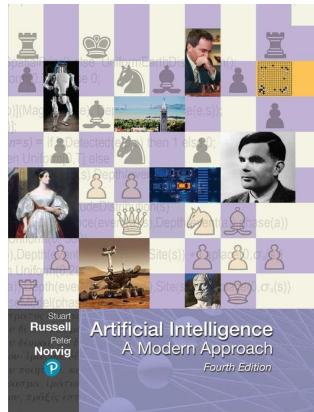


Artificial Intelligence: A Modern Approach

Fourth Edition



Chapter 8

First-order logic

Outline

- ◆ Why FOL?
- ◆ Syntax and semantics of FOL
- ◆ Fun with sentences
- ◆ Wumpus world in FOL
- ◆ Knowledge Engineering in FOL

Pros and cons of propositional logic

- ☺ Propositional logic is **declarative**: pieces of syntax correspond to facts
- ☺ Propositional logic allows partial/disjunctive/negated information (unlike most data structures and databases)
- ☺ Propositional logic is **compositional**: meaning of $B_{1,1} \wedge P_{1,2}$ is derived from meaning of $B_{1,1}$ and of $P_{1,2}$
- ☺ Meaning in propositional logic is **context-independent** (unlike natural language, where meaning depends on context)
- ☺ Propositional logic has very limited expressive power (unlike natural language)
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 world contains **facts**, first-order logic (like natural language) assumes the world contains

- **Objects**: people, houses, numbers, theories, Ronald McDonald, colors, baseball games, wars, centuries . . .
- **Relations**: red, round, bogus, prime, multistoried . . ., brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, . . .
- **Functions**: father of, best friend, third inning of, one more than, end of
. . .

Logics in general

Language	Ontological Commitment	Epistemological Commitment
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	n true/false/unknown
Temporal logic	facts, objects, relations, times	n true/false/unknown
Probability theory	facts	n degree of belief
Fuzzy logic	facts + degree of truth	known interval value

Syntax of FOL: Basic elements

Constants	<i>KingJohn, 2, UCB, .</i>
Predicates	<i>.</i>
Functions	<i>Brother, >, . . .</i>
Variables	<i>Sqrt, LeftLegOf, . . .</i>
Connectiv	<i>. x, y, a, b, . . .</i>
es	<i>^ V ~ ⇒ ⇔</i>
Equality	<i>=</i>
Quantifier	<i>∀ ∃</i>
S	

Atomic sentences

Atomic sentence = $\text{predicate}(\text{term}_1, \dots, \text{term}_n)$
or $\text{term}_1 = \text{term}_2$

Term = $\text{function}(\text{term}_1, \dots, \text{term}_n)$
or *constant* or *variable*

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

Truth in first-order logic

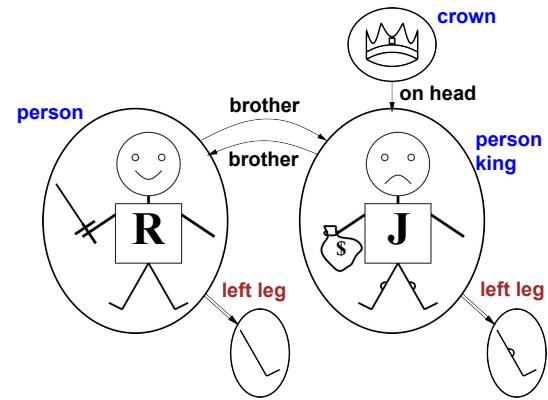
Sentences are true with respect to a *model* and an *interpretation*
Model contains ≥ 1 objects (*domain elements*) and relations

among them Interpretation specifies referents for

constant symbols →
objects predicate symbols
→ relations
function symbols → functional relations

An atomic sentence $\text{predicate}(\text{term}_1, \dots, \text{term}_n)$
is true iff the objects referred to by $\text{term}_1, \dots, \text{term}_n$
are in the relation referred to by *predicate*

Models for FOL: Example



Truth example

Consider the interpretation in
which *Richard* → Richard the
Lionheart *John* → the evil
King John
Brother → the brotherhood relation

Under this interpretation, $\text{Brother}(\text{Richard}, \text{John})$ is
true just in case Richard the Lionheart and the evil
King John are in the brotherhood relation in the
model

Models for FOL: Lots!

Entailment in propositional logic can be computed by enumerating
models

We can enumerate the FOL models for a given KB

vocabulary: For each number of domain elements n
from 1 to ∞

For each k -ary predicate P_k in the
vocabulary For each possible k -ary
relation on n objects

For each constant symbol C in the vocabulary
For each choice of referent for C from n objects . . .

Computing entailment by enumerating FOL models is not easy!

Universal quantification

\forall }variables }sentences

Everyone at Berkeley is smart:

$$\forall x \ At(x, \text{Berkeley}) \Rightarrow \text{Smart}(x)$$

$\forall x \ P$ 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

$$\begin{aligned} & (At(\text{King John}, \text{Berkeley}) \Rightarrow \text{Smart}(\text{King John})) \\ & \wedge (At(\text{Richard}, \text{Berkeley}) \Rightarrow \text{Smart}(\text{Richard})) \\ & \wedge (At(\text{Berkeley}, \text{Berkeley}) \Rightarrow \text{Smart}(\text{Berkeley})) \\ & \wedge \dots \end{aligned}$$

A common mistake to avoid

Typically, \Rightarrow is the main connective with \forall

Common mistake: using \wedge as the main connective with \forall :

$$\forall x \ At(x, \text{Berkeley}) \wedge \text{Smart}(x)$$

means "Everyone is at Berkeley and everyone is smart"

Existential quantification

\exists }variables }sentences

Someone at Stanford is smart:

$$\exists x \ At(x, \text{Stanford}) \wedge \text{Smart}(x)$$

$\exists x \ P$ 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

$$\begin{aligned} & (At(\text{King John}, \text{Stanford}) \wedge \text{Smart}(\text{King John})) \\ & \vee (At(\text{Richard}, \text{Stanford}) \wedge \text{Smart}(\text{Richard})) \\ & \vee (At(\text{Stanford}, \text{Stanford}) \wedge \text{Smart}(\text{Stanford})) \\ & \vee \dots \end{aligned}$$

Another common mistake to avoid

Typically, \wedge is the main connective with \exists

Common mistake: using \Rightarrow as the main connective with \exists :

$$\exists x \ At(x, \text{Stanford}) \Rightarrow \text{Smart}(x)$$

is true if there is anyone who is not at Stanford!

Properties of quantifiers

$\forall \ \forall$ is the same as \forall (why??)

$x \ y \ \forall y$ x

$\exists \ \exists$ is the same as \exists (why??)

$x \ y \ \exists y$ x

$\exists \ \forall$ is not the same $\forall \ \exists$

$\exists x \ y \ \forall y$ as $Loves(x, y)$

"There is a person who loves everyone in the world"

$$\forall y \ \exists x \ Loves(x, y)$$

"Everyone in the world is loved by at least one person"

$\exists x \ Likes(x, \text{IceCream})$: each can be expressed using the other

$$\begin{aligned} \exists x \ Likes(x, \text{Broccoli}) & \quad \neg \forall x \ \neg Likes(x, \text{Broccoli}) \\ \exists x \ Likes(x, \text{Broccoli}) & \end{aligned}$$

Fun with sentences

Brothers are
siblings

Knowledge base for the wumpus world

"Perception"

$$\begin{aligned} \forall b, g, t \quad P \text{ercept}([Smell, b, g], t) &\Rightarrow Smelt(t) \\ \forall s, b, t \quad P \text{ercept}([s, b, Glitter], t) &\Rightarrow AtGold(t) \end{aligned}$$

$$\text{Reflex: } \forall t AtGold(t) \Rightarrow Action(Grab, t)$$

Reflex with internal state: do we have the gold already?

$$\forall t AtGold(t) \wedge \neg Holding(Gold, t) \Rightarrow Action(Grab, t)$$

Holding(Gold, t) cannot be observed

→ keeping track of change is essential

Deducing hidden properties

Properties of locations:

$$\begin{aligned} \forall x, t \quad At(Agent, x, t) \wedge Smelt(t) &\Rightarrow Smelly(x) \\ \forall x, t \quad At(Agent, x, t) \wedge Breeze(t) &\Rightarrow Breezy(x) \end{aligned}$$

Squares are breezy near a pit:

Diagnostic rule—infer cause from effect

$$\forall y Breezy(y) \Rightarrow \exists x Pit(x) \wedge Adjacent(x, y)$$

Causal rule—infer effect from cause

$$\forall x, y Pit(x) \wedge Adjacent(x, y) \Rightarrow Breezy(y)$$

Neither of these is complete—e.g., the causal rule doesn't say whether squares far away from pits can be breezy

Definition for the *Breezy* predicate:

$$\forall y Breezy(y) \Leftrightarrow [\exists x Pit(x) \wedge Adjacent(x, y)]$$

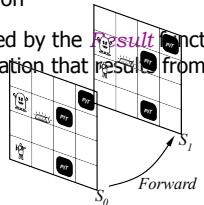
Keeping track of change

Facts hold in **situations**, rather than eternally
E.g., *Holding(Gold, Now)* rather than just
Holding(Gold)

Situation calculus is one way to represent change in FOL:

Adds a situation argument to each non- eternal predicate E.g., *Now* in *Holding(Gold, Now)* denotes a situation

Situations are connected by the *Result* function
Result(a, s) is the situation that results from doing *a* in *s*



Describing actions I

"Effect" axiom—describe changes due to action

$$\forall s AtGold(s) \Rightarrow Holding(Gold, Result(Grab, s))$$

"Frame" axiom—describe non-changes due to action

$$\forall s HaveArrow(s) \Rightarrow HaveArrow(Result(Grab, s))$$

Frame problem: find an elegant way to handle non-change

- representation—avoid frame axioms
- inference—avoid repeated "copy-overs" to keep track of state

Qualification problem: true descriptions of real actions require endless caveats— what if gold is slippery or nailed down or . . .

Ramification problem: real actions have many secondary consequences— what about the dust on the gold, wear and tear on gloves, . . .

Describing actions II

Successor-state axioms solve the representational frame problem

Each axiom is "about" a **predicate** (not an action per se):
 $P \text{ true} \Leftrightarrow [\text{an action made } P \text{ true}]$
 afterwards $\vee P \text{ true already and no action made } P \text{ false}]$

For holding the gold:

$$\begin{aligned} \forall a, s \quad Holding(Gold, Result(a, s)) \\ \Leftrightarrow \\ [(a = Grab \wedge AtGold(s)) \\ \vee (Holding(Gold, s) \wedge a = Release)] \end{aligned}$$

Making plans

Initial condition in KB:

$$\begin{aligned} At(Agent, [1, 1], S_0) \\ At(Gold, [1, 2], S_0) \end{aligned}$$

Query: *Ask(KB, $\exists s Holding(Gold, s)$)*
i.e., in what situation will I be holding the gold?

Answer: $\{s / Result(Grab, Result(Forward, S_0))\}$
i.e., go forward and then grab the gold

This assumes that the agent is interested in plans starting at S_0 and that S_0 is the only situation described in the KB

Making plans: A better way

Represent **plans** as action sequences $[a_1, a_2, \dots, a_n]$

PlanResult(p, s) is the result of executing p in s

Then the query **Ask(KB, $\exists p \text{ Holding(Gold, PlanResult}(p, S_0))$**
has the solution $\{p/\text{[Forward, Grab]}\}$

Definition of **PlanResult** in terms of **Result**:

$$\begin{aligned} \forall s \quad \text{PlanResult}([], s) &= s \\ \forall a, p, s \quad \text{PlanResult}([a[p], s]) &= \text{PlanResult}(p, \text{Result}(a, s)) \end{aligned}$$

Planning systems are special-purpose reasoners designed to do this type of inference more efficiently than a general-purpose reasoner

Knowledge Engineering in FOL

Knowledge engineering: the general process of knowledge-base construction.

The steps used in the knowledge engineering process:

1. Identify the questions.
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 problem instance
6. Pose queries to the inference procedure and get answers
7. Debug and evaluate the knowledge base

Knowledge Engineering in FOL

Applications in the electronic circuits domain

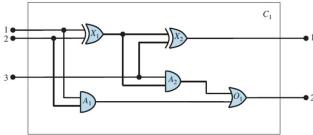


Figure 8.6 A digital circuit C_1 , purporting to be a one-bit full adder. The first two inputs are the two bits to be added, and the third input is a carry bit. The first output is the sum, and the second output is a carry bit for the next adder. The circuit contains two XOR gates, two AND gates, and one OR gate.

1. Identify the questions

- Does the circuit in Figure 8.6 actually add properly?
- If all the inputs are high, what is the output of gate A2?
- Questions about the circuit's structure are also interesting.
- For example, what are all the gates connected to the first input terminal?
- Does the circuit contain feedback loops?

Knowledge Engineering in FOL

Applications in the electronic circuits domain

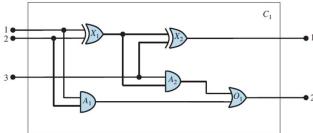


Figure 8.6 A digital circuit C_1 , purporting to be a one-bit full adder. The first two inputs are the two bits to be added, and the third input is a carry bit. The first output is the sum, and the second output is a carry bit for the next adder. The circuit contains two XOR gates, two AND gates, and one OR gate.

3. Decide on a vocabulary

- Each gate is represented as an object named by a constant, about which we assert that it is a gate with
- $\text{Gate}(X_1)$, eg: $\text{Type}(X_1)=\text{XOR}$
- $\text{Circuit}(C_1)$
- $\text{Terminal}(x)$

Knowledge Engineering in FOL

Applications in the electronic circuits domain

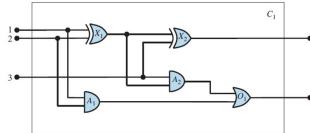


Figure 8.6 A digital circuit C_1 , purporting to be a one-bit full adder. The first two inputs are the two bits to be added, and the third input is a carry bit. The first output is the sum, and the second output is a carry bit for the next adder. The circuit contains two XOR gates, two AND gates, and one OR gate.

4. Encode general knowledge of the domain

- Example:
 - If two terminals are connected, then they have the same signal:
$$\forall t_1, t_2 \text{ Terminal}(t_1) \wedge \text{Terminal}(t_2) \wedge \text{Connected}(t_1, t_2) \Rightarrow \text{Signal}(t_1)=\text{Signal}(t_2)$$

Applications in the electronic circuits domain

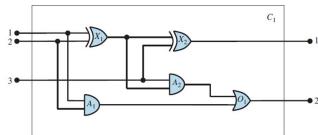


Figure 8.6 A digital circuit C_1 , purporting to be a one-bit full adder. The first two inputs are the two bits to be added, and the third input is a carry bit. The first output is the sum, and the second output is a carry bit for the next adder. The circuit contains two XOR gates, two AND gates, and one OR gate.

5. Encode the specific problem instance

- Categorize the circuit and its component gates & show the connections:

Connected(Out(1,X₁), In(1,X₂)) Connected(In(1,C₁); In(1,X₁))

Applications in the electronic circuits domain

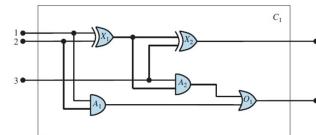


Figure 8.6 A digital circuit C_1 , purporting to be a one-bit full adder. The first two inputs are the two bits to be added, and the third input is a carry bit. The first output is the sum, and the second output is a carry bit for the next adder. The circuit contains two XOR gates, two AND gates, and one OR gate.

6. Pose queries to the inference procedure

- What are the possible sets of values of all the terminals for the adder circuit?
- This final query will return a complete input–output table for the device, which can be used

Applications in the electronic circuits domain

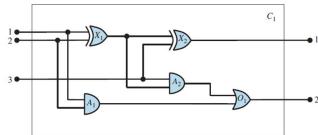


Figure 8.6 A digital circuit C_1 , purporting to be a one-bit full adder. The first two inputs are the two bits to be added, and the third input is a carry bit. The first output is the sum, and the second output is a carry bit for the next adder. The circuit contains two XOR gates, two AND gates, and one OR gate.

7. Debug the knowledge base

- We can perturb the knowledge base in various ways to see what kinds of erroneous behaviors emerge
- Example if no assertion $1 \neq 0$

First-order logic:

- objects and relations are semantic primitives
- syntax: constants, functions, predicates, equality, quantifiers

Increased expressive power: sufficient to define

wumpus world

Situation calculus:

- conventions for describing actions and change in FOL
- can formulate planning as inference on a situation calculus KB

Developing a KB in FOL requires a careful process of analyzing the domain, choosing a vocabulary, and encoding the axioms required to support the desired inferences.

Applications in the electronic circuits domain



Figure 8.6 A digital circuit C_1 , purporting to be a one-bit full adder. The first two inputs are the two bits to be added, and the third input is a carry bit. The first output is the sum, and the second output is a carry bit for the next adder. The circuit contains two XOR gates, two AND gates, and one OR gate.

First-order logic:

- objects and relations are semantic primitives
- syntax: constants, functions, predicates, equality, quantifiers

Increased expressive power: sufficient to define

wumpus world

Situation calculus:

- conventions for describing actions and change in FOL
- can formulate planning as inference on a situation calculus KB

Developing a KB in FOL requires a careful process of analyzing the domain, choosing a vocabulary, and encoding the axioms required to support the desired inferences.

Applications in the electronic circuits domain

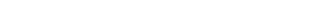


Figure 8.6 A digital circuit C_1 , purporting to be a one-bit full adder. The first two inputs are the two bits to be added, and the third input is a carry bit. The first output is the sum, and the second output is a carry bit for the next adder. The circuit contains two XOR gates, two AND gates, and one OR gate.

First-order logic:

- objects and relations are semantic primitives
- syntax: constants, functions, predicates, equality, quantifiers

Increased expressive power: sufficient to define

wumpus world

Situation calculus:

- conventions for describing actions and change in FOL
- can formulate planning as inference on a situation calculus KB

Developing a KB in FOL requires a careful process of analyzing the domain, choosing a vocabulary, and encoding the axioms required to support the desired inferences.