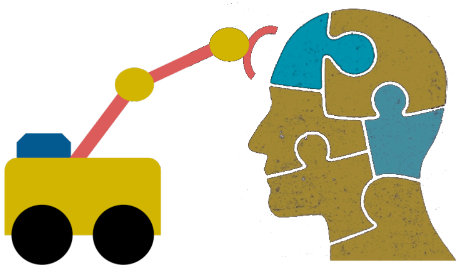
9.3.1



# First-Order Logic (FOL) part 1

# FOL Overview

- First Order logic (FOL) is a powerful knowledge representation (KR) system
- Used in AI systems in various ways, e.g., to
  - Directly represent & reason about concepts & objects
  - Formally specify meaning of KR systems (e.g., OWL)
  - For programming languages (e.g., Prolog) and rule-based systems
  - Make semantic database systems (Datalog) and Knowledge graphs (Wikidata)
  - Provide features useful in neural network deep learning systems

# First-order logic

- First-order logic (FOL) models the world in terms of
  - **Objects,** which are things with individual identities
  - **Properties** of objects that distinguish them from others
  - **Relations** that hold among sets of objects
  - **Functions,** a subset of relations where there is only one "value" for any given "input"
- Examples:
  - Objects: students, lectures, companies, cars ...
  - Relations: isa, hasBrother, biggerThan, outside, hasPart, color, occursAfter, owns, visits, precedes, ...
  - Properties: blue, oval, even, large, ...
  - Functions: hasFather, hasSSN, ...

# User provides

- **Constant symbols** representing individuals in world
  - BarackObama, Green, John, 3, “John Smith”
- **Predicate symbols** map individuals to truth values
  - greater(5,3)
  - green(Grass)
  - color(Grass, Green)
  - hasProperty(Grass, Color, Green)

How to represent properties and relations depends on our goals

- **Function symbols** map individuals to individuals
  - hasFather(SashaObama) = BarackObama
  - colorOf(Sky) = Blue

# What do these mean?

- We must indicate what these mean in ways humans will understand
  - i.e., map to their own internal representations
- May be done via a combination of
  - Choosing good names for formal terms, e.g., calling a concept HumanBeing instead of Q5
  - Comments in the definition `#human being`
  - Descriptions and examples in documentation
  - Reference to other representations , e.g., sameAs Q5 in Wikidata and Person in schema.org
  - Give examples like *Donald Trump* and *Luke Skywalker* to help distinguish concepts of real and fictional person

# FOL Provides

- **Variable symbols**
  - e.g., X, Y, ?foo, ?number
- **Connectives**
  - Same as propositional logic: not ($\neg$), and ($\wedge$), or ($\vee$), implies ($\rightarrow$), iff ($\leftrightarrow$), equivalence ($\equiv$), …
- **Quantifiers**
  - Universal $\forall$**x** or **(Ax)**
  - Existential $\exists$**x** or **(Ex)**

# Sentences: built from terms and atoms

- **term** (denoting an individual): constant or variable symbol, or n-place function of n terms, e.g.:
  - Constants: john, umbc
  - Variables: X, Y, Z
  - Functions: mother_of(john), phone(mother(x))
- **Ground terms** have no variables in them
  - **Ground:** john, father_of(father_of(john))
  - **Not Ground:** father_of(X)
- Syntax varies, e.g., variables start with a "?" or a capital letter, or are identified by quantifiers

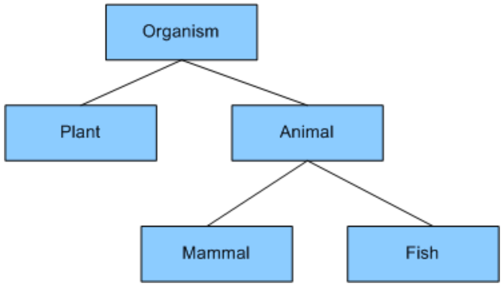
# Sentences: built from terms and atoms

- **atomic sentences** (which are either true or false) are n-place predicates of n terms, e.g.:
  - green(kermit)
  - between(philadelphia, baltimore, dc)
  - loves(X, mother(X))
- **complex sentences** formed from atomic ones connected by the standard logical connectives with quantifiers if there are variables, e.g.:
  - loves(mary, john) $\vee$ loves(mary, bill)
  - $\forall$x loves(mary, x)

# What do atomic sentences mean?

- Unary predicates typically encode a **type**
  - muppet(Kermit): kermit is a kind of muppet
  - green(kermit): kermit is a kind of green thing
  - integer(X): x is a kind of integer
- Non-unary predicates typically encode **relations** or **properties**
  - Loves(john, mary)
  - Greater_than(2, 1)
  - Between(newYork, philadelphia, baltimore)
  - hasName(john, "John Smith")

# Ontology




- Designing a logic representation is like designing a model in an object-oriented language
- **Ontology:** a "formal naming and definition of the types, properties, and relations of entities for a domain of discourse"
- E.g.: schema.org ontology used to put semantic data on Web pages to help search engines
  - Here's the semantic markup Google sees on our site
  - It's encoded as JSON, but the model is logic

# Sentences: built from terms and atoms

- **quantified sentences** adds quantifiers $\forall$ and $\exists$

  $\forall$x loves(x, mother(x))

  $\exists$x number(x) $\wedge$ greater(x, 100), prime(x)

- **well-formed formula (wff):** a sentence with no *free* variables or where all variables are *bound* by a universal or existential *quantifier*

  In **($\forall$x)P(x, y)** x is bound & y is free so it's not a wff

# Quantifiers: ∀ and ∃

- **Universal quantification**
  - $(\forall x)P(X)$ means P holds for **all** values of X in the domain associated with variable[1]
  - E.g., $(\forall X)$ dolphin(X) $\rightarrow$ mammal(X)
- **Existential quantification**
  - $(\exists x)P(X)$ means P holds for **some** value of X in domain associated with variable
  - E.g., ($\exists$**X**) mammal(X) $\wedge$ lays_eggs(X)
  - This lets us make statements about an object without identifying it

[1] a variable's domain is often not explicitly stated and is assumed by the context

# Universal Quantifier: $\forall$

- **Universal quantifiers typically used with *implies* to form *rules*:**

  *Logic:* $\forall X\ student(X) \rightarrow smart(X)$

  Means: All students are smart

- **Universal quantification *rarely* used without implies:**

  *Logic:* $\forall X\ student(X) \wedge smart(X)$

  Means: Everything is a student and is smart

- **What about this, though:**
  - *Logic:* $\forall X\ alive(X) \vee dead(X)$
  - Means: everything is either alive or dead

# Universal Quantifier: $\forall$

- **What about this, though:**
  - *Logic:* $\forall X\ alive(X) \vee dead(X)$
  - Means: everything is either alive or dead
- **Can be rewritten using a standard tautology**
  - $A \vee B \equiv \sim A \rightarrow B$
- **Giving both of these** (since $A \vee B \equiv B \vee A$)
  - $\forall X \sim alive(X) \rightarrow dead(X)$
  - $\forall X\ alive(X) \rightarrow \sim dead(X)$

# Existential Quantifier: $\exists$

- Existential quantifiers usually used with **and** to specify a list of properties about an individual

  *Logic:* $(\exists X)\ student(X) \wedge smart(X)$

  *Meaning:* There is a student who is smart

- Common mistake: represent this in FOL as:

  *Logic:* $(\exists X)\ student(X) \rightarrow smart(X)$

  *Meaning: ?*

# Existential Quantifier: ∃

- Existential quantifiers usually used with **and** to specify a list of properties about an individual

  *Logic:* $(\exists X)\ student(X) \wedge smart(X)$

  *Meaning:* There is a student who is smart

- Common mistake: represent this in FOL as:

  *Logic:* $(\exists X)\ student(X) \rightarrow smart(X)$

  $P \rightarrow Q = \sim P \vee Q$

  $\exists X\ student(X) \rightarrow smart(X) = \exists X \sim student(X) \vee smart(X)$

  *Meaning: There's something that is either not a student or is smart*

# Quantifier Scope

- FOL sentences have structure, like programs
- In particular, variables in a sentence have a **scope**
- Suppose we want to say "everyone who is alive loves someone"

  $(\forall X)\ alive(X) \rightarrow (\exists\ Y)\ loves(X, Y)$

- Here's how we scope the variables

$$(\forall X)\ alive(X) \rightarrow (\exists Y)\ loves(X, Y)$$

Scope of x
Scope of y

# Quantifier Scope

- **Switching order of two universal quantifiers *does not* change the meaning**
  - $(\forall X)(\forall Y)P(X,Y) \leftrightarrow (\forall Y)(\forall X)\ P(X,Y)$
  - Dogs hate cats (i.e., all dogs hate all cats)
- **You can switch order of existential quantifiers**
  - $(\exists X)(\exists Y)P(X,Y) \leftrightarrow (\exists Y)(\exists X)\ P(X,Y)$
  - A cat killed a dog
- **Switching order of universal and existential quantifiers *does* change meaning:**
  - Everyone likes someone: $(\forall X)(\exists Y)$ likes(X,Y)
  - Someone is liked by everyone: $(\exists Y)(\forall X)$ likes(X,Y)

```
def verify1():
    # Everyone likes someone: (∀x)(∃y) likes(x,y)
    for p1 in people():
        foundLike = False
        for p2 in people():
            if likes(p1, p2):
                foundLike = True
                break
        if not foundLike:
            print(p1, ‘does not like anyone ☹’)
            return False
    return True
```

*Every person has at least one individual that they like.*

# Procedural example 1

```
def verify2():
    # Someone is liked by everyone: (∃y)(∀x) likes(x,y)
    for p2 in people():
        foundHater = False
        for p1 in people():
            if not likes(p1, p2):
                foundHater = True
                break
        if not foundHater
            print(p2, 'is liked by everyone ☺')
            return True
    return False
```

*There is a person who is liked by every person in the universe.*

# Procedural example 2

# Connections between ∀ and ∃

- We can relate sentences involving ∀ and ∃ using extensions to **De Morgan's laws**:
  1. $(\forall x)\ P(x) \leftrightarrow \neg(\exists x)\ \neg\ P(x)$
  2. $\neg(\forall x)\ P(x) \leftrightarrow (\exists x)\ \neg P(x)$
  3. $(\exists\ x)\ P(x) \leftrightarrow \neg\ (\forall\ x)\ \neg P(x)$
  4. $(\exists x)\ P(x) \leftrightarrow \neg(\forall x)\ \neg P(x)$
- Examples
  1. All dogs don't like cats ↔ No dog likes cats
  2. Not all dogs bark ↔ There is a dog that doesn't bark
  3. All dogs sleep ↔ There is no dog that doesn't sleep
  4. There is a dog that talks ↔ Not all dogs can't talk

# Notational differences

- **Different symbols** for *and, or, not, implies, ...*
  - $\forall \;\; \exists \;\; \Rightarrow \;\; \Leftrightarrow \;\; \wedge \;\; \vee \;\; \neg \;\; \bullet \;\; \supset$
  - p v (q ^ r)
  - p + (q * r)

- **Different syntax** for variables vs. constants, predicates vs. functions, etc.

# Notational differences

- **Typical logic notation**

  $\forall x\ \exists y\ furry(x) \wedge meows(x) \wedge has(x, y), claw(y) \Rightarrow cat(x)$

- **Prolog**

  ```
  cat(X) :- furry(X), meows (X), has(X, Y), claw(Y).
  ```

- **Lisp notations**

  ```
  (forall ?x (implies (and (furry ?x) (meows ?x)
                           (has ?x ?y) (claw ?y))
                      (cat ?x)))
  ```

- **Python code**

  e.g., AIMA python, [logic.ipynb]

- **Knowledge graph triples**

  e.g., in RDF/OWL

```
#Graph from http://www.doefamily.com/
@prefix doefamily: <http://www.doefamily.com/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix dbpedia: <http://dbpedia.org/resource/> .

doe:John rdf:type foaf:Person .
doe:John owl:sameAs dbpedia:John_Doe .
doe:John foaf:age 72 .
doe:John foaf:mbox <mailto:john@doe.com> .

doe:Jane rdf:type foaf:Person .
doe:Jane owl:sameAs dbpedia:Jane_Doe .
doe:Jane foaf:age 12 .
doe:Jane foaf:phone “512-475-6656” .
```

*Fin*