Chapter 8

(Please turn your mobile devices to silent. Thanks!)
Logical agents apply inference to a knowledge base to derive new information and make decisions

Entailment vs. Inference

Two ways to prove a query
1. Application of inference rules
2. Model checking

Soundness and Completeness as conditions for inference

Resolution is complete for propositional logic in CNF

Forward, backward chaining are linear-time, complete for Horn clauses
Outline

- Why FOL?
- Syntax and semantics of FOL
- Using FOL
- Wumpus world in FOL
- Knowledge engineering in FOL
Pros and cons of propositional logic

😊 Propositional logic is declarative
😊 Propositional logic allows partial/disjunctive/negated information
  o (unlike most data structures and databases)
😊 Propositional logic is compositional:
  o meaning of $B_{1,1} \land P_{1,2}$ is derived from meaning of $B_{1,1}$ and of $P_{1,2}$
😊 Meaning in propositional logic is context-independent
  o (unlike natural language, where meaning depends on context)

😡 Propositional logic has very limited expressive power
  o (unlike natural language)
  o E.g., cannot say "pits cause breezes in adjacent squares"
    - except by writing one sentence for each square
First-order logic

- Whereas propositional logic assumes the world contains facts,
- first-order logic (like natural language) assumes the world contains
  - Objects: people, houses, numbers, colors, baseball games, wars, …
  - Relations: red, round, prime, brother of, bigger than, part of, comes between, …
  - Functions: father of, best friend, one more than, plus, …
Syntax of FOL: Basic elements

- Constants  KingJohn, 2, NUS,...
- Predicates  Brother, >,...
- Functions  Sqrt, LeftLegOf,...
- Variables  x, y, a, b,...
- Connectives  ¬, ⇒, ∧, ∨, ⇔
- Equality  =
- Quantifiers  ∀, ∃
**Atomic sentences**

Atomic sentence = \( \text{predicate} (\text{term}_1, \ldots, \text{term}_n) \)

or \( \text{term}_1 = \text{term}_2 \)

Term = \( \text{function} (\text{term}_1, \ldots, \text{term}_n) \)

or constant or variable

- E.g., \( \text{Brother(KingJohn,RichardTheLionheart)} > (\text{Length(LeftLegOf(Richard))}, \text{Length(LeftLegOf(KingJohn))}) \)

Functions can be viewed as complex names for constants
Complex sentences

Made from atomic sentences using connectives

\[ \neg S, \ S_1 \land S_2, \ S_1 \lor S_2, \ S_1 \Rightarrow S_2, \ S_1 \Leftrightarrow S_2, \]


\[ Sibling(KingJohn, Richard) \Rightarrow Sibling(Richard, KingJohn) \]

\[ >(1,2) \lor \leq (1,2) \]

\[ >(1,2) \land \neg >(1,2) \]
Truth in first-order logic

- Sentences are true with respect to a model and an interpretation.
- Model contains objects (domain elements) and relations among them.
- Interpretation specifies referents for:
  - constant symbols → objects
  - predicate symbols → relations
  - function symbols → functional relations
- An atomic sentence $\text{predicate}(\text{term}_1, \ldots, \text{term}_n)$ is true iff the objects referred to by $\text{term}_1, \ldots, \text{term}_n$ are in the relation referred to by $\text{predicate}$.
Model Example
Universal quantification

∀<variables> <sentence>

Everyone at NUS is smart:
∀x At(x,NUS) ⇒ Smart(x)

∀x \( P \) is true in a model \( m \) iff \( P \) is true with \( x \) being each possible object in the model

Roughly speaking, equivalent to the conjunction of instantiations of \( P \)

\[
\begin{align*}
\text{At}(\text{KingJohn},\text{NUS}) & \Rightarrow \text{Smart}(\text{KingJohn}) \\
\& \text{At}(\text{Richard},\text{NUS}) & \Rightarrow \text{Smart}(\text{Richard}) \\
\& \text{At}(\text{NUS},\text{NUS}) & \Rightarrow \text{Smart}(\text{NUS}) \\
\& \ldots
\end{align*}
\]
A common mistake to avoid

- Typically, $\Rightarrow$ is the main connective with $\forall$

- Common mistake: using $\land$ as the main connective with $\forall$:
  $\forall x \text{ At}(x,\text{NUS}) \land \text{Smart}(x)$
  means
Existential quantification

- $\exists <variables> <sentence>$

Someone at NUS is smart:
$\exists x \text{At}(x,\text{NUS}) \land \text{Smart}(x)$

- $\exists x P$ is true in a model $m$ iff $P$ is true with $x$ being some possible object in the model

- Roughly speaking, equivalent to the disjunction of instantiations of $P$
  
  $\text{At}(\text{KingJohn},\text{NUS}) \land \text{Smart}(\text{KingJohn})$
  $\lor \text{At}(\text{Richard},\text{NUS}) \land \text{Smart}(\text{Richard})$
  $\lor \text{At}(\text{NUS},\text{NUS}) \land \text{Smart}(\text{NUS})$
  $\lor \ldots$
Another common mistake

- Typically, $\land$ is the main connective with $\exists$

- Common mistake: using $\Rightarrow$ as the main connective with $\exists$:
  $$\exists x \text{ At}(x, \text{NUS}) \Rightarrow \text{Smart}(x)$$

  is true if
Properties of quantifiers

- $\forall x \forall y$ is the same as $\forall y \forall x$
- $\exists x \exists y$ is the same as $\exists y \exists x$
- $\exists x \forall y$ is not the same as $\forall y \exists x$

$\exists x \forall y \text{ Loves}(x,y)$
- “There is a person who loves everyone in the world”

$\forall y \exists x \text{ Loves}(x,y)$
- “Everyone in the world is loved by at least one person”

 Quantifier duality: each can be expressed using the other
- $\forall x \text{ Likes}(x,\text{IceCream}) \equiv \neg \exists x \neg \text{Likes}(x,\text{IceCream})$
- $\exists x \text{ Likes}(x,\text{Broccoli}) \equiv \neg \forall x \neg \text{Likes}(x,\text{Broccoli})$
Equality

- $\text{term}_1 = \text{term}_2$ is true under a given interpretation if and only if $\text{term}_1$ and $\text{term}_2$ refer to the same object.

- E.g., definition of $\text{Sibling}$ in terms of $\text{Parent}$:

$$\forall x, y \; \text{Sibling}(x, y) \iff \neg (x = y) \land \exists m, f \; \neg (m = f) \land \text{Parent}(m, x) \land \text{Parent}(f, x) \land \text{Parent}(m, y) \land \text{Parent}(f, y)$$
Using FOL

In the kinship domain:

- Brothers are siblings
  \[ \forall x, y \ Brother(x, y) \Rightarrow Sibling(x, y) \]

- One's mother is one's female parent
  \[ \forall m, c \ Mother(c) = m \iff (Female(m) \land Parent(m, c)) \]

- “Sibling” is symmetric
  \[ \forall x, y \ Sibling(x, y) \iff Sibling(y, x) \]
Using FOL

The set domain:

- $\forall s \text{ Set}(s) \iff (s = \{\} ) \vee (\exists x, s_2 \text{ Set}(s_2) \land s = \{x|s_2\})$
- $\neg \exists x, s \{x|s\} = \{\}$
- $\forall x, s \ x \in s \iff s = \{x|s\}$
- $\forall x, s \ x \in s \iff [\exists y, s_2 \ (s = \{y|s_2\} \land (x = y \lor x \in s_2))]$
- $\forall s_1, s_2 \ s_1 \subseteq s_2 \iff (\forall x \ x \in s_1 \Rightarrow x \in s_2)$
- $\forall s_1, s_2 \ (s_1 = s_2) \iff (s_1 \subseteq s_2 \land s_2 \subseteq s_1)$
- $\forall x, s_1, s_2 \ x \in (s_1 \cap s_2) \iff (x \in s_1 \land x \in s_2)$
- $\forall x, s_1, s_2 \ x \in (s_1 \cup s_2) \iff (x \in s_1 \lor x \in s_2)$
Interacting with FOL KBs

Suppose a wumpus-world agent is using an FOL KB and perceives a smell and a breeze (but no glitter) at $t=5$:

$\text{Tell}(KB, \text{Percept}([\text{Smell}, \text{Breeze}, \text{None}], 5))$

$\text{Ask}(KB, \exists a \ \text{BestAction}(a, 5))$

i.e., does the KB entail some best action at $t=5$?

Answer: Yes, $\{a/\text{Shoot}\}$ ← substitution (binding list)

Given a sentence $S$ and a substitution $\sigma$,

$S\sigma$ denotes the result of plugging $\sigma$ into $S$; e.g.,

$S = \text{Smarter}(x, y)$

$\sigma = \{x/\text{Hillary}, y/\text{Bill}\}$

$S\sigma = \text{Smarter}(\text{Hillary}, \text{Bill})$

$\text{Ask}(KB, S)$ returns some/all $\sigma$ such that $KB \models \sigma$
KB for the wumpus world

- **Perception**
  - $\forall t,s,b \text{ Percept}([s,b,\text{Glitter}],t) \Rightarrow \text{Glitter}(t)$

- **Reflex**
  - $\forall t \text{ Glitter}(t) \Rightarrow \text{BestAction}(*\text{Grab},t)$
Deducing hidden properties

- $\forall x,y,a,b \ Adjacent([x,y],[a,b]) \iff [a,b] \in \{[x+1,y], [x-1,y],[x,y+1],[x,y-1]\}$

Properties of squares:
- $\forall s,t \ At(Agent,s,t) \land \ Breeze(t) \Rightarrow Breezy(s)$

Squares are breezy near a pit:
- **Diagnostic** rule - infer cause from effect
  $\forall s \ Breezy(s) \Rightarrow \exists r \ Adjacent(r,s) \land Pit(r)$
- **Causal** rule - infer effect from cause
  $\forall r \ Pit(r) \Rightarrow [\forall s \ Adjacent(r,s) \Rightarrow Breezy(s)]$
Knowledge engineering in FOL

1. Identify the task
2. Assemble the relevant knowledge
3. Decide on a vocabulary of predicates, functions, and constants
4. Encode general knowledge about the domain
5. Encode a description of the specific problem instance
6. Pose queries to the inference procedure and get answers
7. Debug the knowledge base
The electronic circuits domain

One-bit full adder
The electronic circuits domain

1. Identify the task
   - Does the circuit actually add properly? (circuit verification)

2. Assemble the relevant knowledge
   - Composed of wires and gates; Types of gates (AND, OR, XOR, NOT)
   - Irrelevant: size, shape, color, cost of gates

3. Decide on a vocabulary
   - Alternatives:
     - Type($X_1$) = XOR
     - Type($X_1$, XOR)
     - XOR($X_1$)
The electronic circuits domain

4. Encode general knowledge of the domain
   - $\forall t_1, t_2 \text{ Connected}(t_1, t_2) \implies \text{Signal}(t_1) = \text{Signal}(t_2)$
   - $\forall t \text{ Signal}(t) = 1 \lor \text{Signal}(t) = 0$
   - $1 \neq 0$
   - $\forall t_1, t_2 \text{ Connected}(t_1, t_2) \implies \text{Connected}(t_2, t_1)$
   - $\forall g \text{ Type}(g) = \text{OR} \implies \text{Signal}(\text{Out}(1,g)) = 1 \iff \exists n \text{ Signal}(\text{In}(n,g)) = 1$
   - $\forall g \text{ Type}(g) = \text{AND} \implies \text{Signal}(\text{Out}(1,g)) = 0 \iff \exists n \text{ Signal}(\text{In}(n,g)) = 0$
   - $\forall g \text{ Type}(g) = \text{XOR} \implies \text{Signal}(\text{Out}(1,g)) = 1 \iff \text{Signal}(\text{In}(1,g)) \neq \text{Signal}(\text{In}(2,g))$
   - $\forall g \text{ Type}(g) = \text{NOT} \implies \text{Signal}(\text{Out}(1,g)) \neq \text{Signal}(\text{In}(1,g))$
5. Encode the specific problem instance

Type($X_1$) = XOR  
Type($A_1$) = AND  
Type($O_1$) = OR

Type($X_2$) = XOR  
Type($A_2$) = AND

Connected(Out(1,$X_1$),In(1,$X_2$))
Connected(Out(1,$X_1$),In(2,$A_2$))
Connected(Out(1,$A_2$),In(1,$O_1$))
Connected(Out(1,$A_1$),In(2,$O_1$))
Connected(Out(1,$X_2$),Out(1,$C_1$))
Connected(Out(1,$O_1$),Out(2,$C_1$))

Connected(In(1,$C_1$),In(1,$X_1$))
Connected(In(1,$C_1$),In(1,$A_1$))
Connected(In(2,$C_1$),In(2,$X_1$))
Connected(In(2,$C_1$),In(2,$A_1$))
Connected(In(3,$C_1$),In(2,$X_2$))
Connected(In(3,$C_1$),In(1,$A_2$))
6. Pose queries to the inference procedure
What are the possible sets of values of all the terminals for the adder circuit?

∃i_1,i_2,i_3,o_1,o_2 \text{ Signal(In}(1,C_1)) = i_1 \land \text{ Signal(In}(2,C_1)) = i_2 \land \text{ Signal(In}(3,C_1)) = i_3 \land \text{ Signal(Out}(1,C_1)) = o_1 \land \text{ Signal(Out}(2,C_1)) = o_2

7. Debug the knowledge base
May have omitted assertions like 1 ≠ 0
First-order logic:
- objects and relations are semantic primitives
- syntax: constants, functions, predicates, equality, quantifiers

Increased expressive power: sufficient to define wumpus world
What is Logic Programming?

- A type of programming consisting of facts and relationships from which the programming language can draw a conclusion.
  - In *imperative programming* languages, we tell the computer what to do by programming the procedure by which program states and variables are modified.
  - In contrast, in *logical programming*, we don’t tell the computer exactly what it should do (i.e., how to derive a conclusion). User-provided facts and relationships allow it to derive answers via logical inference.

- Prolog is the most widely used logic programming language.
Prolog Features

- Prolog uses **logical variables**. These are not the same as variables in other languages. Programmers can use them as ‘holes’ in data structures that are gradually filled in as computation proceeds.

- **Unification** is a built-in term-manipulation method that passes parameters, returns results, selects and constructs data structures.

- Basic control flow model is **backtracking**.

- **Program clauses and data** have the same form.
  - A Prolog program can also be seen as a relational database containing rules as well as facts.
Example: Concatenate lists a and b

In an imperative language

```c
list procedure cat(list a, list b)
{
    list t = list u = copylist(a);
    while (t.tail != nil) t = t.tail;
    t.tail = b;
    return u;
}
```

In a functional language

```
cat(a,b) ≡
if b = nil then a
else cons(head(a), cat(tail(a),b))
```

In a declarative language

```
cat([], Z, Z).
cat([H|T], L, [H|Z]) :- cat(T, L, Z).
```
Outline

- General Syntax
- Terms
- Operators
- Rules
- Queries
.pl files contain lists of clauses
Clauses can be either facts or rules

Predicate, arity 1 (male/1)
Terminates a clause

Argument to predicate

Indicates a rule

“and”

No space between functor and argument list
Complete Syntax of Terms

**Term**
- **Constant**
  - *Names an individual*
  - **Atom**
    - names an individual that has parts
  - **Number**
    - names an individual
    - unable to be named when program is written
- **Compound Term**
- **Variable**
  - *Stands for an individual*

### Examples
- **Atom**
  - alpha17
  - gross_pay
  - john_smith
  - dyspepsia
- **Number**
  - 0
  - 1
  - 57
  - 1.618
  - 2.04e-27
  - -13.6
  - '12Q&A'
- **Compound Term**
  - likes(john, mary)
  - book(dickens, Z, cricket)
  - f(x)
  - [1, 3, g(a), 7, 9]
  - -(+(15, 17), t)
  - 15 + 17 - t
- **Variable**
  - X
  - Gross_pay
  - Diagnosis
  - _257

### N.B.
- case of variables and terms and constants switched from FOL

A list is made of terms, separated by commas and enclosed by brackets.
The parents of Spot are Fido and Rover.

Functor (an atom) of arity 3. components (any terms)

It is possible to depict the term as a tree:

```
parents
  
spot   fido   rover
```
Examples of operator properties

Prolog has shortcuts in notation for certain operators (especially arithmetic ones)

<table>
<thead>
<tr>
<th>Position</th>
<th>Operator Syntax</th>
<th>Normal Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefix:</td>
<td>-2</td>
<td>-(2)</td>
</tr>
<tr>
<td>Infix:</td>
<td>5+17</td>
<td>+(17,5)</td>
</tr>
</tbody>
</table>

Associativity: left, right, none.
- \( X+Y+Z \) is parsed as \( (X+Y)+Z \) because addition is left-associative.

Precedence: an integer.
- \( X+Y*Z \) is parsed as \( X+(Y*Z) \) because multiplication has higher precedence.

These are all the same as the normal rules of arithmetic.
Rules

- Rules combine facts to increase knowledge of the system

\[ \text{son}(X,Y) :- \]
\[ \quad \text{male}(X), \text{child}(X,Y). \]

- X is a son of Y if X is male and X is a child of Y
Interpretation of Rules

Rules can be given a declarative reading or a procedural reading.

**Form of rule:**

\[ H :\!\!\!: G_1, G_2, ..., G_n. \]

**Declarative reading:**

“That H is provable follows from goals \( G_1, G_2, ..., G_n \) being provable.”

**Procedural reading:**

“To execute procedure H, the procedures called by goals \( G_1, G_2, ..., G_n \) are executed first.”
Queries

- Prolog is interactive; you load a KB and then ask queries
- Composed at the ?- prompt
- Returns values of bound variables and yes or no

?- son(bob, harry).
yes
?- king(bob, france).
no
Another example

likes(george,kate).
likes(george,susie).
likes(george,wine).

?- likes(george,X)
X = kate
; 
X = susie
; 
X = wine
; 
no

Answer: kate or susie or wine or false
Quantifiers

When a variable appears in the specification of a database, the variable is universally quantified. Example:

likes(susie,Y) One interpretation: ‘Susie likes everyone’

For the existential quantifier one may do two things:

a. Enter the value directly into the database
   likes(george,Z) becomes likes(george,wine)

b. Query the interpreter
   ?- likes(george,Z) returns a value for Z if one exists
Points to consider

- Variables are bound by Prolog, not by the programmer
  - You can’t assign a value to a variable.

- Successive user prompts ; cause the interpreter to return all terms that can be substituted for X.
  - They are returned in the order found.
  - Order is important

- PROLOG adopts the closed-world assumption:
  - All knowledge of the world is present in the database.
  - If a term is not in the database assume is false.
  - Prolog’s ‘yes’ = I can prove it, ‘no’ = I can’t prove it.

Two things to think about:
When would the closed-world assumption lead to false inferences? When would the different ordering of solutions cause problems?
Queries

- Can bind answers to questions to variables
- Who is bob the son of? (X=harry)
  ?- son(bob, X).
- Who is male? (X=bob, harry)
  ?- male(X).
- Is bob the son of someone? (yes)
  ?- son(bob, _).

_ = Anonymous variable, don’t care what it’s bound to.
Lists

- The first element of a list can be separated from the tail using operator |.

Example:

Match the list [tom,dick,harry,fred] to

- \([X|Y]\) then \(X = \text{tom}\) and \(Y = [\text{dick},\text{harry},\text{fred}]\)
- \([X,Y|Z]\) then \(X = \text{tom}\), \(Y = \text{dick}\), and \(Z = [\text{harry},\text{fred}]\)
- \([V,W,X,Y,Z|U]\) will not match
- \([\text{tom},X|[\text{harry},\text{fred}]\)] gives \(X = \text{dick}\)
Example: List Membership

- We want to write a function `member` that works as follows:

```prolog
?- member(a,[a,b,c,d,e])
yes
?- member(a,[1,2,3,4])
no
?- member(X,[a,b,c])
X = a
;  
X = b
;  
X = c
;  
no
```

Can you do it?
Function Membership Solution

Define two predicates:

- `member(X,[X|T]).`
- `member(X,[Y|T]) :- member(X,T).`

A more elegant definition uses anonymous variables:

- `member(X,[X,_]).`
- `member(X,[_,T]) :- member(X,T).`

Again, the symbol `_` indicates that the contents of that variable is unimportant.
Notes on running Prolog

You will often want to load a KB on invocation of Prolog

- Use “consult(‘mykb.pl’).” at the “?-” prompt.
- Or add it on the command line as a standard input “pl < mykb.pl”

If you want to modify facts once Prolog is invoked:

- Use “assert(p).”
- Or “retract(p).” to remove a fact
● A Prolog program is a set of specifications in FOL. The specification is known as the database of the system.
● Prolog is an interactive language (the user enters queries in response to a prompt).
● PROLOG adopts the closed-world assumption

● How does Prolog find the answer(s)? We return to this next week in *Inference in FOL*