (A \land C) \lor (C \land D)) \lor (A \land C)$
  - Horn Form
    - Conjunction of Horn clauses (clauses with ≤1 positive literal)

Proof Methods

- Model checking
  - Truth table enumerating (sound and complete for propositional logic)
  - Heuristic search in model space
- Application of inference rules
  - Legitimate (sound) generation of new sentences from old
  - Proof: a sequence of inference rule applications. We can use inference rules as operators in a standard search algorithm.

Inference Rules for Propositional Logic

- Resolution (for CNF): complete for propositional logic
  \[ \frac{\alpha \lor \beta, \neg \beta \lor \gamma}{\alpha \lor \gamma} \]
- Modus Ponens (for Horn Form): complete for Horn KBs
  \[ \frac{\alpha_1, \ldots, \alpha_n, \alpha_1 \land \ldots \land \alpha_n \Rightarrow \beta}{\beta} \]

Inference Rules

- Modus tollens:
  \[ P \Rightarrow Q \text{ is known to be true and } Q \text{ is known to be false, then we can infer } \lnot P \]
- And elimination: \[ P \land Q \text{ is known to be true then we can conclude } P \text{ and } Q \]
- And introduction: vice-versa of above
- Universal instantiation:
  - If \( a \) if from the domain of \( x \), \( \forall x \ P(x) \) lets us infer \( P(a) \)

Resolution

- Ch. 7
- A complete inference algorithm when coupled with any complete search algorithm
Resolution: how

- Procedure:
  - Put premises/axioms into clause form
  - Add negation of what is to be proved, in clause form, to the set of axioms.
  - Resolve these clauses together, producing new clauses that logically follow from them.
  - Produce a contradiction by generating the empty clause.

- This is \textit{reductio ad absurdum} (proof by reduction to an absurd thing) or proof by contradiction

Proving in Pico’s World

- If pico has visited [1,1], [2,1], and [1,2], prove that snoopy is in square [3,1]
  - What is in the KB?

Example World

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Another problem

- Given the following, can you prove that the unicorn is mythical? Magical? horned?
  - If the unicorn is mythical, then is is immortal, but if it is not mythical, then it is a mortal mammal. If the unicorn is either immortal or a mammal, then it is horned. The unicorn is magical if it is horned.

For you to do

- If there is goodness or happiness then there are no jelly beans or killing. There is goodness. Prove: There are no jelly beans.
- If there are people then it is not quiet. There is no rest if it is not quiet. Prove: If there are people there is no rest.
- Prove: ((PvQ) ^ ~P) -> Q

Resolution Heuristics

- Clause elimination:
  - Certain types of clauses can be ignored
    - Pure clauses (where P appears and \sim P doesn’t appear elsewhere)
    - Tautologies
- Resolving:
  - Unit clauses first
  - Start with what you’re trying to prove
  - Aim to shorten clauses
Problems?

- What kinds of problems exist for propositional logic?

Predicate Calculus

- A way to access the components of an individual assertion
- Predicate Calculus: used extensively in many AI programs, especially expert systems

Syntax

- Constants: Jessica, 2, RIT
- Predicates: Sister, >, …
- Functions: Sqr, Eat, …
- Variables: x, y, a, b
- Connectives: = ∧ ∨ ¬ ⇒ ⇔
- Equality: =
- Quantifiers: ∀ ∃

Atomic Sentences

- Atomic sentences:
  - predicate(term₁, …, termₙ)
  - term₁ = term₂
- Term:
  - function(term₁, …, termₙ)
  - constant
  - variable
- Examples
  - Daughter(Lisa, Homer)
  - (Length(LeftLegOf(Homer)), Length(LeftLegOf(Marge)))

Truth

- Sentences are true with respect to a model and an interpretation
- Models contain objects and relations among them
- Interpretation specifies values referred to for
  - Constant symbols
  - Predicate symbols
  - Function symbols
- An atomic sentence predicate(term₁, …, termₙ) is true if the objects referred to by term₁, …, termₙ are in the relation referred to by predicate

Complex Sentences

- Complex sentences are made from atomic sentences using connectives
  \( \neg S \quad S \quad \wedge S \quad S \quad \vee S \quad S \quad \Rightarrow S \quad S \quad \Leftrightarrow S \)
- Examples:
  - Sibling(Pico, HAL) \( \Rightarrow \) Sibling(HAL, Pico)
  - \( (42,3) \leq (42,3) \)
  - \( (42,3) \wedge \neg > (42,3) \)
Universal Quantification

∀ <variables> <sentence>

- “Everyone in Computer Science is smart”
  \[∀x \text{ In(x, ComputerScience)} \Rightarrow \text{Smart(x)}\]
- \[∀x \text{ P is equivalent to the conjunction of instantiations of P}\]
  \[\text{In(Matt, ComputerScience)} \Rightarrow \text{Smart(Matt)}\]
  \[\wedge \text{In(Dan, ComputerScience)} \Rightarrow \text{Smart(Dan)}\]
  \[\wedge \text{In(Tim, ComputerScience)} \Rightarrow \text{Smart(Tim)}\]
  \[\wedge ...\]

Common Mistake

Typically, \(\Rightarrow\) is the main connective with \(∀\).
- Common mistake: using \(∧\) as the main connective
  \[∀x \text{ In(x, ComputerScience)} \wedge \text{Smart(x)}\]
- Means “Everyone is in Computer Science and everyone is smart”

Existential Quantification

∃ <variables> <sentence>

- “Somebody in Computer Science is smart”
  \[∃x \text{ In(x, ComputerScience)} \wedge \text{Smart(x)}\]
- \[∃x \text{ P is equivalent to the disjunction of instantiations of P}\]
  \[\text{In(Matt, ComputerScience)} \wedge \text{Smart(Matt)}\]
  \[\vee \text{In(Dan, ComputerScience)} \wedge \text{Smart(Dan)}\]
  \[\vee \text{In(Tim, ComputerScience)} \wedge \text{Smart(Tim)}\]
  \[\vee ...\]

Common Mistake

Typically, \(∧\) is the main connective with \(∃\).
- Common mistake: using \(\Rightarrow\) as the main connective
  \[∃x \text{ In(x, ComputerScience)} \Rightarrow \text{Smart(x)}\]
- Is true if there’s anyone who’s not in Computer Science!

More about Quantifiers

- \(∀x∀y\) is the same as \(∀y∀x\)
- \(∃x∃y\) is the same as \(∃y∃x\)
- \(∃x∀y\) is not the same as \(∀x∃y\)
- There is a person who loves everyone in the world.
  \[∃x∀y \text{ Loves(x,y)}\]
- Everyone in the world is loved by at least one person
  \[∀x∃y \text{ Loves(x,y)}\]

Quantifier Duality

- Each can be expressed using the other
  \[∀x \text{ Eats(x, Mushrooms)} \Rightarrow ∃x \neg \text{Eats(x, Mushrooms)}\]
  \[∃x \text{ Eats(x, Vegetables)} \Rightarrow ∃x \neg \text{Eats(x, Vegetables)}\]
English into First Order Logic

- Can you do these?
  - Every dog hates cats.
  - All purple mushrooms are tasty.
  - You can fool some of the people all of the time.
  - You can fool all of the people some of the time.
- [answers discussed in class]

Equality

- term₁ = term₂ is true under a given interpretation if and only if term₁ and term₂ refer to the same object
  - 1=2 and \( \forall x \quad (\text{Sqrt}(x), \text{Sqrt}(x)) = x \) are satisfiable
  - 2=2 is valid
- Definition of (full) Sibling in terms of Parent
  \( \forall x, y \quad \text{Sibling}(x, y) \iff \neg(x = y) \land \exists m, f \quad \neg(m = f) \land \\
  \text{Parent}(m, x) \land \text{Parent}(f, x) \land \text{Parent}(m, y) \land \text{Parent}(f, y) \)

Prolog

- PROgramming in LOGic

Declarative vs. Procedural Programming

- Declarative:
  - The programmer must know the relationships between different entities
- Procedural:
  - The programmer must tell the computer how to do something

Note of Caution

- Prolog is difficult to master, because it doesn’t have the same structures as most other programming languages

A Couple of Prolog Basics

- Starting Prolog: pl
- You probably want to type your facts/rules in a file and then load the file:
  - [filename].
  - Note: filename is really filename.pl
- Stopping the Prolog interpreter:
  - halt.
- Help on a topic: help(topic).
- Edit a file test.pl: edit(test).
- List child to screen: listing(child).
- Trace descendant: trace(descendant).
- Switch tracing off: trace(descendant, -all).
How Do We Display Them in Prolog?

- Based on First order predicate logic
  - Uses Horn clause form
  - $q_1 \land q_2 \land \ldots \land q_n \rightarrow q_0$ where each $q_i$ and $q_0$ is an **atomic sentence** and all variables are universally quantified.

Multiple Clauses

- A clause is basically a prolog sentence (and ends with a period)
  - Example: `eats(snoopy, birdFood)`.
- How do we represent?: Snoopy eats bird food and wood.

Answer

- FOPL:
  - `Eats(Snoopy, BirdFood) \land Eats(Snoopy, Wood)`
- Prolog:
  - `eats(snoopy, birdFood),eats(snoopy, wood)`.

What about?

- The square root of 25 is 5 or it is going to rain.
- Perl, Java, and eLisp are all languages.

Rules

- If a dog is wet and dirty, then he is smelly
- FOPL:
  - `Ax\, Wet(x) \land Dirty(x) \land Dog(x) \rightarrow Smelly(x)`
- Prolog:
  - `smelly(X) :- wet(X),dirty(X),dog(X)`.
- Only one goal is allowed at the head of the rule (before the :-)

Goals and Subgoals

- `smelly(X) :- wet(X),dirty(X),dog(X)`.
  - In prolog `smelly(X)` is a goal and `wet(X)`/`dirty(X)`/`dog(X)` are subgoals
  - Why?
**Represent These Statements**

- In predicate logic and Prolog
  - All animals eat jelly beans.
  - Everyone loves Wumpus.
  - Snoopy likes to eat Jessica’s furniture.

**Translate**

- Two people live in the same house if they have the same address.
- Two people are siblings if they have the same parents.
- Someone is happy if they’re healthy, wealthy, or wise.
- A dog is happy if he’s healthy, wealthy, or wise.
  - Note that predicates are called “functors”

**Some Basic Prolog**

**Facts**
- likes(joe, eLisp).
- likes(john, eLisp).
- likes(joe, mary).
- likes(mary, books).
- likes(mary, frankenstein).
- likes(john, frankenstein).

**Questions**
- ?- likes(joe, eLisp).
- Yes
- ?- likes(mary, joe).
- No
- ?- likes(mary, frankenstein).
- Yes
- ?- dislikes(mary, joe).
- No

**Multiple Matches**

**Facts**
- likes(joe, eLisp).
- likes(john, eLisp).
- likes(joe, mary).
- likes(mary, books).
- likes(mary, frankenstein).

**Questions**
- ?- likes(joe, X).
- If you type ‘;’ prolog will search for more matches.
- Prolog searches the facts from top to bottom

**How Do We Ask?**

- Is there anything that John and Mary both like?
  - likes(john, X), likes(mary, X).
- In this case, Prolog will backtrack to find the right answer.

**Rules**

- How do we write?
  - Jack likes anyone who likes books.
  - ~likes(jack, X), likes(X, books).
  - ~likes(jack, likes(X, books)).
  - ~likes(jack, X) :- likes(X, books).
  - Jack likes anyone who likes books and fish.
  - Happy people are people who like themselves.
  - ~happy(Person) :- likes(Person, Person).
Recursion

- descendant(X,Y) :- child(X,Y).
- descendant(X,Y) :- child(X,Z), descendant(Z,Y).

- Avoid circular definitions:
  - child(X,Y) :- parent(Y,X).
  - parent(X,Y) :- child(Y,X).

Logically Correct, But…

- descendant(X,Y) :- child(X,Y).
- descendant(X,Y) :- descendant(Z,Y), child(X,Z).

- The above statement is logically correct, but may cause Prolog to go into an infinite loop, because of Prolog’s left-to-right, depth-first order of evaluation.

Predicates and Operators

- =, \=, <, <=, >, >=
- Arithmetic ops: +, -, *, etc.
- not

Prolog Variables

- Variables only refer to the same entity if they’re inside the same clause.

- Note: logical variables are a bit different from other variables
  - X=1; X=2; will result in 2 in Java
  - X=1,X=2 won’t work since the value X can only be one value

Prolog Functions

- Defining factorial:
  - You use the word “is” to do assignment
    - Example: A is 1*1.

  - The base case will be a separate rule from the regular case and will be stated separately.
  - Rather than a regular return type you normally have the return be a variable argument to the function