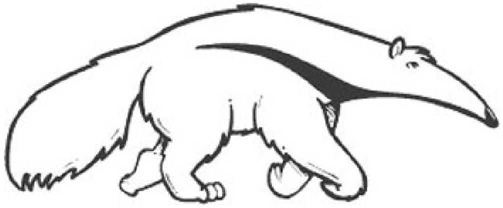
# First-Order Logic C: Knowledge Engineering

Introduction to Artificial Intelligence
Prof. Richard Lathrop



**Read Beforehand: R&N 8, 9.1-9.2, 9.5.1-9.5.5**

BREN:ICS
INFORMATION AND COMPUTER SCIENCES

UNIVERSITY of CALIFORNIA IRVINE

# Outline

- Review --- Syntactic Ambiguity
- Using FOL
  - Tell, Ask
- Example: Wumpus world
- Deducing Hidden Properties
  - Keeping track of change
  - Describing the results of Actions
- Set Theory in First-Order Logic
- Knowledge engineering in FOL
- The electronic circuits domain

# You will be expected to know

- Seven steps of Knowledge Engineering (R&N section 8.4.1)

- Given a simple Knowledge Engineering problem, produce a simple FOL Knowledge Base that solves the problem

# Review --- Syntactic Ambiguity

- FOPC provides many ways to represent the same thing.
- E.g., “Ball-5 is red.”
  - HasColor(Ball-5, Red)
    - Ball-5 and Red are objects related by HasColor.
  - Red(Ball-5)
    - Red is a unary predicate applied to the Ball-5 object.
  - HasProperty(Ball-5, Color, Red)
    - Ball-5, Color, and Red are objects related by HasProperty.
  - ColorOf(Ball-5) = Red
    - Ball-5 and Red are objects, and ColorOf() is a function.
  - HasColor(Ball-5(), Red())
    - Ball-5() and Red() are functions of zero arguments that both return an object, which objects are related by HasColor.
  - …

- This can GREATLY confuse a pattern-matching reasoner.
  - Especially if multiple people collaborate to build the KB, and they all have different representational conventions.

# Review --- Syntactic Ambiguity --- Partial Solution

- FOL can be TOO expressive, can offer TOO MANY choices
- Likely confusion, especially for **teams** of Knowledge Engineers
- Different team members can make different representation choices
  - E.g., represent “Ball43 is Red.” as:
    - a predicate (= verb)? E.g., “Red(Ball43)” ?
    - an object (= noun)? E.g., “Red = Color(Ball43))” ?
    - a property (= adjective)? E.g., “HasProperty(Ball43, Red)” ?
- PARTIAL SOLUTION:
  - An upon-agreed **ontology** that settles these questions
  - Ontology = what exists in the world & how it is represented
  - The Knowledge Engineering teams agrees upon an ontology BEFORE they begin encoding knowledge

# Using FOL

- We want to TELL things to the KB, e.g.

  TELL(KB, $\forall$ x King(x) $\Rightarrow$ PersonX) )

  TELL(KB, *King(John)* )

  These sentences are assertions

- We also want to ASK things to the KB,

  ASK(KB, $\exists$ x Person(x) )

  these are queries or goals

  The KB should return the list of x's for which Person(x) is true:

  {x/John, x/Richard,...}

# 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

# FOL Version of Wumpus World

- Typical percept sentence:
  Percept([Stench,Breeze,Glitter,None,None],5)
- Actions:
  Turn(Right), Turn(Left), Forward, Shoot, Grab, Release, Climb
- To determine best action, construct query:

  $\exists$ a BestAction(a,5)

- ASK solves this and returns {a/Grab}
  - And TELL about the action.

# Knowledge Base for Wumpus World

- Perception
  - $\forall$s,g,x,y,t Percept([s,Breeze,g,x,y],t) $\Rightarrow$ Breeze(t)
  - $\forall$s,b,x,y,t Percept([s,b,Glitter,x,y],t) $\Rightarrow$ Glitter(t)

- Reflex action
  - $\forall$t Glitter(t) $\Rightarrow$ BestAction(Grab,t)

- Reflex action with internal state
  - $\forall$t Glitter(t) $\wedge\neg$Holding(Gold,t) $\Rightarrow$ BestAction(Grab,t)

Holding(Gold,t) can not be observed: keep track of change.

# Deducing hidden properties

Environment definition:

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

Properties of locations:

$\forall$s,t *At*(Agent,s,t) $\wedge$ Breeze(t) $\Rightarrow$ Breezy(s)

Squares are breezy near a pit:

- Diagnostic rule---infer cause from effect
  $\forall$s Breezy(s) $\Leftrightarrow$ $\exists$ r Adjacent(r,s) $\wedge$ Pit(r)

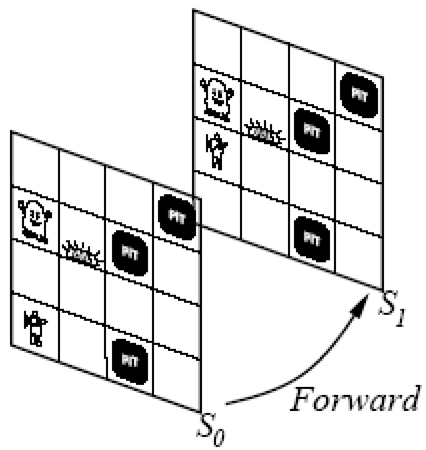
- Causal rule---infer effect from cause (model based reasoning)
  $\forall$r Pit(r) $\Rightarrow$ [$\forall$s Adjacent(r,s) $\Rightarrow$ Breezy(s)]

# 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 $s$



# Yale shooting problem

- The **Yale shooting problem** illustrates the frame problem. (Its inventors were working at Yale University when they proposed it.)
- Fred (a turkey) is initially alive and a gun is initially unloaded. Loading the gun, waiting for a moment, and then shooting the gun at Fred is expected to kill Fred.
- However, in one solution, Fred indeed dies; in another (also logically correct) solution, the gun becomes mysteriously unloaded and Fred survives.
- By Hanks and McDermott, adapted from Wikipedia

# 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
(a) representation—avoid frame axioms
(b) 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):

$$\begin{array}{lcl} \text{P true afterwards} & \Leftrightarrow & [\text{an action made P true} \\ & \vee & \text{P true already and no action made P false}] \end{array}$$

For holding the gold:

$$\forall\, a, s \;\; Holding(Gold, Result(a,s)) \;\Leftrightarrow$$
$$[(a = Grab \wedge AtGold(s))$$
$$\vee\, (Holding(Gold, s) \wedge a \neq Release)]$$

# Set Theory in First-Order Logic

Can we define set theory using FOL?

- individual sets, union, intersection, etc

Answer is yes.

Basics:

- empty set = constant = { }

- unary predicate Set( ), true for sets

- binary predicates:

  $x \in s$ (true if x is a member of the set s)

  $s_1 \subseteq s_2$ (true if s1 is a subset of s2)

- binary functions:

  intersection $s_1 \cap s_2$, union $s_1 \cup s_2$, adjoining {x|s}

# A Possible Set of FOL Axioms for Set Theory

The only sets are the empty set and sets made by adjoining an element to a set

$\forall s \; Set(s) \Leftrightarrow (s = \{\} ) \vee (\exists x,s_2 \; Set(s_2) \wedge s = \{x|s_2\})$

The empty set has no elements adjoined to it

$\neg \exists x,s \; \{x|s\} = \{\}$

Adjoining an element already in the set has no effect

$\forall x,s \; x \in s \Leftrightarrow s = \{x|s\}$

The only elements of a set are those that were adjoined into it. Expressed recursively:

$\forall x,s \quad x \in s \Leftrightarrow [\; \exists y,s_2 \;\; (s = \{y|s_2\} \wedge (x = y \vee x \in s_2))]$

# A Possible Set of FOL Axioms for Set Theory

A set is a subset of another set iff all the first set's members are members of the 2nd set

$\forall s_1, s_2 \; s_1 \subseteq s_2 \Leftrightarrow (\forall x \; x \in s_1 \Rightarrow x \in s_2)$

Two sets are equal iff each is a subset of the other

$\forall s_1, s_2 \; (s_1 = s_2) \Leftrightarrow (s_1 \subseteq s_2 \wedge s_2 \subseteq s_1)$

An object is in the intersection of 2 sets only if a member of both

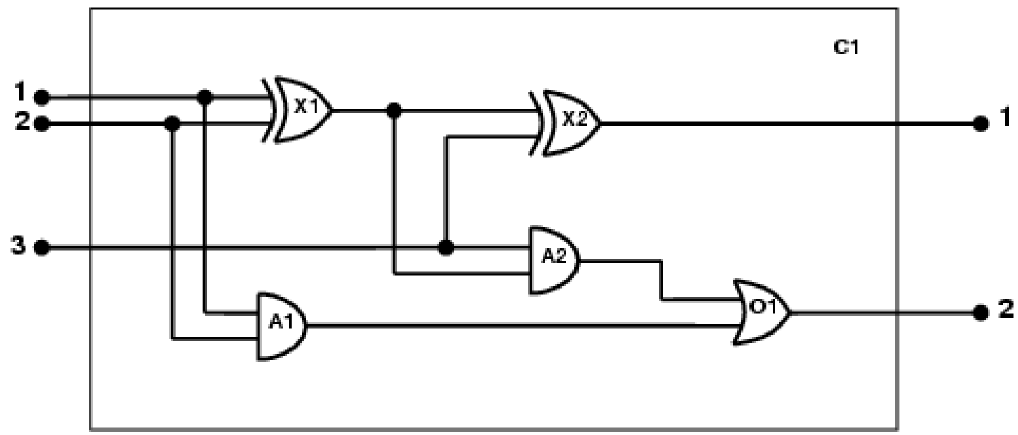
$\forall x, s_1, s_2 \; x \in (s_1 \cap s_2) \Leftrightarrow (x \in s_1 \wedge x \in s_2)$

An object is in the union of 2 sets only if a member of either

$\forall x, s_1, s_2 \; x \in (s_1 \cup s_2) \Leftrightarrow (x \in s_1 \vee x \in s_2)$

# The electronic circuits domain

One-bit full adder




Possible queries:

- does the circuit function properly?
- what gates are connected to the first input terminal?
- what would happen if one of the gates is broken?

and so on

# The electronic circuits domain

1. Identify the task
   – Does the circuit actually add properly?
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
   – Many alternative ways to say X1 is an OR gate:
   –
   – $Type(X_1) = XOR$ (function)
     $Type(X_1, XOR)$ (binary predicate)
     $XOR(X_1)$ (unary predicate)
     etc.

# The electronic circuits domain

4. Encode general knowledge of the domain

- $\forall t_1,t_2$ Connected$(t_1, t_2) \Rightarrow$ Signal$(t_1)$ = Signal$(t_2)$
- $\forall t$ Signal(t) = 1 $\vee$ Signal(t) = 0
- $1 \neq 0$
- $\forall t_1,t_2$ Connected$(t_1, t_2) \Rightarrow$ Connected$(t_2, t_1)$
- $\forall g$ Type(g) = OR $\Rightarrow$ Signal(Out(1,g)) = 1 $\Leftrightarrow \exists n$ Signal(In(n,g)) = 1
- $\forall g$ Type(g) = AND $\Rightarrow$ Signal(Out(1,g)) = 0 $\Leftrightarrow \exists n$ Signal(In(n,g)) = 0
- $\forall g$ Type(g) = XOR $\Rightarrow$ Signal(Out(1,g)) = 1 $\Leftrightarrow$ Signal(In(1,g)) $\neq$ Signal(In(2,g))
- $\forall g$ Type(g) = NOT $\Rightarrow$ Signal(Out(1,g)) $\neq$ Signal(In(1,g))

# The electronic circuits domain

5. Encode the specific problem instance

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

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

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

$Connected(In(1,C_1),In(1,X_1))$
$Connected(In(1,C_1),In(1,A_1))$
$Connected(In(2,C_1),In(2,X_1))$
$Connected(In(2,C_1),In(2,A_1))$
$Connected(In(3,C_1),In(2,X_2))$
$Connected(In(3,C_1),In(1,A_2))$

# The electronic circuits domain

6. Pose queries to the inference procedure:

What are the possible sets of values of all the terminals for the adder circuit?

$$\exists i_1,i_2,i_3,o_1,o_2\ Signal(In(1,C_1)) = i_1 \wedge Signal(In(2,C_1)) = i_2 \wedge Signal(In(3,C_1)) = i_3 \wedge Signal(Out(1,C_1)) = o_1 \wedge Signal(Out(2,C_1)) = o_2$$

7. Debug the knowledge base

May have omitted assertions like $1 \neq 0$

# Review --- 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

# Summary

- First-order logic:
  - Much more expressive than propositional logic
  - Allows objects and relations as semantic primitives
  - Universal and existential quantifiers
  - syntax: constants, functions, predicates, equality, quantifiers
- Knowledge engineering using FOL
  - Capturing domain knowledge in logical form