CS5100 Foundations of Artificial Intelligence

Module 04 Lesson 7

First-Order Logic

Overview

- First-Order Logic (FOL)
- Syntax and Semantics Of FOL
- Using FOL
- Knowledge Engineering in FOL
- Inference using FOL

Propositional logic
Pros

Propositional Logic

- Is declarative
- Allows partial, disjunctive and negated information to be represented (compare with programming languages and databases)
- Is compositional: meaning of a sentence = sum of meaning of parts
  - Meaning of $B_{1,1} \wedge P_{1,2}$ is derived from meaning of $B_{1,1}$ and of $P_{1,2}$
- Has context-independent meaning
  - Unlike natural language, where meaning depends on context

Propositional logic
Cons

Propositional logic has very limited expressive power

- E.g., How do we say ”It is breezy in squares adjacent to pits”
  - except by writing one such sentence for each square like:

$B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})$

$B_{2,1} \Leftrightarrow (P_{1,1} \lor P_{2,2} \lor P_{3,1})$

etc.

➔ First-Order Logic (FOL)

First-Order Logic

Propositional logic assumes the world contains facts that are *true* or *false*

First-Order Logic assumes the world contains

- Objects: people, houses, numbers, colors, baseball games, wars, …
- Relations: red, round, prime, brother of, bigger than, part of, comes between, …
  [like descriptions]
- Functions: father of, best friend, one more than, plus, …

BNF Grammar of First-Order Logic

Sentence ➔ Atomic Sentence | Complex Sentence

Items in black were in Propositional Logic
Items in blue in this slide are new, in FOL

Atomic Sentence ➔ Predicate | Predicate(Term,...) | Term = Term
Complex Sentence ➔ (Sentence) | ¬ Sentence |
Sentence ∧ Sentence | Sentence ∨ Sentence |
Sentence ⇒ Sentence | Sentence ⇔ Sentence |
Quantifier Variable, ... Sentence

Term ➔ Function(Term, ...) | Constant | Variable
Quantifier ➔ ∀ | ∃
Constant ➔ A | X_1 | Mary | ....
Variable ➔ a | x | y | ... [Note: Prolog uses different convention for constants & variables !]
Predicate ➔ True | False | P | Q | Snowing | ...
Function ➔ Parent() | Sister()| LeftEyebrow()

Operator Precedence: ¬, ∧ , ∨ , ⇒ , ⇔

Constants, Predicates, Functions

Constant symbols: objects in the world

- John, Mary, 5, Blue

Predicate symbols, which map to truth values

- greater(5,3) = true, sky(Blue) = true

Function symbols, which map individuals to individuals (like a ‘complicated’ name)

- father-of(Mary) = John
- color-of(Sky) = Blue
- Soccer parents: Georgia’s Dad: father-of(Georgia) = Rob

Predicates and functions defined by name and ‘arity’ (#args)
e.g. greater/2, color-of/1
Note: greater/3 different from greater/2 etc.

Terms, Sentences and Atoms

- A term (denoting a real-world object) is a constant symbol, a variable symbol, or a function of n terms.
  - x and $f(t_1, \ldots, t_n)$ are terms, where each t_i is a term.
    Function is not a subroutine.
  - A term with no variables is a ground term
- An atom (or atomic sentence) has value true or false
  - Is, in general, a predicate of n terms
- A complex sentence is formed from atoms with connectives,
  e.g. $\neg P$, $P \vee Q$, $P \wedge Q$, $P \Rightarrow Q$, $P \Leftrightarrow Q$ where P and Q are atoms

Quantifiers

Two kinds of quantifiers

- **Universal** $\forall$
- **Existential** $\exists$

Used to describe things without naming them

Universal quantification

$\forall x\ P(x)$: P holds **FOR-ALL values** of variable x in the domain associated with that variable; i.e., P is true for **every** object x

Everyone in the CS5100 course is smart

$\forall x$ **incourse**(x, CS5100) $\Rightarrow$ smart(x)

or $\forall x$ **inCourseCS5100**(x) $\Rightarrow$ smart(x)

All dolphins are mammals

$\forall x$ dolphin(x) $\Rightarrow$ mammal(x)

All humans are mortal

$\forall y$ human(y) $\Rightarrow$ mortal(y)

[By the way, does it matter if we use x or y or … here?]

Existential quantification

∃x P(x): P holds **for some value** of x in the domain associated with that variable;

that is, P is true for **at least one** object x (There exists an x such that P is true …)

Some one at NEU is smart ☺

$\exists x\ at(x,NEU) \land smart(x)$

Some birds cannot fly

$\exists x\ bird(x) \land cannotfly(x)$

Care with Quantifiers ..1

Universal quantifiers often used with $\Rightarrow$

e.g. $\forall x \;\; incourse(x, CS5100) \Rightarrow smart(x)$ [define a subgroup and say what's true about them]

- Universal quantification rarely used to make statements about every individual in the world:

e.g. $\forall x \;\; incourse(x, CS5100) \wedge smart(x)$

meaning "Everyone in the world
is in CS5100 and is smart"

Care with Quantifiers ..2

Existential quantifiers usually used with $\wedge$
to specify properties of an individual:

$\exists x\ at(x,NEU) \wedge smart(x)$
meaning: There is a NEU student who is smart

Common mistake, representing this as:

$\exists x\ at(x,NEU) \Rightarrow smart(x)$
meaning: If there is a student at NEU, s/he is smart

Nested Quantifiers ..1

Quantifiers can be nested

Switching the order of a number of universal quantifiers does not change the meaning:

$\forall x \forall y \; P(x,y) = \forall y \forall x \; P(x,y)$

$\forall x \forall y$ can also be written as $\forall x, y$

Similarly, you can switch the order of a number of existential quantifiers:

$\exists x \exists y \; P(x,y) = \exists y \exists x \; P(x,y)$

$\exists x \exists y$ can also be written as $\exists x, y$

Nested Quantifiers ..2

However: Switching the order of a mix of universals and existentials changes meaning:

$\forall x \exists y \text{ likes}(x,y) \quad = \quad \forall x (\exists y \text{ likes}(x,y))$

Everyone likes someone.

For everyone, there's at least one person they like.

$\exists y \forall x \text{ likes}(x,y)$

Someone is liked by everyone

There is (at least) one person (Mr. Bean? George Clooney? Dalai Lama?) that everyone likes.

Very different!

Connections between ∀ and ∃

Can relate sentences involving ∀ and ∃ using De Morgan's laws:
Similar to AND and OR.
Parentheses added just for clarity.

$$(\forall x)\ \neg P(x) \equiv \neg\ (\exists x)\ P(x)$$

$$\neg(\forall x)\ P(x) \equiv (\exists x)\ \neg P(x)$$

$$(\forall x)\ P(x) \equiv \neg\ (\exists x)\ \neg P(x)$$

$$(\exists x)\ P(x) \equiv \neg\ (\forall x)\ \neg P(x)$$

Equality

$term_1 = term_2$ is true under a given interpretation
if and only if $term_1$ and $term_2$ refer to the same object

E.g. definition of *Sibling* in terms of *Parent*:

$$
\begin{aligned}
&\forall x,y\ Sibling(x,y) \Leftrightarrow \\
&\qquad [\neg(x = y) \wedge \\
&\qquad \exists m,f\ \ \neg(m = f) \wedge \mathrm{Parent}(m,x) \wedge \mathrm{Parent}(f,x) \\
&\qquad\qquad \wedge \mathrm{Parent}(m,y) \wedge \mathrm{Parent}(f,y)\ ]
\end{aligned}
$$

Quick-checks .. 1

1. Every gardener likes rain.

   $\forall x$ gardener(x) $\Rightarrow$ likes(x,Rain)

2. Everyone likes icecream

   $\forall x$ likes(x,Icecream)

3. Everyone dislikes broccoli

   $\forall x$ $\neg$likes(x,Broccoli) [what about "No one likes broccoli."]

4. You can fool some of the people all of the time.

   $\exists$ x $\forall$t person(x) $\wedge$ time(t) $\Rightarrow$ can-fool(x,t)

5. You can fool all of the people some of the time.

   $\forall$x $\exists$ t person(x) $\wedge$ time(t) $\Rightarrow$ can-fool(x,t)

Quick-checks .. 2

6. All red mushrooms are poisonous.

   $\forall x\ \text{mushroom}(x) \land \text{red}(x) \Rightarrow \text{poisonous}(x)$

7. No purple mushroom is poisonous.

   $\neg(\exists\, x)\ (\text{purple}(x) \land \text{mushroom}(x) \land \text{poisonous}(x))$

   or, equivalently,

   $\forall x\ \text{mushroom}(x) \land \text{purple}(x) \Rightarrow \neg\text{poisonous}(x)$

8. There are exactly two pink mushrooms.

   $\exists x\ \exists y\ \text{mushroom}(x) \land \text{pink}(x) \land$
   $\text{mushroom}(y) \land \text{pink}(y) \land$
   $\neg(x=y) \land$
   $\forall z\ \text{mushroom}(z) \land \text{pink}(z) \Rightarrow ((z=x) \lor (z=y))$



Using FOL
Kinship

Brothers are siblings:

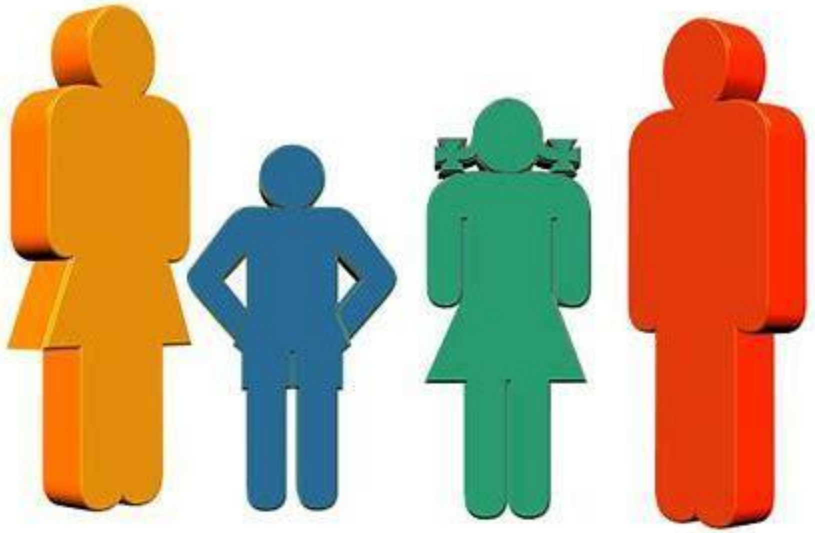
$\forall x,y\ Brother(x,y) \Rightarrow Sibling(x,y)$

One's mother is one's female parent:

$\forall m,c\ Mother(c) = m \Leftrightarrow (Female(m) \wedge Parent(m,c))$

"Sibling" is symmetric:

$\forall x,y\ Sibling(x,y) \Leftrightarrow Sibling(y,x)$



Using FOL
Sets

Sets are empty ones, and those got by adding to a set; empty set cannot be decomposed

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

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

Adding the same element has no effect

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

The only elements in a set are those adjoined to it:

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

Subsets

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

Two sets are equal if they are subsets of each other

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

Set Intersection

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

Set Union

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

Knowledge base for the Wumpus World

Perception

$\forall$t,s,b Percept([s,b,Glitter],t) $\Rightarrow$ Glitter(t)

s, b, Glitter are percepts

s,b = stench, breeziness

Reflex Action

$\forall$t Glitter(t) $\Rightarrow$ BestAction(Grab,t)

t = time t

…

Wumpus
Deducing hidden properties

Defining adjacent squares (using (x, y) coordinates):

$\forall x,y,a,b\ Adjacent([x,y],[a,b]) \Leftrightarrow$
$(x = a \ \wedge\ (y = b-1 \vee y = b+1)) \vee (y = b \ \wedge\ (x = a-1 \vee x = a+1))$

Properties of squares:

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

Where s is a square, t is a time

Squares are breezy near a pit:

Diagnostic rule---infer cause from effect

$\forall s\ Breezy(s) \Rightarrow \exists r\ Adjacent(r,s) \wedge Pit(r)$

Causal rule---infer effect from cause

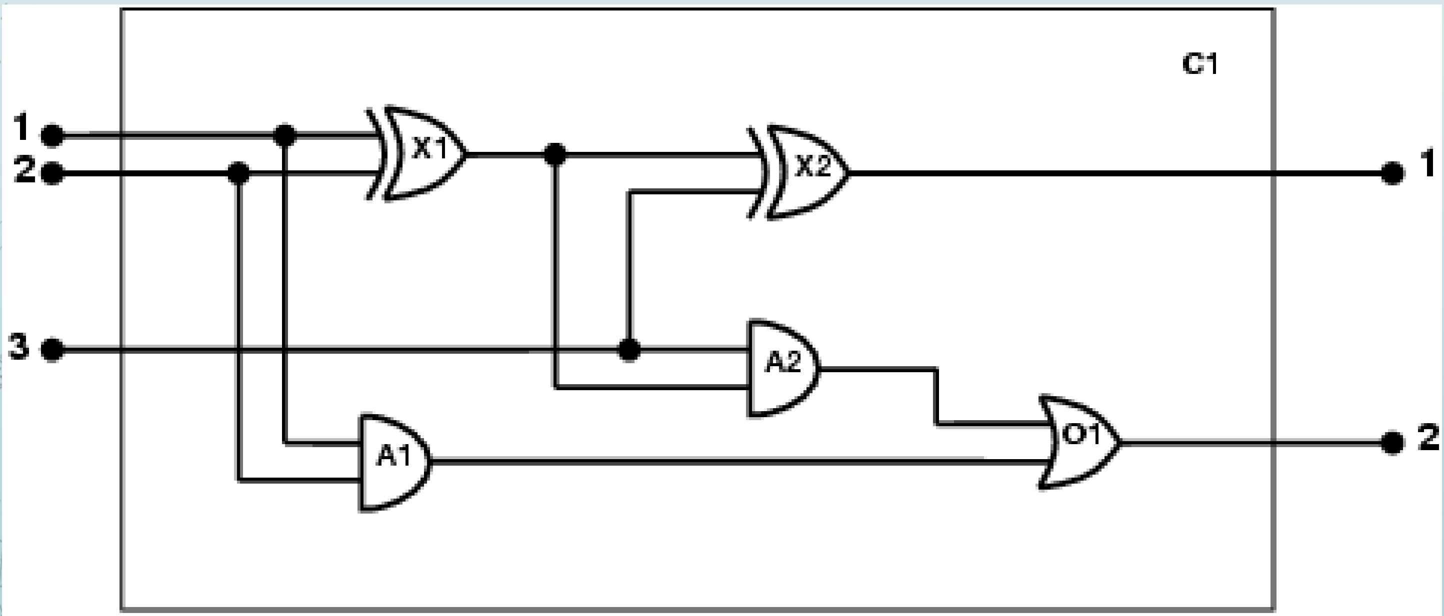
$\forall r\ Pit(r) \Rightarrow [\ \forall s\ Adjacent(r,s) \Rightarrow Breezy(s)\ ]$

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 ..1

One-bit full-adder C1



1-bit Full Adder, with 2 XOR gates X1, X2
2 AND gates A1, A2
1 OR gate O1

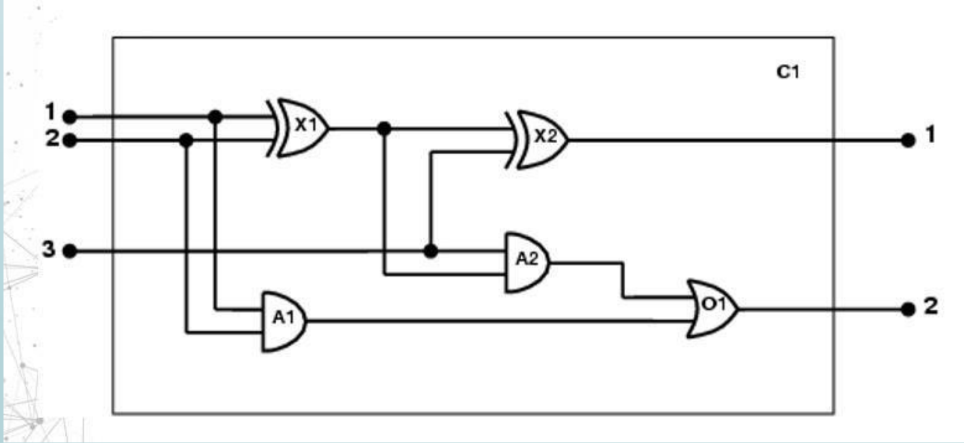
The electronic circuits domain ..2

1. Identify the task

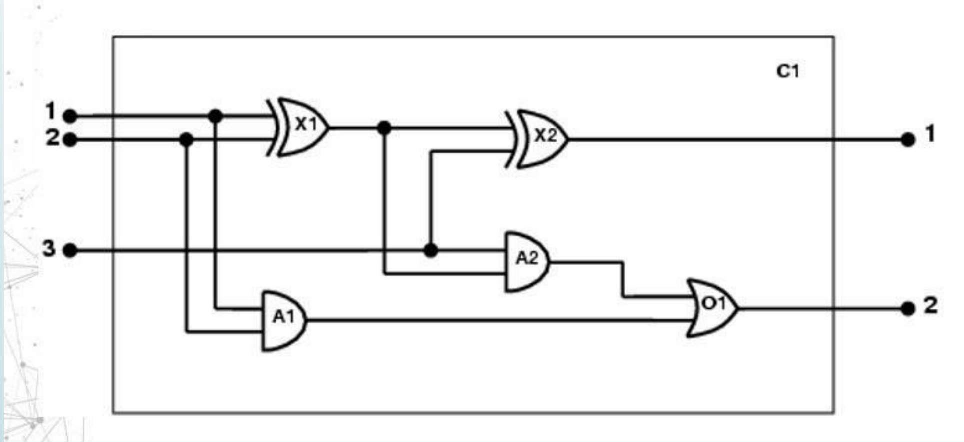
   Does the circuit actually add properly? (circuit verification)

2. Assemble the relevant knowledge
   Composed of wires and gates; Types of gates (AND, OR, XOR)
   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 ..3



4. Encode general knowledge of the domain
   1. $\forall t_1,t_2\ \text{Connected}(t_1, t_2) \Rightarrow \text{Signal}(t_1) = \text{Signal}(t_2)$
   2. $\forall t\ \text{Signal}(t) = 1 \vee \text{Signal}(t) = 0$
   3. $\forall t_1,t_2\ \text{Connected}(t_1, t_2) \Rightarrow \text{Connected}(t_2, t_1)$

   4. $\forall g\ \text{Type}(g) = \text{OR} \Rightarrow \text{Signal}(\text{Out}(1,g)) = 1 \Leftrightarrow \exists n\ \text{Signal}(\text{In}(n,g)) = 1$
   5. $\forall g\ \text{Type}(g) = \text{AND} \Rightarrow \text{Signal}(\text{Out}(1,g)) = 0 \Leftrightarrow \exists n\ \text{Signal}(\text{In}(n,g)) = 0$

   6. $\forall g\ \text{Type}(g) = \text{XOR} \Rightarrow \text{Signal}(\text{Out}(1,g)) = 1 \Leftrightarrow \text{Signal}(\text{In}(1,g)) \neq \text{Signal}(\text{In}(2,g))$
   7. $\forall g\ \text{Type}(g) = \text{NOT} \Rightarrow \text{Signal}(\text{Out}(1,g)) \neq \text{Signal}(\text{In}(1,g))$
   8. …

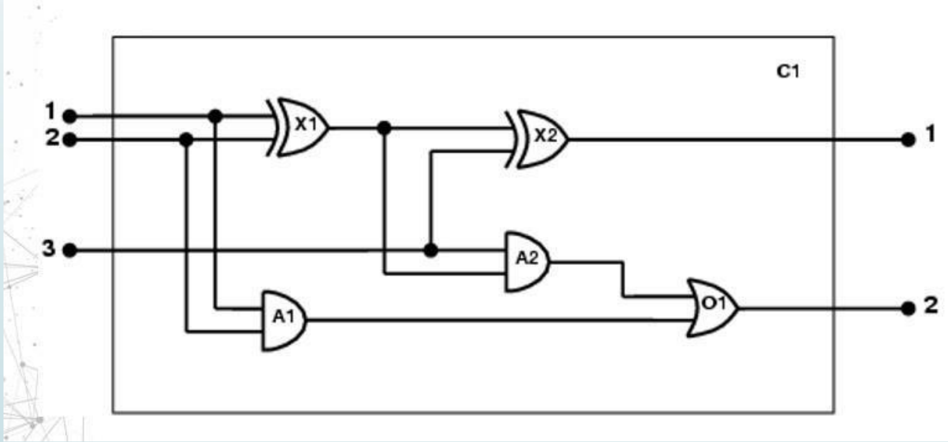
The electronic circuits domain ..4

5. Encode the specific problem instance

$Type(X_1) = XOR$ $Type(X_2) = XOR$

$Type(A_1) = AND$ $Type(A_2) = AND$

$Type(O_1) = OR$



$Connected(Out(1,X_1),In(1,X_2))$ $Connected(In(1,C_1),In(1,X_1))$

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

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

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

$Connected(Out(1,X_2),Out(1,C_1))$ $Connected(In(3,C_1),In(2,X_2))$

$Connected(Out(1,O_1),Out(2,C_1))$ $Connected(In(3,C_1),In(1,A_2))$

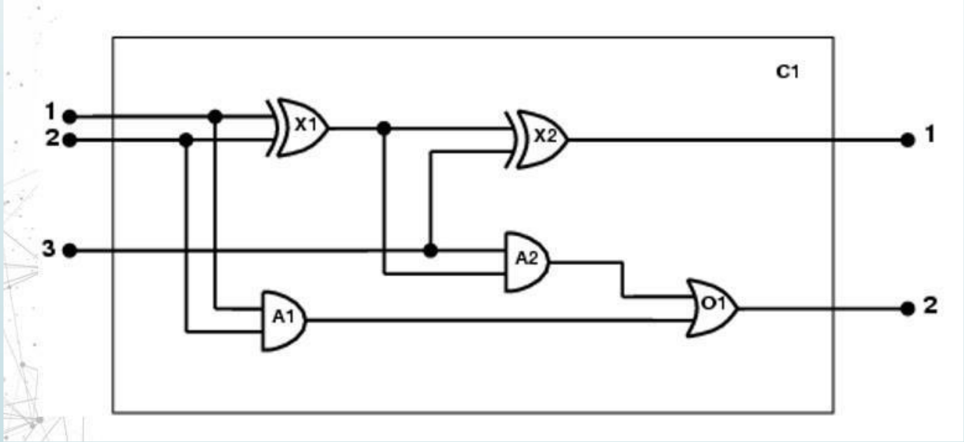
Key: $Connected(Out(1,X_1),In(1,X_2))$ = output 1 of X_1 is connected to input 1 of X_2

C_1 is the whole circuit

The electronic circuits domain ..5

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

FOL Summary

First-Order Logic:

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

Increased expressive power: sufficient to define wumpus world, sets, …

But how do you infer new info?

➔ Inference in FOL (drum roll!)

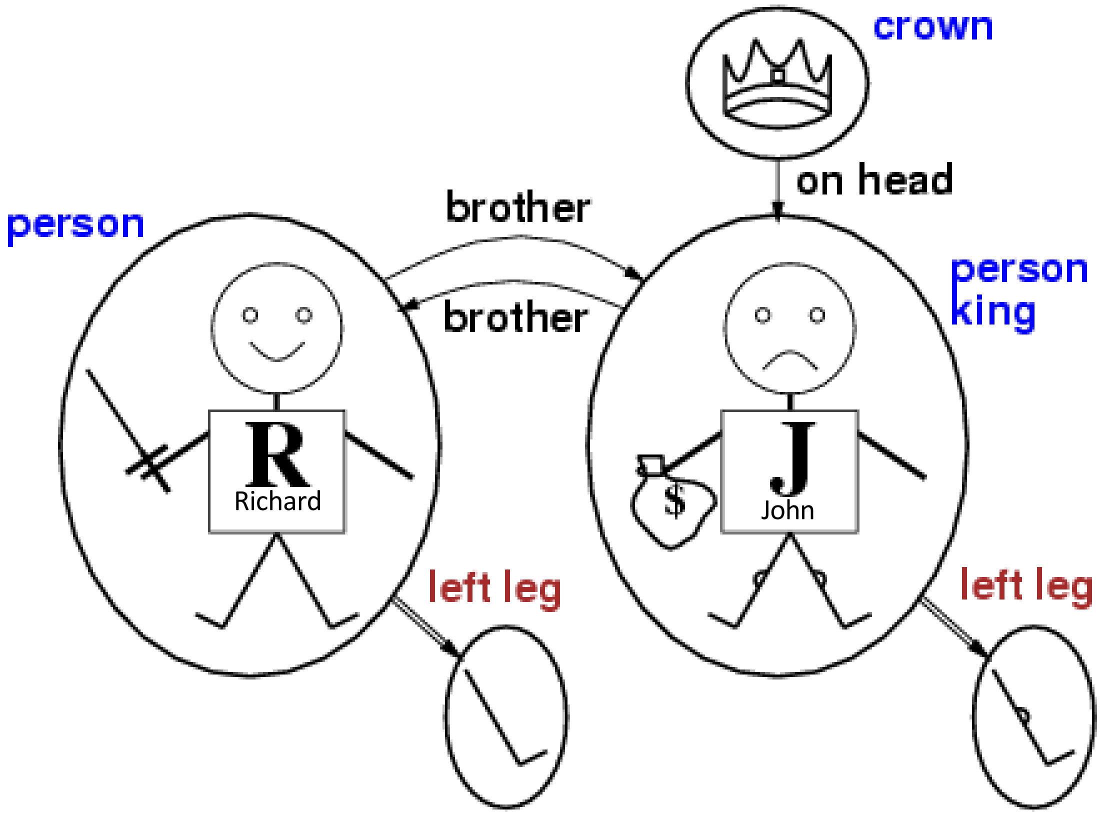
Inference in First-Order Logic

Inference in First-Order Logic

Outline

- Reducing First-Order inference to propositional inference
  - Dealing with quantified information
- Generalized Modus Ponens
- Unification
- Forward Chaining
- Backward Chaining
- Resolution

Models for FOL Example



Inference in First-Order Logic

Outline

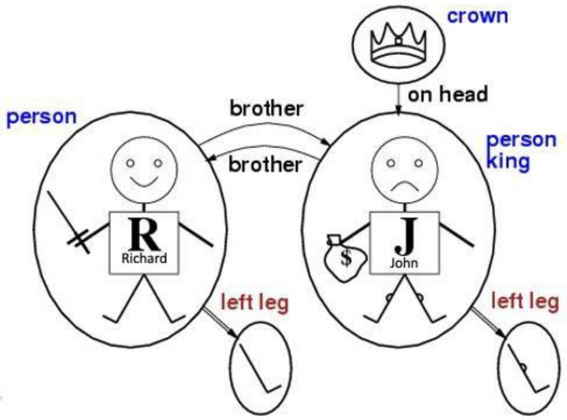
- Reducing First-Order inference to propositional inference
  - Dealing with quantified information
- Generalized Modus Ponens
- Unification
- Forward Chaining
- Backward Chaining
- Resolution

Universal Instantiation (UI)

- Every instantiation of a universally quantified sentence is entailed by it:

$$\frac{\forall v \;\; \alpha}{\mathrm{Subst}(\{v/g\}, \alpha)}$$

for any variable v and ground term g



- E.g., $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$ yields,

  for the three substitutions
  $\{x/John\}$, $\{x/Richard\}$ and $\{x/Father(John)\}$:

  $King(John) \wedge Greedy(John) \Rightarrow Evil(John)$

  The quantification goes away after this step.

  $King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$

  $King(Father(John)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John))$

Existential Instantiation (EI)

- For any sentence α, variable *v*, and constant symbol *k* that does not appear elsewhere in the knowledge base:

$$\frac{\exists v\ \alpha}{\text{Subst}(\{v/k\}, \alpha)}$$

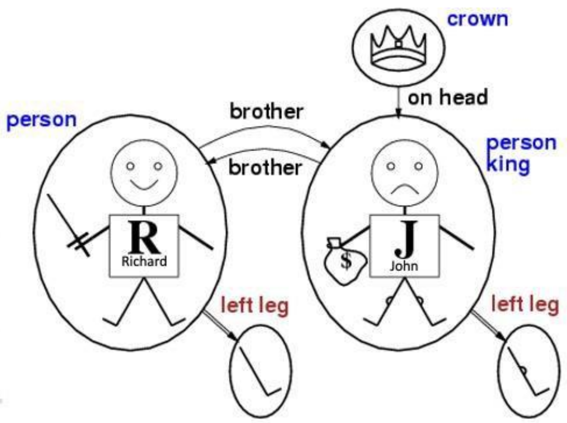
- E.g., $\exists x\ Crown(x) \wedge OnHead(x, John)$ yields:

$$Crown(C1) \wedge OnHead(C1, John)$$

provided *C1* is a new constant symbol, called a Skolem constant

The quantification goes away after this step.

Think of it as a kind of naming.



Reduction to propositional inference ..1

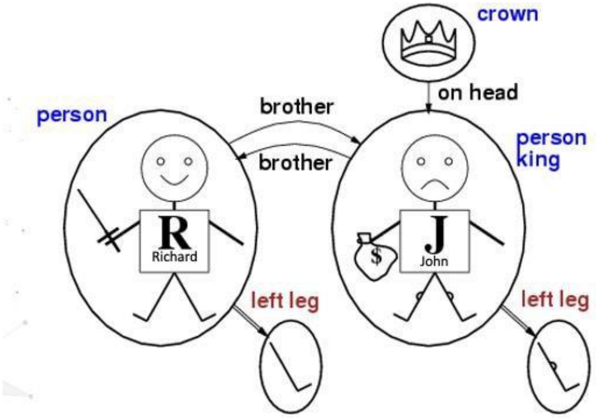
Suppose the KB contains just the following:

$\forall x$ King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x)

King(John)

Greedy(John)

Brother(Richard,John)



- Instantiating the universal sentence (with constants John, Richard), we have:

King(John) $\wedge$ Greedy(John) $\Rightarrow$ Evil(John)

King(Richard) $\wedge$ Greedy(Richard) $\Rightarrow$ Evil(Richard)

King(John)

Greedy(John)

Brother(Richard,John)

No more quantification!

The new KB is propositionalized: proposition symbols are King(John), Greedy(John), etc. Can infer e.g. Evil(John) from these.

Problems with propositionalization

- Propositionalization generates lots of irrelevant sentences, not very efficient

- E.g., from:

  $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$

  King(John)

  $\forall y\ Greedy(y)$

  Brother(Richard,John)

- it seems obvious that *Evil*(*John*), but propositionalization produces lots of facts such as *Greedy*(*Richard*) that are irrelevant

Inference in First-Order Logic
Outline

- Reducing First-Order inference to propositional inference
  - Dealing with quantified information
- Generalized Modus Ponens
- Unification
- Forward Chaining
- Backward Chaining
- Resolution

Generalized Modus Ponens (GMP)

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{\text{SUBST}(q,\theta)}$$

where $\text{SUBST}(p_i',\theta) = \text{SUBST}(p_i, \theta)$ for all i

King(*John*), *Greedy*(*y*) , (King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x))

p_1' is *King*(*John*)	p_1 is *King*(*x*)	θ is {x/John}
p_2' is *Greedy*(*y*)	p_2 is *Greedy*(*x*)	θ is {x/John,y/John}
q is *Evil*(*x*)		θ applied to q is *Evil*(*John*)

GMP used with KB of definite clauses (*exactly* one positive)

All variables assumed universally quantified ($\forall$x, y …)

Unification

Unification – takes two similar sentences and computes the substitution that makes them look the same

$$\text{Unify}(\alpha,\beta) = \theta \text{ if } \text{subs}(\alpha,\theta) = \text{subs}(\beta,\theta)$$

p	q	θ
Knows(Ray,x)	Knows(Ray,Jane)	{x/Jane}
Knows(Ray,x)	Knows(y,Geoff)	{x/Geoff,y/Ray}
Knows(Ray,x)	Knows(y,Mother(y))	{y/Ray,x/Mother(Ray)}
Knows(Ray,x)	Knows(x,Liz)	fail

The Unification Algorithm

function UNIFY(x, y, θ) **returns** a substitution to make x and y identical
  inputs: x, a variable, constant, list, or compound
    y, a variable, constant, list, or compound
    θ, the substitution built up so far

  if θ = failure **then return** failure
  else if $x = y$ **then return** θ
  else if VARIABLE?(x) **then return** UNIFY-VAR(x, y, θ)
  else if VARIABLE?(y) **then return** UNIFY-VAR(y, x, θ)
  else if COMPOUND?(x) **and** COMPOUND?(y) **then**
    return UNIFY(ARGS[x], ARGS[y], UNIFY(OP[x], OP[y], θ))
  else if LIST?(x) **and** LIST?(y) **then**
    return UNIFY(REST[x], REST[y], UNIFY(FIRST[x], FIRST[y], θ))
  else return failure

The Unification Algorithm

function UNIFY-VAR(var, x, θ) **returns** a substitution
  inputs: var, a variable
    x, any expression
    θ, the substitution built up so far

  if $\{var/val\} \in \theta$ **then return** UNIFY(val, x, θ)
  else if $\{x/val\} \in \theta$ **then return** UNIFY(var, val, θ)
  else if OCCUR-CHECK?(var, x) **then return** failure
  else return add $\{var/x\}$ to θ

Forward Chaining
Basic Idea

Similar to Forward Chaining on Propositions

Start with atomic sentences and keep making inferences till no more inferences can be made

FOL definite clauses are also disjunctions of literals, of which exactly one is positive

Definite clauses are either atomic, or

an implication with a conjunction of positive literals as antecedent, and a single positive literal as consequent $A \wedge B \ldots \Rightarrow Q$

E.g. $King(x) \wedge Greedy(x) \Rightarrow Evil(x)$

Variables are assumed to be universally quantified, and we omit quantifiers

Many FOL KBs can be converted into definite clauses.

Example knowledge base ..1

Consider:

- The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

- Prove that Colonel West is a criminal

Represent facts as First-Order definite clauses, then apply forward chaining

Example knowledge base ..2

... it is a crime for an American to sell weapons to hostile nations:

$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$

Nono … has some missiles, i.e., $\exists x\ Owns(Nono,x) \wedge Missile(x)$:

$Owns(Nono,M_1)$.

$Missile(M_1)$ *[using Existential Instantiation, Skolemization]*

… all of its missiles were sold to it by Colonel West

$Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$

Missiles are weapons:

$Missile(x) \Rightarrow Weapon(x)$

An enemy of America counts as "hostile“:

$Enemy(x,America) \Rightarrow Hostile(x)$

West, who is American …

$American(West)$

The country Nono, an enemy of America …

$Enemy(Nono,America)$

Forward chaining proof

... it is a crime for an American to sell weapons to hostile nations:

$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$

Nono ... has some missiles, i.e., $\exists x\ Owns(Nono,x) \wedge Missile(x)$:

$Owns(Nono,M_1)$.

$Missile(M_1)$ *[using Existential Instantiation, Skolemization]*

... all of its missiles were sold to it by Colonel West

$Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$

Missiles are weapons:

$Missile(x) \Rightarrow Weapon(x)$

An enemy of America counts as "hostile“:

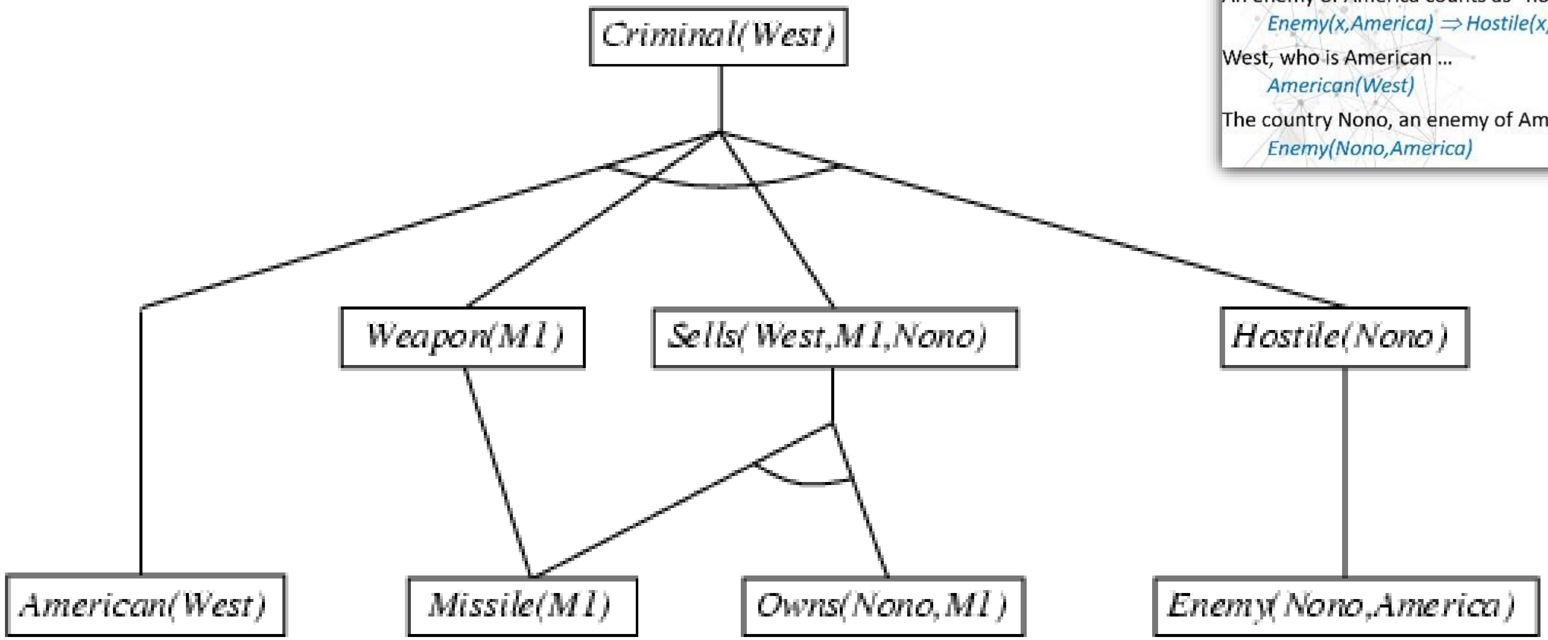
$Enemy(x,America) \Rightarrow Hostile(x)$

West, who is American ...

$American(West)$

The country Nono, an enemy of America ...

$Enemy(Nono,America)$



$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$

$Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$

$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x,America) \Rightarrow Hostile(x)$

Properties of forward chaining

Sound and complete for First-Order definite clauses

May not terminate in general if α is not entailed

- This is unavoidable: entailment with definite clauses is semi-decidable

Efficiency of forward chaining

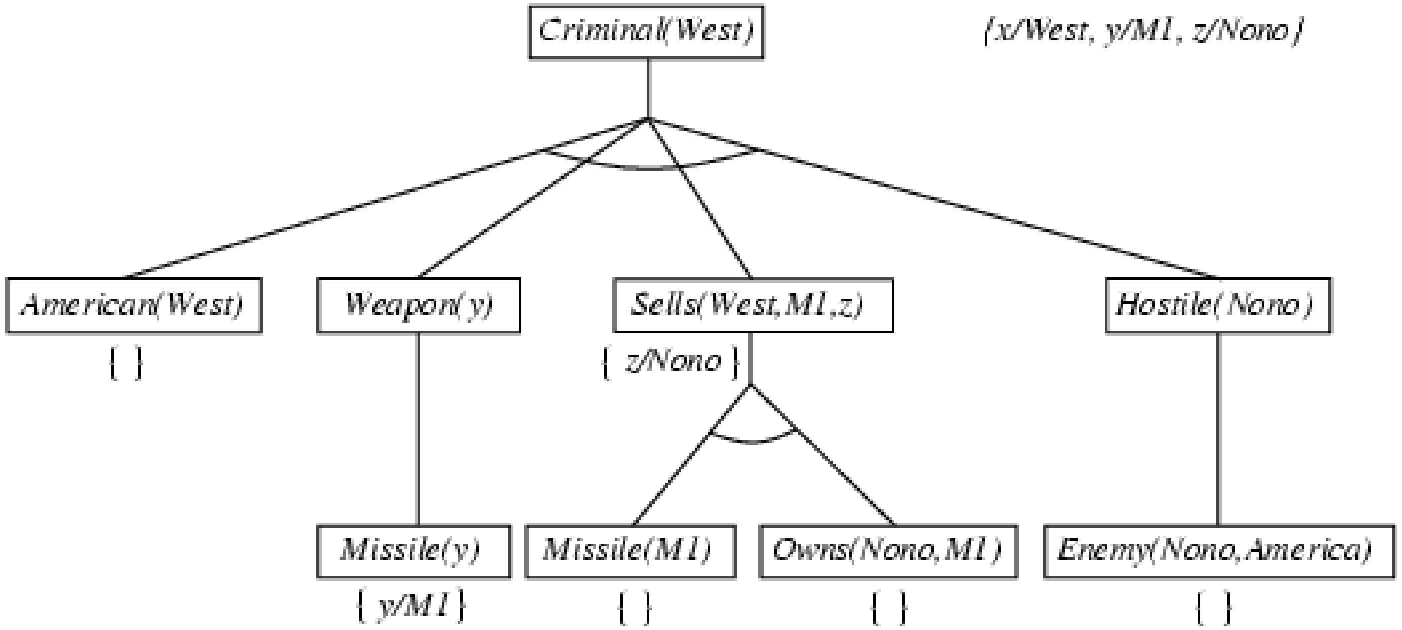
- Incremental forward chaining: no need to match a rule on iteration *k* if a premise wasn't added on iteration *k-1*
  - $\Rightarrow$ match each rule whose premise contains a newly added positive literal
- Matching itself can be expensive, but…
- Database indexing allows O(1) retrieval of known facts
  - e.g., query *Missile(x)* retrieves $Missile(M_1)$
- Forward chaining is widely used in deductive databases,
- Basis for production systems (rule-based systems)

Backward Chaining algorithm

- Work backwards from the goal

- Like for Propositional Logic, kind of AND/OR search
- Multiple substitutions possible, so implemented usually as a generator
  ➔ returns multiple times, each time giving a result

Backward Chaining example



Properties of backward chaining

- Depth-first recursive proof search: space is linear in size of proof
- Incomplete due to infinite loops
  - $\Rightarrow$ fix by checking current goal against every goal on stack
- Inefficient due to repeated subgoals (both success and failure)
  - $\Rightarrow$ fix using caching of previous results (extra space)
- Widely used for logic programming

PROLOG & Resolution

Logic programming: Prolog .. 1

- Algorithm = Logic + Control [Robert Kowalski]
- Basis: Definite clauses + bells & whistles

  Widely used in Europe, Japan (basis of 5th Generation project)

  Compilation techniques $\Rightarrow$ 60 million LIPS
- Program = set of clauses

  `head :- literal`$_1$`, … literal`$_n$`.`

```
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
```

[equivalent of *American(x)* $\wedge$ *Weapon(y)* $\wedge$ *Sells(x,y,z)* $\wedge$ *Hostile(z)* $\Rightarrow$ *Criminal(x)*

  uppercase for vars, lowercase for constants!

  read :- as 'if', ',' as AND, and note period at the end]

Logic programming: Prolog ..2

- Depth-first, left-to-right backward chaining;
  clause-order is important
- Built-in predicates for arithmetic etc.,
  e.g.,   X is Y*Z+3
- Built-in predicates that have side effects (e.g., input and output predicates, assert/retract predicates)
- **Closed-world assumption** ("negation as failure")
  - e.g., given alive(X) :- not dead(X).
  - alive(joe) succeeds if dead(joe) fails

Prolog Example

- Appending two lists to produce a third:

```
append([],Y,Y).
append([X|L],Y,[X|Z]) :- append(L,Y,Z).
```

- query: `?- append(A,B,[1,2])`

- answers:

```
A=[]     B=[1,2]
A=[1]    B=[2]
A=[1,2]  B=[]
```

Resolution
Brief summary

- Full First-Order version:

$$\frac{l_1 \vee \cdots \vee l_k, \qquad m_1 \vee \cdots \vee m_n}{(l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \quad \vee \quad m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\ \theta}$$

where $\mathrm{Unify}(l_i, \neg m_j) = \theta$.

- The two clauses are assumed to be standardized apart so that they share no variables.
- For example, $\dfrac{\neg Rich(x) \vee Unhappy(x), \quad Rich(Ken)}{Unhappy(Ken)}$

  with $\theta = \{x/Ken\}$
- Apply resolution steps to CNF(**KB** $\wedge \neg$**α**); complete for FOL

Conversion to CNF ..1

Everyone who loves all animals is loved by someone:

$\forall x\ [\forall y\ \mathit{Animal}(y) \Rightarrow \mathit{Loves}(x,y)] \Rightarrow [\exists y\ \mathit{Loves}(y,x)]$

1. Eliminate biconditionals and implications

$\forall x\ [\neg\forall y\ \neg\mathit{Animal}(y) \lor \mathit{Loves}(x,y)] \lor [\exists y\ \mathit{Loves}(y,x)]$

2. Move $\neg$ inwards: $\neg\forall x\ p \equiv \exists x\ \neg p$, $\neg\exists x\ p \equiv \forall x\ \neg p$

$\forall x\ [\exists y\ \neg(\neg\mathit{Animal}(y) \lor \mathit{Loves}(x,y))] \lor [\exists y\ \mathit{Loves}(y,x)]$

$\forall x\ [\exists y\ \neg\neg\mathit{Animal}(y) \land \neg\mathit{Loves}(x,y)] \lor [\exists y\ \mathit{Loves}(y,x)]$

$\forall x\ [\exists y\ \mathit{Animal}(y) \land \neg\mathit{Loves}(x,y)] \lor [\exists y\ \mathit{Loves}(y,x)]$

Conversion to CNF ..2

3. Standardize variables: each quantifier should use a different one

   $\forall x\ [\exists y\ Animal(y) \wedge \neg Loves(x,y)] \vee [\exists z\ Loves(z,x)]$

4. Skolemize: a more general form of existential instantiation.

   Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

   $\forall x\ [Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(z),x)$

5. Drop universal quantifiers:

   $[Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(z),x)$

6. Distribute $\vee$ over $\wedge$:

   $[Animal(F(x)) \vee Loves(G(z),x)] \wedge [\neg Loves(x,F(x)) \vee Loves(G(z),x)]$

Resolution proof definite clauses

... it is a crime for an American to sell weapons to hostile nations:

American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

Nono ... has some missiles, i.e., ∃x Owns(Nono,x) ∧ Missile(x):

Owns(Nono,M_1).

Missile(M_1) *[using Existential Instantiation, Skolemization]*

... all of its missiles were sold to it by Colonel West

Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

Missiles are weapons:

Missile(x) ⇒ Weapon(x)

An enemy of America counts as "hostile":

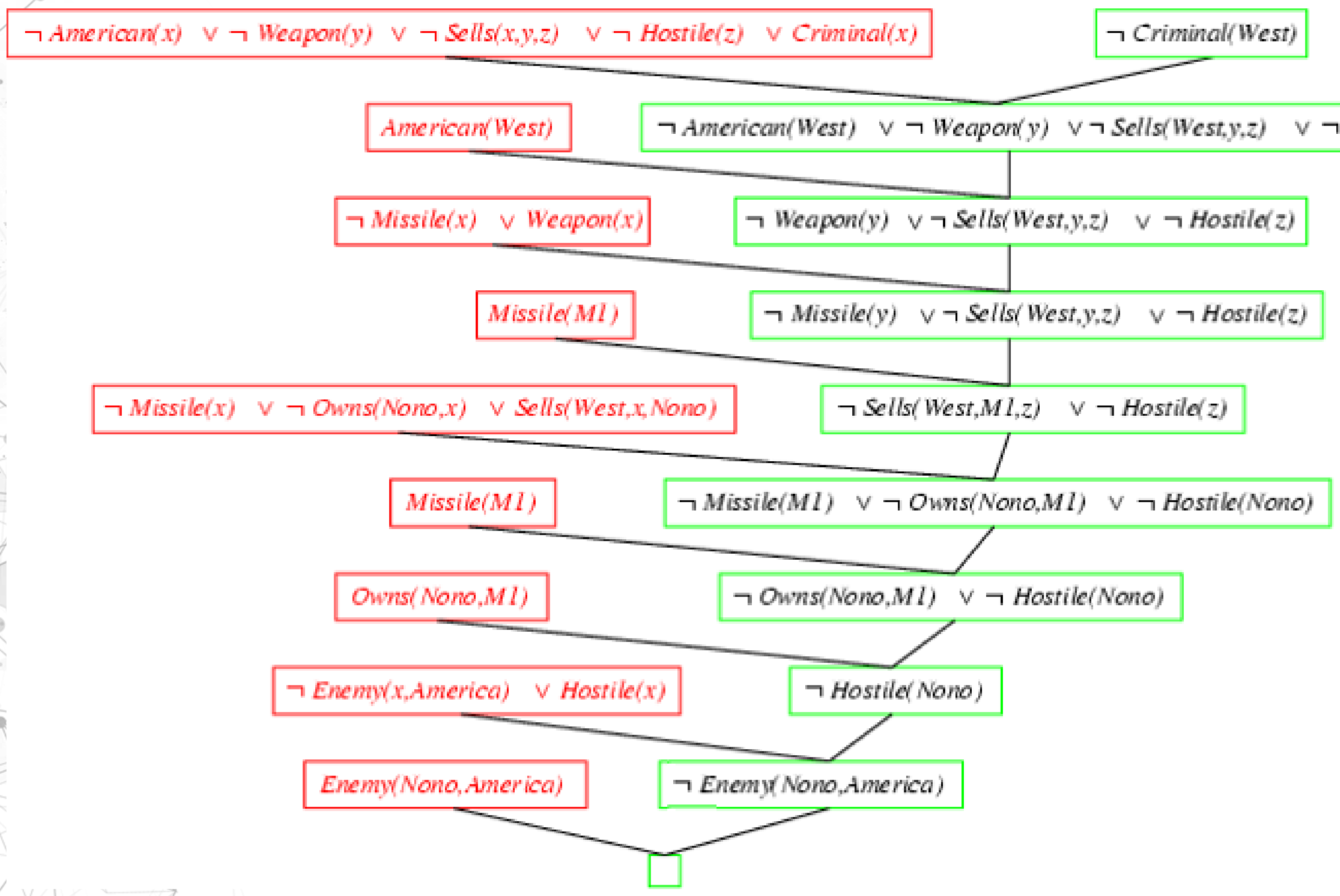
Enemy(x,America) ⇒ Hostile(x)

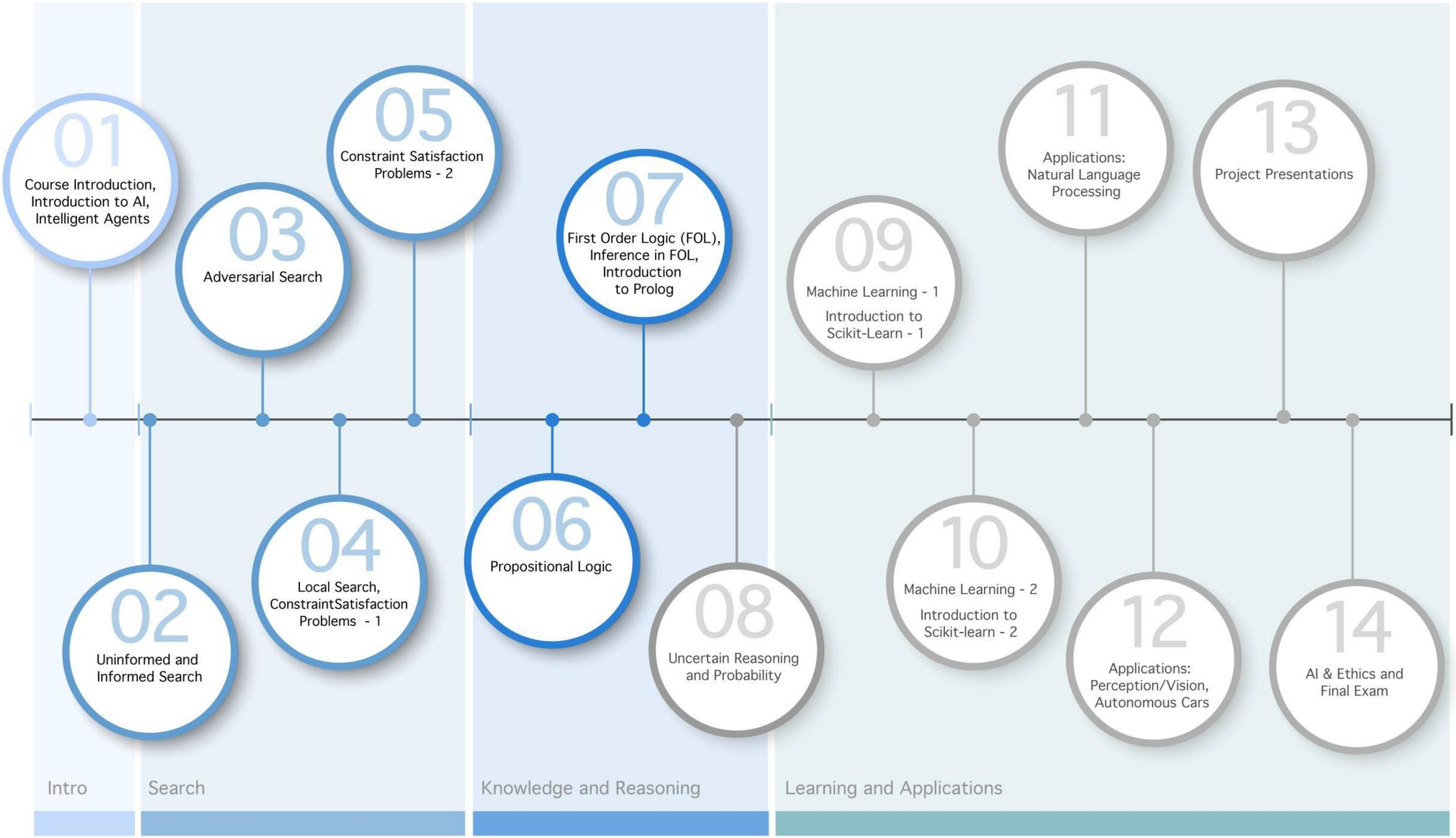
West, who is American ...

American(West)

The country Nono, an enemy of America ...

Enemy(Nono,America)




01
Course Introduction, Introduction to AI, Intelligent Agents
02
Uninformed and Informed Search
03
Adversarial Search
04
Local Search, ConstraintSatisfaction Problems - 1
05
Constraint Satisfaction Problems - 2
06
Propositional Logic
07
First Order Logic (FOL), Inference in FOL, Introduction to Prolog
08
Uncertain Reasoning and Probability
09
Machine Learning - 1
Introduction to Scikit-Learn - 1
10
Machine Learning - 2
Introduction to Scikit-learn - 2
11
Applications: Natural Language Processing
12
Applications: Perception/Vision, Autonomous Cars
13
Project Presentations
14
AI & Ethics and Final Exam
Intro
Search
Knowledge and Reasoning
Learning and Applications