ICS-381
Principles of Artificial Intelligence

Lectures 9-10

Knowledge Representation Methods

Dr. Tarek Helmy El-Basuny
Topics to be Covered

- What Is Knowledge?
- Fundamental Activities in AI Correlated with Knowledge
- Representation = Syntax + Semantics + Reasoning
- Knowledge Representation Methods
  - Logic as a Knowledge Representation Language
    - Propositional Logic
    - Predicate Logic
  - Semantic Networks
  - Frames
  - Production Rules
  - Java Classes
- Getting Started with Prolog
- Types of Search
What Is Knowledge?

- **Knowledge** is what I know and **Information** is what we know
- Knowledge can be **considered as the distillation of information** that has been collected, classified, organized, integrated, abstracted and value added.
- Intelligent behavior is not dependent so much on the methods of reasoning as on the knowledge one has to reason with.
- Characteristics of Knowledge
  - Knowledge is huge (**Large in number or quantity**).
  - Knowledge is hard to characterize accurately.
  - Knowledge differs from data in that it is organized such that it corresponds to the ways it will be used.
  - Knowledge is interpreted differently by different people.
If we are going to act rationally in our environment, then we must have some way of describing that environment.

1. How do we describe what we know about the world?
2. How do we describe it **concisely**/**in brief**?
3. How do we describe it so that we can get the right piece of knowledge when we need it?
4. How do we generate new pieces of knowledge?
5. How do we deal with **uncertain** knowledge?
Fundamental Activities in AI about Knowledge

- **Acquisition**
  - Creating, finding, collecting, eliciting the necessary information and knowledge to create a cognitive model
  - Highly dependent on technology being utilized

- **Representation**
  - Transforming the acquired information and knowledge into a computer-readable model.
  - Represent the computer’s knowledge of the world by some kind of data structures in the machine’s memory.
  - Representation = Syntax + Semantics + Reasoning

- **Search**
  - A problem solving technique that systematically explores a space of problem states.
Representation = Syntax + Semantics + Reasoning

- **Syntax**
  - Can be checked by “compiler”
  - Can take on many forms
  - Mathematical, logical, model, cases, rules, graphs, frames, symbols

- **Semantics**
  - Must be checked by human
  - Specification of “behaviors”
  - Careful attention to semantics allows us to be precise about system capabilities

- **Reasoning with Certainty**
  - Deterministic
    - Propositional Logic
    - Predicate Logic

- **Reasoning with Uncertainty**
  - Non-Deterministic
    - Multi-Valued Logic
    - Probability Theory and Bayesian Deduction
    - Certainty Factors
    - Fuzzy Logic
Knowledge Representation Methods

- Knowledge may be represented as “symbol structures” representing bits of knowledge.
  - E.g., “red” represents color red. “car1” represents my car.
  - red(car1) represents fact that my car is red.

- Intelligent behavior can be achieved through manipulation of symbol structures.

- Knowledge representation languages have been designed to facilitate this.

- Rather than use general C++/Java data structures, use special purpose formalisms.

- A Knowledge representation language should allow you to:
  - Represent adequately the knowledge you need for your problem
  - Do it in a clear, precise and “natural” way.
  - Allow you to reason on that knowledge, drawing new conclusions.

- Knowledge Representation schemes
  - Logical Representation Schemes (e.g. Propositional or Predicate Logic, …)
  - Network Representation Schemes (e.g. Semantic Networks)
  - Structured Representation Schemes (e.g. Frame)
  - Procedural Representation Schemes (e.g. Production Rules)
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

[Rogers 1999]
Logic as a Knowledge Representation Language

- A Logic is a formal language, with precisely defined syntax and semantics, which supports reasoning.
- Independent of the domain of application.
- Different logics exist, which allow you to represent different kinds of things, and which allow more or less efficient inference.
  - Propositional logic, Predicate logic, Temporal logic, Modal logic, Description logic.
- But representing some things in logic may not be very natural, and inferences may not be efficient.
Propositional Logic

- In general a logic is defined by
  - Syntax: Describes the possible configurations that can constitute sentences.
  - Semantics: Determines the facts in the world to which the sentences refer.
  - Reasoning: Process of constructing new sentences from old ones
    - This relationship is called entailment and can be expressed as:
      \[ KB \models \alpha \] (knowledge base KB entails the sentence alpha)

- Propositional Logic: Syntax
  - Symbols (e.g., letters, words) are used to represent facts about the world: e.g.
    - “P” represents the fact “Ali likes chocolate”
    - “Q” represents the fact “Ali has chocolate”
  - These are called atomic propositions
  - Logical connectives are used to represent and: \( \land \), or: \( \lor \), if-then: \( \Rightarrow \), not: \( \neg \),
    equality: IFF \( \iff \).
  - Statements or sentences in the language are constructed from atomic propositions
    and logical connectives.
Propositional Logic (Sentences)

- $P \land \neg Q$ “Ali likes chocolate and he doesn’t have any.”
- $P \Rightarrow Q$ “If Ali likes chocolate then Ali has chocolate”

- **Propositional Logic Sentences**
  - Every propositional symbol and truth symbol is a sentence.
    - Ex) true, P, Q, and R are sentences.
  - The *negation* of a sentence is a sentence.
    - Ex) $\neg P$ and $\neg false$ are sentences.
  - The *conjunction*, or *and*, of two sentences is a sentence.
    - Ex) $P \land \neg P$ is a sentence.
  - The *disjunction*, or *or*, of two sentences is a sentence.
    - Ex) $P \lor \neg P$ is a sentence.
  - The *implication* of one sentence for another is a sentence.
    - Ex) $P \Rightarrow Q$ is a sentence.
  - The *equivalence* of two sentences is a sentence.
    - Ex) $P \lor Q = R$ is a sentence.
  - Legal sentences are called *well-formed formulas* or *WFFs*. 
Semantics: What does it all mean?

Sentences in propositional logic tell you about what is true or false.

Propositional Logic Semantics

- The truth assignment of negation, \( \neg P \), where \( P \) is any propositional symbol, is \( F \) if the assignment to \( P \) is \( T \), and \( T \) if the assignment to \( P \) is \( F \).

- The truth assignment of conjunction, \( \wedge \), is \( T \) only when both conjuncts have truth value \( T \); otherwise it is \( F \).

- The truth assignment of disjunction, \( \vee \), is \( F \) only when both disjunction have truth value \( F \); otherwise it is \( T \).

- The truth assignment of implication, \( \Rightarrow \), is \( F \) only when the premise or symbol before the implication is \( T \) and the truth value of the consequent or symbol after the implication is \( F \); otherwise it is \( T \).

- The truth assignment of equivalence, \( \equiv \), is \( T \) only when both expressions have same truth assignment for all possible interpretations; otherwise it is \( F \).
Propositional Logic: Semantics

- Sentences in propositional logic tell you about what is true or false.
  - $P \land Q$ means that both $P$ and $Q$ are true.
  - $P \lor Q$ means that either $P$ or $Q$ is true (or both)
  - $P \Rightarrow Q$ means that if $P$ is true, so is $Q$.
  - IFF $\Leftrightarrow$, equality: the two terms are equivalent

- This is all formally defined using *truth tables.*

<table>
<thead>
<tr>
<th>$P$</th>
<th>$Q$</th>
<th>$\neg P$</th>
<th>$P \land Q$</th>
<th>$P \lor Q$</th>
<th>$P \Rightarrow Q$</th>
<th>$P \Leftrightarrow Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>
Propositional Logic (Equivalences)

- Equivalences
  - \( \neg (\neg P) = P \)
  - \( (P \lor Q) = (\neg P \Rightarrow Q) \)
  - The contra positive law
    - \( (P \Rightarrow Q) = (\neg Q \Rightarrow \neg P) \)
  - De Morgan’s law
    - \( \neg (P \lor Q) = (\neg P \land \neg Q) \) and \( \neg (P \land Q) = (\neg P \lor \neg Q) \)
  - The commutative laws
    - \( (P \land Q) = (Q \land P) \) and \( (P \lor Q) = (Q \lor P) \)
  - Associative law
    - \( ((P \land Q) \land R) = (P \land (Q \land R)) \) and \( ((P \lor Q) \lor R) = (P \lor (Q \lor R)) \)
  - Distributive law
    - \( P \lor (Q \land R) = (P \lor Q) \land (P \lor R) \) and \( P \land (Q \lor R) = (P \land Q) \lor (P \land R) \)

- The proof of these laws can be done by using truth tables and is left for you as exercises.
• Propositional logic isn’t powerful enough as a general knowledge representation language.

• Impossible to make general statements. E.g., “all students sit exams” or “if any student sits an exam they either pass or fail”.

• So we need predicate logic.

• In predicate logic the basic unit is a predicate/argument structure called an atomic sentence:
  - likes(ali, chocolate)
  - tall(fred)

• Arguments can be any of:
  - constant symbol, such as ‘ali’
  - variable symbol, such as X
  - function expression, e.g., motherof(fred)

• So we can have the following predicates:
  - likes(X, richard)
  - friends(motherof(joe), motherof(jim))
Predicate Logic (Sentences)

Predicate Logic Sentences

Every atomic sentence is a sentence

- If $s$ is a sentence, then so is its negation, $\neg s$.
- If $s_1$ and $s_2$ are sentences, then so is their conjunction, $s_1 \land s_2$.
- If $s_1$ and $s_2$ are sentences, then so is their disjunction, $s_1 \lor s_2$.
- If $s_1$ and $s_2$ are sentences, then so is their implication, $s_1 \Rightarrow s_2$.
- If $s_1$ and $s_2$ are sentences, then so is their equivalence, $s_1 = s_2$.
- If $X$ is a variable and $s$ a sentence, then $\forall X \ s$ is a sentence.
- If $X$ is a variable and $s$ a sentence, then $\exists X \ s$ is a sentence.
Sentences can also be formed using quantifiers ∀ (forall) and ∃ (there exists) to indicate how to treat variables:

- ∀ X lovely(X)  Everything is lovely.
- ∃ X lovely(X)  Something is lovely.
- ∀ X in(X, garden) ⇒ lovely(X)  Everything in the garden is lovely.
- If it doesn’t rain tomorrow, Tom will go to the mountains
  - ¬weather(rain, tomorrow) ⇒ go(tom, mountains)
- Emma is a Doberman pinscher and a good dog
  - gooddog(emma) ∧ isa(emma, doberman)
- All basketball players are tall
  - ∀ X (basketball_player(X) ⇒ tall(X))
- If wishes were horses, beggars would ride.
  - equal(wishes, horses) ⇒ ride(beggars).
- Nobody likes taxes
  - ¬∃ X likes(X, taxes)

So, try to represent the followings:

- All men are mortal.
- No one likes brussel sprouts.
- Everyone taking AI will pass their exams.
- Every race has a winner.
- John likes everyone who is tall.
- John doesn’t like anyone who likes brussel sprouts.
- There is something small and slimy on the table.
There is a precise meaning to expressions in predicate logic. Like in propositional logic, it is all about determining whether something is true or false.

∀ X P(X) means that P(X) must be true for every object X in the domain of interest.

∃ X P(X) means that P(X) must be true for at least one object X in the domain of interest.

So if we have a domain of interest consisting of just two people, john and mary, and we know that tall(mary) and tall(john) are true, we can say that ∀ X tall(X) is true.
Predicate Logic: Example

- **Representing a blocks world**

  ![Blocks World Diagram]

  - For all $X$, $X$ is clear if there does not exist a $Y$ such that $Y$ is on $X$.
    - $\forall X \ (\neg \exists Y \ on(Y, X) \Rightarrow clear(X))$.
  - To stack $X$ on $Y$ first empty the hand, then clear $X$, then clear $Y$, and then pick_up $X$ and put_down $X$ on $Y$.
    - $\forall X \ \forall Y \ ((hand_empty \ \land \ clear(X) \ \land \ clear(Y) \ \land \ pick_up(X) \ \land \ put_down(X, Y)) \Rightarrow stack(X,Y))$. 

  

- on(c,a).
- on(b,d).
- ontable(a).
- ontable(d).
- clear(b).
- clear(c).
- hand_empty.
Again we can define inference rules allowing us to say that if certain things are true, certain other things are sure to be true, e.g.

$$\forall X P(X) \Rightarrow Q(X)$$

$P(something)$

------------ (so we can conclude)

$Q(something)$

This involves matching $P(X)$ against $P(something)$ and binding the variable $X$ to the symbol something.

What can we conclude from the following?

- $\forall X \text{tall}(X) \Rightarrow \text{strong}(X)$
- $\text{tall}(john)$
- $\forall X \text{strong}(X) \Rightarrow \text{loves}(mary, X)$
- $\forall X \exists Y \text{loves}(X, Y)$
- $\forall X \exists Y \text{loves}(Y, X)$
Other Knowledge Representation Methods

- Logic isn’t the only method of representing knowledge.
- There are other methods which are less general, but more natural, and arguably easier to work with:
  - Semantic Nets
  - Frames
  - Objects
- To some extent modern OOP has superseded the first two, with the ability to represent knowledge in the object structures of your programming language.
- Semantic nets, frames and objects all allow you to define relations between objects, including class relations (X isa Y).
- Only restricted inference supported by the methods - that based on inheritance.
  - *Fido is a dog,*
  - *dogs have 4 legs,*
  - *so Fido has 4 legs.*
Semantic networks and Frames are simple and perceptive ways to represent information in a knowledge base.

Both use graphs to represent the information.

Semantic Network: Collection of (named) nodes and (labeled) links.

Nodes connected by vertices, which represent relationships between the concepts.

The subclass & inheritance relationships can be used to infer new information.

A very object oriented approach to knowledge representation.
Knowledge Representation: Semantic Network

- Many AI problem domains require large amounts of highly structured interrelated knowledge.
- We can use the logical predicates to represent the relationship between the events over time.
  
  - hassize(bluebird,small)
  - hascovering(bird,feathers)
  - hascolor(bluebird,blue)
  - hasproperty(bird,flies)
  - isa(bluebird,bird)
  - isa(bird,vertebrate)

Assumed hierarchy that explains response data

- hassize(bluebird,small)
- hascovering(bird,feathers)
- hascolor(bluebird,blue)
- hasproperty(bird,flies)
- isa(bluebird,bird)
- isa(bird,vertebrate)
- Knowledge represented as a network or graph

[Semantic Networks diagram]

- Animal
  - subclass Reptile
  - subclass Mammal
  - haspart head

- Mammal
  - subclass haspart Reptile

- Reptile
  - subclass livesin Africa

- Elephant
  - subclass size large
  - instance Nellie

- Nellie
  - likes apples

- Person
  - type Dr
  - name Ahmed
  - email ah@kfupm.edu

- Con-ID
  - title
  - email

Dr. Tarek Helmy, ICS-KFUPM
By traversing network we can find:

- That Nellie has a head (by inheritance)
- That certain concepts related in certain ways (e.g., apples and elephants).

BUT: Meaning of semantic networks was not always well defined.

- Are all Elephants big, or just typical elephants?
- Do all Elephants live in the “same” Africa?
- Do all animals have the same head?

For machine processing these things must be defined.
Java Classes

- A set of classes like **Character** and **Item** with instances like Weebl, Bob and Bull, Pie and Canon + ordered methods calls.
- Each class has a number of instance variables to describe the features of the class/object, and a number of methods to define how objects can interact.
- **Problem:**
  - Hard to represent ‘likes’ and ‘hates’ directly.
  - Also quite awkward to represent firing the canon at someone.
Similar to semantic networks, but all the information relevant to an object is held inside the node

- **Frames look much like modern classes, without the methods:**

Frames have two main parts:
- Slots which hold variable data
- Fixed parts which hold static data

Data inside the frame can be of any kind, including procedures.

<table>
<thead>
<tr>
<th>Panda</th>
<th>Fixed part</th>
<th>Slot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colour: black and white</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NoEars: 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Food: Bamboo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EatFunc: ...........</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>Height:</th>
<th>Age: 0</th>
</tr>
</thead>
</table>

**mammal:**
- **subclass:** animal

**elephant:**
- **subclass:** mammal
- **size:** large
- **haspart:** trunk

**Nellie:**
- **instance:** elephant
- **likes:** apples
Frames often allowed you to say which things were just typical of a class, and which were definitional, so couldn’t be overridden.

Using an asterix to denote typical values:

Elephant:
- subclass: mammal
- haspart: trunk
  * color: grey
  * size: large

Frames also allowed multiple inheritance (Nellie is an Elephant and is a circus animal).
Many AI problem domains require large amounts of highly structured interrelated knowledge. We can use frame to represent the entities of the system.
Semantic Networks and Frames – Some Problems

- It is difficult to implement some logical relationships between concepts
  - How could you represent the fact that an animal is either male OR female? Or that a person prefers either trains OR airplanes
  - What about the “there exists” relationship – there exists a person who does not like chocolate?

- For frames, instantiating new frames by matching to archetypes is very difficult
  - How closely must a new concept match an archetype before it is instantiated?
Knowledge Representation: Production Rules

- Production Rules
  - Rule 1
    - IF X has a risk of heart attack
      AND X has had a previous heart attack
    THEN give X the drug Digitalis
  - Rule 2
    - IF X has left quadratic pain
      AND X has high blood pressure
    THEN X has a risk of a heart attack
  - Rule 3
    - IF X has raised intraocular pressure
    THEN X has high blood pressure
Can we program in logic?

- Yes! The main logic programming language is called Prolog.

What is Prolog?

- What it says, ‘PROgramming in LOGic’.
- It’s a simple subset of predicate logic.
- Everything in Prolog is based around *implication*
  - i.e. ‘B => A’ which means that B implies A; B can have one or more conjoined parts.
- Everything in Prolog is *relational*, i.e. it’s based around predicates.
- It works by *searching* its given database of facts in a special way.
A Prolog program consists of a number of relations, where each relation is made up of clauses of the form A :- B:

relation(Var1, Var2, Var3) :-
    relation1(Var1, Var2, VarX),
    relation2(Var3, VarX),
    ...
    relationN(VarX, VarY).

- Head/consequent ‘A’
- Implication
- Comma means ‘AND’
- Body/antecedents ‘B’
- Full stop means ‘end’

In other words, “if relation1 and relation2 and … and relationN are true, then so is ‘relation’.”
Prolog and Logic

Prolog is based on predicate logic, but with slightly different syntax.

- \( a(X) :- b(X), c(X) \). *Equivalent to*

- \( \forall X \ a(X) \iff b(X) \land c(X) \) *Or equivalently*

- \( \forall X \ b(X) \land c(X) \implies a(X) \)

Prolog has a built-in proof/inference procedure, that lets you determine what is true given some initial set of facts. Proof method called “resolution”.

Getting Started with Prolog

- Prolog is a language based on first order predicate logic. (Will revise/introduce this later).
- We can assert some facts and some rules, then ask questions to find out what is true.
- Facts:
  
  ```
  likes(john, mary).
  tall(john).
  tall(sue).
  short(fred).
  teaches(alison, ai3).
  ```

- Note: lower case letters, full stop at end.
**Rules:**

- `likes(fred, X) :- tall(X).`
- `examines(Person, Course) :- teaches(Person, Course).`

- John likes *someone if* that someone is tall.
- A person examines a course *if* they teach that course.
- NOTE: “:-” used to mean IF. Meant to look a bit like a backwards arrow.
- NOTE: Use of capitals (or words starting with capitals) for variables.
Your “program” consists of a file containing facts and rules.

You “run” your program by asking “questions” at the prolog prompt.

?- likes(fred, X).

John likes who?

Answers are then displayed. Type “;” to get more answers: (Note: darker font for system output)

X = john ? ;
X = sue ? ;
no
Note that Prolog can return more than one answer to a question.
It has a built-in search method for going through all the possible rules and facts to obtain all possible answers.
It uses “depth first search” with “backtracking”.

**Blind Search**
1. **Breadth-first search**: Search technique that looks for a solution along all of the nodes on one level of a problem space before considering nodes at the next lower level.
2. **Depth-first search**: Search technique that looks for a solution along each branch of a problem space to its full vertical length, then proceeds in some defined order, such as from left to right.

**Heuristic Search**
1. **Best-first search**: Search technique that uses knowledge about the problem to guide the search. It guides the search towards the solution nodes of the problem space.
2. **Hill-Climbing Search**: The Hill-Climbing search always moves towards the goal. Using heuristics it finds which direction will take it closest to the goal. The hiker knows that every step he takes up the mountain is a step towards his goal.
So a hill-climbing search always goes to the node closest to the goal.
The End!!

Thank you

Any Questions?