# First-Order Logic
# Syntax

Reading: Chapter 8, ~~9.1-9.2, 9.5.1-9.5.5~~

FOL Syntax and Semantics read: 8.1-8.2

FOL Knowledge Engineering read: 8.3-8.5

~~FOL Inference read: Chapter 9.1-9.2, 9.5.1-9.5.5~~

(Please read lecture topic material before and after each lecture on that topic)

# Review: Resolution as Efficient Implication

(OR A B C D) ->Same -> (NOT (OR B C D)) => A
(OR ¬A E F G) ->Same -> A => (OR E F G)

----------------------------- ---------------------------------------------------

(OR B C D E F G) (NOT (OR B C D)) => (OR E F G)

---------------------------------------------------

(OR B C D E F G)

# Outline for First-Order Logic (FOL, also called FOPC)

- Propositional Logic is **Useful** --- but has **Limited Expressive Power**
- First Order Predicate Calculus (FOPC), or First Order Logic (FOL).
  - FOPC has greatly expanded expressive power, though still limited.
- New Ontology
  - The world consists of OBJECTS (for propositional logic, the world was facts).
  - OBJECTS have PROPERTIES and engage in RELATIONS and FUNCTIONS.
- New Syntax
  - Constants, Predicates, Functions, Properties, Quantifiers.
- New Semantics
  - Meaning of new syntax.
- Knowledge engineering in FOL
- ~~Inference in FOL~~

# FOL Syntax: You will be expected to know

- FOPC syntax
  - Syntax: Sentences, predicate symbols, function symbols, constant symbols, variables, quantifiers
- De Morgan's rules for quantifiers
  - connections between $\forall$ and $\exists$
- Nested quantifiers
  - Difference between "$\forall$ x $\exists$ y P(x, y)" and "$\exists$ x $\forall$ y P(x, y)"
  - $\forall$ x $\exists$ y Likes(x, y) --- "Everybody likes somebody."
  - $\exists$ x $\forall$ y Likes(x, y) --- "Somebody likes everybody."
- Translate simple English sentences to FOPC and back
  - $\forall$ x $\exists$ y Likes(x, y) $\Leftrightarrow$ "Everyone has someone that they like."
  - $\exists$ x $\forall$ y Likes(x, y) $\Leftrightarrow$ "There is someone who likes every person."

# Pros and cons of propositional logic

☺ Propositional logic is declarative

- Knowledge and inference are separate

☺ Propositional logic allows partial/disjunctive/negated information

- unlike most programming languages 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 **limited expressive power**

- E.g., cannot say "Pits cause breezes in adjacent squares."
  - except by writing one sentence for each square
- Needs to refer to objects in the world,
- Needs to express general rules

## First-Order Logic (FOL), also called First-Order Predicate Calculus (FOPC)

- 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, …
  - **Functions:** father of, best friend, one more than, plus, …
    - Function arguments are objects; function returns an object
  - **Objects generally correspond to English NOUNS**
  - **Predicates/Relations/Properties:** red, round, prime, brother of, bigger than, part of, comes between, …
    - Predicate arguments are objects; predicate returns a truth value
  - **Predicates generally correspond to English VERBS**
    - **First argument is generally the subject, the second the object**
    - Hit(Bill, Ball) usually means "Bill hit the ball."
    - Likes(Bill, IceCream) usually means "Bill likes IceCream."
    - Verb(Noun1, Noun2) usually means "Noun1 verb noun2."

# Aside: First-Order Logic (FOL) vs. Second-Order Logic

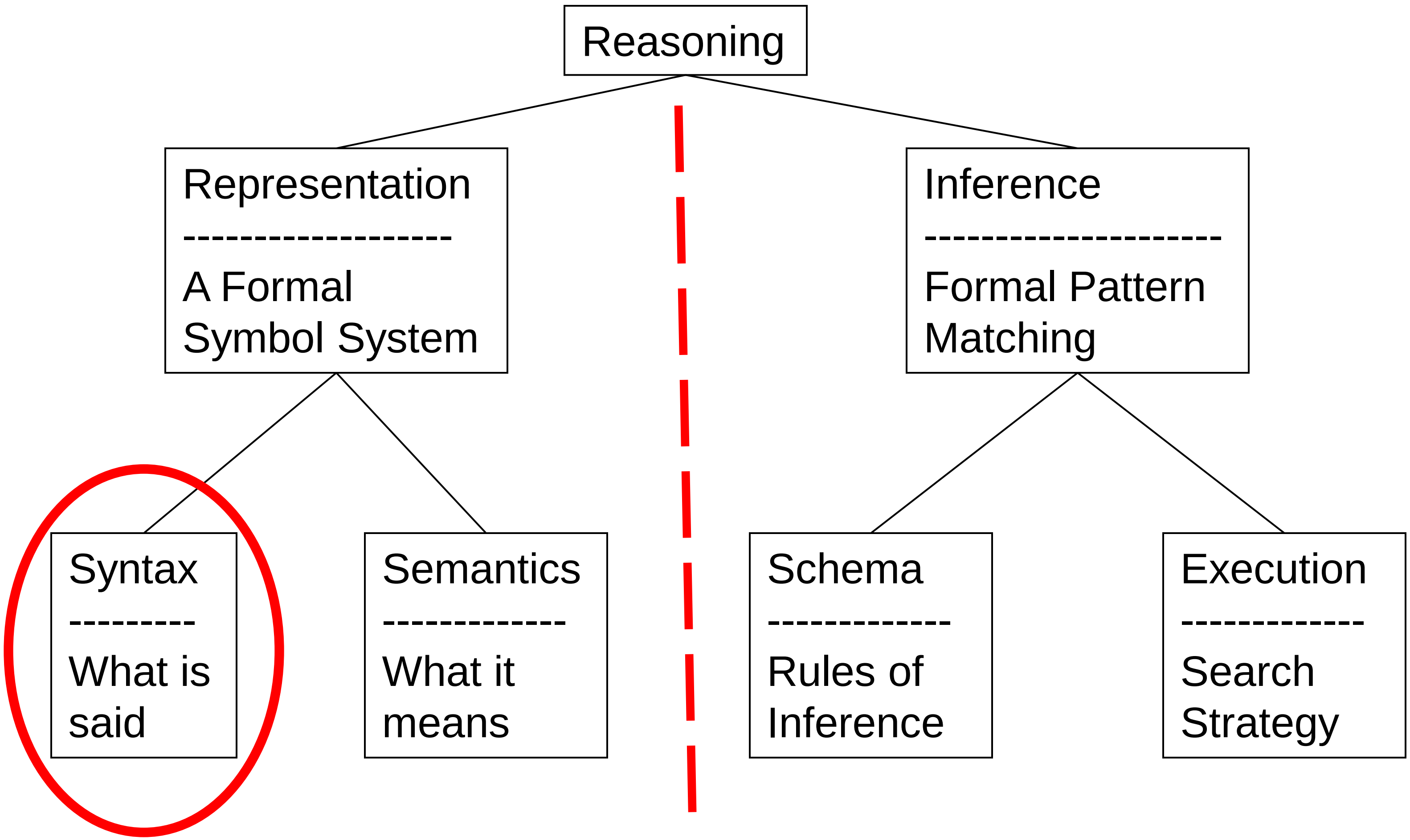
- First Order Logic (FOL) allows variables and general rules
  - "First order" because quantified variables represent objects.
  - "Predicate Calculus" because it quantifies over predicates on objects.
    - E.g., "Integral Calculus" quantifies over functions on numbers.
- Aside: Second Order logic
  - "Second order" because quantified variables can also represent predicates and functions.
    - E.g., can define "Transitive Relation," which is beyond FOPC.
- Aside: In FOL we can state that a relationship is transitive
  - E.g., BrotherOf is a transitive relationship
  - $\forall$ x, y, z BrotherOf(x,y) $\wedge$ BrotherOf(y,z) => BrotherOf(x,z)
- Aside: In Second Order logic we can define "Transitive"
  - $\forall$ P, x, y, z Transitive(P) $\Leftrightarrow$ ( P(x,y) $\wedge$ P(y,z) => P(x,z) )
  - Then we can state directly, Transitive(BrotherOf)

**FOL (or FOPC) Ontology:**

What kind of things exist in the world?

What do we need to describe and reason about?

Objects --- with their relations, functions, predicates, properties, and general rules.



## Syntax of FOL: Basic elements

- Constants KingJohn, 2, UCI,...
- Predicates Brother, >,...
- Functions Sqrt, LeftLegOf,...
- Variables x, y, a, b,...
- Quantifiers $\forall$, $\exists$
- Connectives $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$ (standard)
- Equality = (but causes difficulties....)

# Syntax of FOL: Basic syntax elements are symbols

- **Constant** Symbols (correspond to English nouns)
  - Stand for objects in the world.
    - E.g., KingJohn, 2, UCI, ...

- **Predicate** Symbols (correspond to English verbs)
  - Stand for relations (maps a tuple of objects to a **truth-value**)
    - E.g., Brother(Richard, John), greater_than(3,2), ...
  - P(x, y) is usually read as “x is P of y.”
    - E.g., Mother(Ann, Sue) is usually “Ann is Mother of Sue.”

- **Function** Symbols (correspond to English nouns)
  - Stand for functions (maps a tuple of objects to an **object**)
    - E.g., Sqrt(3), LeftLegOf(John), ...

- **Model** (world) = set of domain objects, relations, functions
- **Interpretation** maps symbols onto the model (world)
  - Very many interpretations are possible for each KB and world!
  - Job of the KB is to rule out models inconsistent with our knowledge.

## Syntax : Relations, Predicates, Properties, Functions

- Mathematically, all the Relations, Predicates, Properties, and Functions CAN BE represented simply as sets of *m*-tuples of objects:

- Let *W* be the set of objects in the world.

- Let $W^m$ = *W x W x ... (m times) ... x W*
  - The set of all possible *m*-tuples of objects from the world

- An ***m*-ary Relation** is a subset of $W^m$.
  - Example: Let *W* = *{John, Sue, Bill}*
  - Then $W^2$ = *{<John, John>, <John, Sue>, ..., <Sue, Sue>}*
  - E.g., *MarriedTo* = *{<John, Sue>, <Sue, John>}*
  - E.g., *FatherOf* = *{<John, Bill>}*

- Analogous to a constraint in CSPs
  - The constraint lists the *m*-tuples that satisfy it.
  - The relation lists the *m*-tuples that participate in it.

## Syntax : Relations, Predicates, Properties, Functions

- A **Predicate** is a list of *m*-tuples making the predicate true.
  - E.g., PrimeFactorOf = {<2,4>, <2,6>, <3,6>, <2,8>, <3,9>, ...}
  - This is the same as an *m*-ary Relation.
  - Predicates (and properties) generally correspond to English verbs.

- A **Property** lists the m-tuples that have the property.
  - Formally, it is a predicate that is true of tuples having that property.
  - E.g., IsRed = {<Ball-5>, <Toy-7>, <Car-11>, ...}
  - This is the same as an *m*-ary Relation.

- A **Function** CAN BE represented as an *m*-ary relation
  - the first *(m-1)* objects are the arguments and the $m^{th}$ is the value.
  - E.g., Square = *{<1, 1>, <2, 4>, <3, 9>, <4, 16>, ...}*

- An **Object** CAN BE represented as a function of zero arguments that returns the object.
  - This is just a 1-ary relationship.

## Syntax of FOL: Terms

- **Term** = logical expression that **refers to an object**

- **There are two kinds of terms:**

  – **Constant Symbols** stand for (or name) objects:
    - E.g., KingJohn, 2, UCI, Wumpus, ...

  – **Function Symbols** map tuples of objects to an object:
    - E.g., LeftLeg(KingJohn), Mother(Mary), Sqrt(x)
    - This is nothing but a complicated kind of name
      – No “subroutine” call, no “return value”

# Syntax of FOL: Atomic Sentences

- **Atomic Sentences** state facts (logical truth values).
  - An **atomic sentence** is a Predicate symbol, optionally followed by a parenthesized list of any argument terms
  - E.g., *Married( Father(Richard), Mother(John) )*
  - An **atomic sentence** asserts that some relationship (some predicate) holds among the objects that are its arguments.

- An **Atomic Sentence is true** in a given model if the relation referred to by the predicate symbol holds among the objects (terms) referred to by the arguments.

# Syntax of FOL: Atomic Sentences

- Atomic sentences in logic state facts that are true or false.

- Properties and *m*-ary relations do just that:

  LargerThan(2, 3) is false.

  BrotherOf(Mary, Pete) is false.

  Married(Father(Richard), Mother(John)) could be true or false.

  Properties and *m*-ary relations are Predicates that are true or false.

- Note: Functions refer to objects, do not state facts, and form no sentence:
  - Brother(Pete) refers to John (his brother) and is neither true nor false.
  - Plus(2, 3) refers to the number 5 and is neither true nor false.

- BrotherOf( Pete, Brother(Pete) ) is True.

Binary relation is a truth value.

Function refers to John, an object in the world, i.e., John is Pete's brother. (Works well iff John is Pete's only brother.)

## Syntax of FOL: Connectives & Complex Sentences

- **Complex Sentences** are formed in the same way, and are formed using the same logical connectives, as we already know from propositional logic

- The **Logical Connectives**:
  - $\Leftrightarrow$ biconditional
  - $\Rightarrow$ implication
  - $\wedge$ and
  - $\vee$ or
  - $\neg$ negation

- **Semantics** for these logical connectives are the same as we already know from propositional logic.

## Complex Sentences

- We make complex sentences with connectives (just like in propositional logic).

property

$$\neg Brother(LeftLeg(Richard), John) \vee (Democrat(Bush))$$

binary relation

function

objects

connectives

## Examples

- Brother(Richard, John) $\wedge$ Brother(John, Richard)
- King(Richard) $\vee$ King(John)
- King(John) => $\neg$ King(Richard)
- LessThan(Plus(1,2) ,4) $\wedge$ GreaterThan(1,2)

(Semantics of complex sentences are the same as in propositional logic)

## Syntax of FOL: Variables

- **Variables** range over objects in the world.

- A **variable** is like a **term** because it represents an object.

- A **variable** may be used wherever a **term** may be used.
  - **Variables** may be arguments to functions and predicates.

- (A **term with NO variables** is called a **ground term**.)
- (A **variable not bound by a quantifier** is called **free**.)

## Syntax of FOL: Logical Quantifiers

- There are two **Logical Quantifiers:**
  - **Universal:** $\forall$ x P(x) means “For all x, P(x).”
    - The “upside-down A” reminds you of “ALL.”
  - **Existential:** $\exists$ x P(x) means “There exists x such that, P(x).”
    - The “upside-down E” reminds you of “EXISTS.”

- Syntactic “sugar” --- we really only need one quantifier.
  - $\forall x\ P(x) \equiv \neg\exists x\ \neg P(x)$
  - $\exists x\ P(x) \equiv \neg\forall x\ \neg P(x)$
  - You can ALWAYS convert one quantifier to the other.

- **RULES:** $\forall \equiv \neg\exists\neg$ and $\exists \equiv \neg\forall\neg$

- **RULE:** To move negation “in” across a quantifier,
  change the quantifier to “the other quantifier”
  and negate the predicate on “the other side.”
  - $\neg\forall x\ P(x) \equiv \exists x\ \neg P(x)$
  - $\neg\exists x\ P(x) \equiv \forall x\ \neg P(x)$

# Universal Quantification $\forall$

- $\forall$ means “for all”

- Allows us to make statements about all objects that have certain properties

- Can now state general rules:

$\forall$ x King(x) => Person(x) “All kings are persons.”

$\forall$ x Person(x) => HasHead(x) “Every person has a head.”

$\forall$ i Integer(i) => Integer(plus(i,1)) “If i is an integer then i+1 is an integer.”

Note that
$\forall$ x King(x) $\wedge$ Person(x) is not correct!
This would imply that all objects x are Kings and are People

$\forall$ x King(x) => Person(x) is the correct way to say this

Note that => is the natural connective to use with $\forall$ .

# Universal Quantification $\forall$

- Universal quantification is equivalent to:
  - Conjunction of all sentences obtained by substitution of an object for the quantified variable.

- All Cats are Mammals.
  - $\forall x\ Cat(x) \Rightarrow Mammal(x)$

- Conjunction of all sentences obtained by substitution of an object for the quantified variable:

  $Cat(Spot) \Rightarrow Mammal(Spot) \wedge$

  $Cat(Rick) \Rightarrow Mammal(Rick) \wedge$

  $Cat(LAX) \Rightarrow Mammal(LAX) \wedge$

  $Cat(Shayama) \Rightarrow Mammal(Shayama) \wedge$

  $Cat(France) \Rightarrow Mammal(France) \wedge$

  $Cat(Felix) \Rightarrow Mammal(Felix) \wedge$

  *...*

# Existential Quantification $\exists$

- $\exists$ x means "there exists an x such that…." (at least one object x)
- Allows us to make statements about some object without naming it
- Examples:

$\exists$ x King(x) "Some object is a king."

$\exists$ x Lives_in(John, Castle(x)) "John lives in somebody's castle."

$\exists$ i Integer(i) $\wedge$ GreaterThan(i,0) "Some integer is greater than zero."

Note that $\wedge$ is the natural connective to use with $\exists$

(And note that => is the natural connective to use with $\forall$ )

# Existential Quantification ∃

- Existential quantification is equivalent to:
  - Disjunction of all sentences obtained by substitution of an object for the quantified variable.

- Spot has a sister who is a cat.
  - $\exists x$ Sister(x, Spot) $\wedge$ Cat(x)

- Disjunction of all sentences obtained by substitution of an object for the quantified variable:

  *Sister(Spot, Spot)* $\wedge$ *Cat(Spot)* $\vee$

  *Sister(Rick, Spot)* $\wedge$ *Cat(Rick)* $\vee$

  *Sister(LAX, Spot)* $\wedge$ *Cat(LAX)* $\vee$

  *Sister(Shayama, Spot)* $\wedge$ *Cat(Shayama)* $\vee$

  *Sister(France, Spot)* $\wedge$ *Cat(France)* $\vee$

  *Sister(Felix, Spot)* $\wedge$ *Cat(Felix)* $\vee$

  ***...***

# Combining Quantifiers --- Order (Scope)

The order of “unlike” quantifiers is important.

**Like nested variable scopes in a programming language**

**Like nested ANDs and ORs in a logical sentence**

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

- For everyone (“all x”) there is someone (“exists y”) whom they love.
- There might be a different y for each x (y is inside the scope of x)

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

- There is someone (“exists y”) whom everyone loves (“all x”).
- Every x loves the same y (x is inside the scope of y)

Clearer with parentheses: $\exists$ y ( $\forall$ x Loves(x,y) )

The order of “like” quantifiers does not matter.

**Like nested ANDs and ANDs in a logical sentence**

$$\forall x\ \forall y\ P(x, y) \equiv \forall y\ \forall x\ P(x, y)$$

$$\exists x\ \exists y\ P(x, y) \equiv \exists y\ \exists x\ P(x, y)$$

# Fun with sentences

Brothers are siblings

# Fun with sentences

Brothers are siblings

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

"Sibling" is symmetric

# Fun with sentences

Brothers are siblings

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

"Sibling" is symmetric

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

One's mother is one's female parent

# Fun with sentences

Brothers are siblings

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

"Sibling" is symmetric

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

One's mother is one's female parent

$$\forall x, y \;\; Mother(x, y) \;\Leftrightarrow\; (Female(x) \wedge Parent(x, y)).$$

A first cousin is a child of a parent's sibling

# Fun with sentences

Brothers are siblings

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

"Sibling" is symmetric

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

One's mother is one's female parent

$\forall x, y \;\; Mother(x, y) \;\Leftrightarrow\; (Female(x) \wedge Parent(x, y)).$

A first cousin is a child of a parent's sibling

$\forall x, y \;\; FirstCousin(x, y) \;\Leftrightarrow\; \exists\, p, ps \;\; Parent(p, x) \wedge Sibling(ps, p) \wedge Parent(ps, y)$

## Connections between Quantifiers

- Asserting that all x have property P is the same as asserting that does not exist any x that does not have the property P

$\forall x \;\; Likes(x, \text{CS-171 class}) \;\; \Leftrightarrow \;\; \neg \exists x \; \neg \; Likes(x, \text{CS-171 class})$

- Asserting that there exists an x with property P is the same as asserting that not all x do not have the property P

$\exists x \;\; Likes(x, IceCream) \;\; \Leftrightarrow \;\; \neg \forall x \; \neg \; Likes(x, IceCream)$

In effect:

- $\forall$ is a conjunction over the universe of objects
- $\exists$ is a disjunction over the universe of objects

Thus, DeMorgan's rules can be applied

# De Morgan's Law for Quantifiers

De Morgan's Rule

$$P \wedge Q \equiv \neg(\neg P \vee \neg Q)$$

$$P \vee Q \equiv \neg(\neg P \wedge \neg Q)$$

$$\neg(P \wedge Q) \equiv \neg P \vee \neg Q$$

$$\neg(P \vee Q) \equiv \neg P \wedge \neg Q$$

Generalized De Morgan's Rule

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

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

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

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

Rule is simple: if you bring a negation inside a disjunction or a conjunction, always switch between them (or →and, and → or).

## Aside: More syntactic sugar --- uniqueness

- $\exists!$ x is “syntactic sugar” for “There exists a unique x”
  - “There exists one and only one x”
  - “There exists exactly one x”
  - Sometimes $\exists!$ is written as $\exists^1$

- For example, E! x PresidentOfTheUSA(x)
  - “There is exactly one PresidentOfTheUSA.”

- This is just syntactic sugar:
  - $\exists!$ x P(x) is the same as $\exists$ x P(x) $\wedge$ ($\forall$ y P(y) => (x = y) )

# 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*:

$$\forall x,y\ Sibling(x,y) \Leftrightarrow$$
$$[\neg(x = y) \wedge$$
$$\exists m,f\ \neg (m = f) \wedge Parent(m,x) \wedge Parent(f,x)$$
$$\wedge\ Parent(m,y) \wedge\ Parent(f,y)]$$

**Equality can make reasoning much more difficult!**

**(See R&N, section 9.5.5, page 353)**

You may not know when two objects are equal.

E.g., Ancients did not know (MorningStar = EveningStar = Venus)

You may have to prove x = y before proceeding

E.g., a resolution prover may not know 2+1 is the same as 1+2

## 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.

## 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

## 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

- Nested quantifiers
  - Order of unlike quantifiers matters (the outer scopes the inner)
    - Like nested ANDs and ORs
  - Order of like quantifiers does not matter
    - like nested ANDS and ANDs

- Translate simple English sentences to FOPC and back