CMSC 471 Propositional and First-Order Logic

KMA Solaiman <u>ksolaima@umbc.edu</u>

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: brother-of, bigger-than, outside, part-of, hascolor, occurs-after, owns, visits, precedes, ...
 - Properties: blue, oval, even, large, ...
 - Functions: father-of, best-friend, more-than ...

Quantifiers: \forall and \exists

- Universal quantification
 - (∀x)P(X) means P holds for all values of X in the domain associated with variable¹
 - E.g., $(\forall X)$ dolphin(X) \rightarrow mammal(X)
- Existential quantification
 - (∃x)P(X) means P holds for some value of X in domain associated with variable
 - E.g., $(\exists X)$ mammal(X) \land 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: (∀X) student(X) → smart(X)
 Means: All students are smart
- Universal quantification *rarely* used without implies: Logic: (∀X) student(X) ∧ smart(X)

Means: Everything is a student and is smart

Existential Quantifier: ∃

- Existential quantifiers usually used with and to specify a list of properties about an individual Logic: (∃X) student(X) ∧ smart(X) Meaning: There is a student who is smart
- Common mistake: represent this in FOL as: Logic: (∃X) student(X) → smart(X) Meaning: ?

Existential Quantifier: ∃

- Existential quantifiers usually used with and to specify a list of properties about an individual Logic: (∃X) student(X) ∧ 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)

 $P \rightarrow Q = {}^{\sim}P \lor Q$

 $\exists X \ student(X) \rightarrow smart(X) = \exists X \ \ \ student(X) \ v \ smart(X)$ 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"

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

• Here's how we scope the variables

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

Quantifier Scope

- Switching order of 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)

Procedural example 1 (Illustrative only!)

def verify1(): # Everyone likes someone: $(\forall x)(\exists y)$ likes(x,y)for p1 in people(): foundLike = False Every person has at for p2 in people(): least one individual that if likes(p1, p2): they like. foundLike = True break if not foundLike: print(p1, 'does not like anyone \mathfrak{S}') return False return True

Procedural example 2 (Illustrative only!) def verify2(): # Someone is liked by everyone: $(\exists y)(\forall x)$ likes(x,y)for p2 in people(): foundHater = False for p1 in people(): There is a person who is if not likes(p1, p2): liked by every person in foundHater = True the universe. break if not foundHater print(p2, 'is liked by everyone \odot ') return True return False

Connections between \forall and \exists

 We can relate sentences involving ∀ and ∃ using extensions to <u>De Morgan's laws</u>:

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. $\neg (\exists x) P(x) \leftrightarrow (\forall x) \neg P(x)$

• 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, ...
 - $\bullet \forall \exists \Rightarrow \Leftrightarrow \land \lor \neg \bullet \supset$
 - p v (q ^ r)
 - p + (q * r)
- Prolog

cat(X) :- furry(X), meows (X), has(X, claws)

Lisp notations

(forall ?x (implies (and (furry ?x) (meows ?x) (has ?x claws))

(cat ?x)))

Every gardener likes the sun

All purple mushrooms are poisonous

Every gardener likes the sun $\forall x \text{ gardener}(x) \rightarrow \text{likes}(x, \text{Sun})$

All purple mushrooms are poisonous

Every gardener likes the sun $\forall x \text{ gardener}(x) \rightarrow \text{likes}(x, \text{Sun})$

All purple mushrooms are poisonous $\forall x (mushroom(x) \land purple(x)) \rightarrow poisonous(x)$

Every gardener likes the sun $\forall x \text{ gardener}(x) \rightarrow \text{likes}(x, \text{Sun})$

All purple mushrooms are poisonous $\forall x (mushroom(x) \land purple(x)) \rightarrow poisonous(x)$

No purple mushroom is poisonous (two ways)

Every gardener likes the sun $\forall x \text{ gardener}(x) \rightarrow \text{likes}(x, \text{Sun})$

All purple mushrooms are poisonous $\forall x (mushroom(x) \land purple(x)) \rightarrow poisonous(x)$

No purple mushroom is poisonous (two ways) $\neg \exists x \text{ purple}(x) \land \text{mushroom}(x) \land \text{poisonous}(x)$

Every gardener likes the sun $\forall x \text{ gardener}(x) \rightarrow \text{likes}(x, \text{Sun})$

All purple mushrooms are poisonous $\forall x (mushroom(x) \land purple(x)) \rightarrow poisonous(x)$

No purple mushroom is poisonous (two ways) $\neg \exists x \text{ purple}(x) \land \text{mushroom}(x) \land \text{poisonous}(x)$ $\forall x \text{ (mushroom}(x) \land \text{purple}(x)) \rightarrow \neg \text{poisonous}(x)$

English to FOL: Counting



Use = predicate to identify different individuals

- There are <u>at least</u> two purple mushrooms
 ∃x ∃y mushroom(x) ∧ purple(x) ∧ mushroom(y) ∧ purple(y) ∧ ¬(x=y)
- There are <u>exactly</u> two purple mushrooms
 ∃x ∃y mushroom(x) ∧ purple(x) ∧ mushroom(y) ∧
 purple(y) ∧ ¬(x=y) ∧
 ∀z (mushroom(z) ∧ purple(z)) → ((x=z) ∨ (y=z))

Saying there are 802 different Pokemon will be hard!



What do these mean?

• You can fool some of the people all of the time

• You can fool all of the people some of the time

What do these mean?



Both English statements are ambiguous

• You can fool some of the people all of the time

There is a nonempty subset of people so easily fooled that you can fool that subset every time* For any given time, there is a non-empty subset at that time that you can fool

• You can fool all of the people some of the time

There are one or more times when it's possible to fool everyone*

Everybody can be fooled at some point in time



Some terms we will need

• person(x): True iff x is a person

- time(t): True iff t is a point in time
- canFool(x, t): True iff x can be fooled at time t

Note: *iff* = *if and only if* =
$$_{17} \leftrightarrow$$



You can fool some of the people all of the time

- There is a nonempty group of people so easily fooled that you can fool that group every time*
- ≡ There's (at least) one person you can fool every time
- $\exists x \forall t \text{ person}(x) \land time(t) \rightarrow canFool(x, t)$
- For any given time, there is a non-empty group at that time that you can fool
- ≡ For every time, there's a person at that time that you can fool
- $\forall t \exists x \text{ person}(x) \land time(t) \rightarrow canFool(x, t)$



You can fool all of the people some of the time

There's at least one time when you can fool everyone* $\exists t \forall x time(t) \land person(x) \rightarrow canFool(x, t)$

Everybody can be fooled at some point in time $\forall x \exists t \text{ person}(x) \land time(t) \rightarrow canFool(x, t)$

Limits of classical logic

- Note that there's no easy, natural way to talk about a few, many, most, almost all ...
- This is natural in human languages
 - There are **many** people you can fool **most** of the time
 - There are a **few** people you can fool **almost every** time
- We also can't have exceptions naturally as in human languages
 - All birds can fly, **except for** penguins, ostriches and a few other species
 - This can be represented in FOL, but it may be challenging lot of new relations, paraphrasing, and conditions needed.
 - "For all entities x, if x is a bird and x is not a penguin, not an ostrich, and not one of the specified other species, then x can fly."
- There are non-classical logic systems that can handle these problems

Limits of classical logic

- Note that there's no easy, natural way to talk about a few, many, most, almost all ...
- This is natural in human languages
 - There are **many** people you can fool **most** of the time
 - There are a **few** people you can fool **almost every** time
- We also can't have exceptions naturally as in human languages
 - All birds can fly, **except for** penguins, ostriches and a few other species
 - This can be represented in FOL, but it may be challenging lot of new relations, paraphrasing, and conditions needed.
 - "For all entities x, if x is a bird and x is not a penguin, not an ostrich, and not one of the specified other species, then x can fly."
- There are non-classical logic systems that can handle these problems

 $orall x \left(B(x) \land \neg(\operatorname{Penguin}(x) \lor \operatorname{Ostrich}(x) \lor \operatorname{OtherSpecies}(x))
ightarrow F(x)
ight)$

Representation Design



- Many options for representing even a simple fact, e.g., something's color as red, green or blue, e.g.:
 - green(kermit)
 - color(kermit, green)
 - hasProperty(kermit, color, green)
- Choice can influence how easy it is to use
- Last option of representing properties & relations as <u>triples</u> used by modern <u>knowledge graphs</u>
 - Easy to ask: What color is Kermit? What are Kermit's properties?, What green things are there? Tell me everything you know, ...

Simple genealogy KB in FOL



Design a knowledge base using FOL that

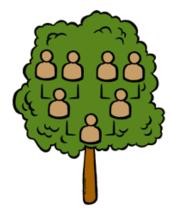
- Has facts of immediate family relations, e.g., spouses, parents, etc.
- Defines more complex relations (ancestors, relatives)
- Detect conflicts, e.g., you are your own parent
- Infers relations, e.g., grandparent from parent
- Answers queries about relationships between people

How do we approach this?

- Design an initial ontology of types, e.g.
 - e.g., person, man, woman, male, female
- Extend ontology by defining simple two argument relations, e.g.
 - spouse, has_child, has_parent
- Add general constraints to relations, e.g.
 - spouse(X,Y) => ~ X = Y
 - spouse(X,Y) => person(X), person(Y)
- Add FOL sentences for inference, e.g.
 - spouse(X,Y) ⇔ spouse(Y,X)
 - man(X) ⇔ person(X) ∧ male(X)
- Add instance data
 - e.g., spouse(djt, mt)



23



Example: A simple genealogy KB by FOL

Predicates:

- parent(x, y), child(x, y), father(x, y), daughter(x, y), etc.
- spouse(x, y), husband(x, y), wife(x,y)
- ancestor(x, y), descendant(x, y)
- male(x), female(y)
- relative(x, y)

Facts:

- husband(Joe, Mary), son(Fred, Joe)
- spouse(John, Nancy), male(John), son(Mark, Nancy)
- father(Jack, Nancy), daughter(Linda, Jack)
- daughter(Liz, Linda)
- etc.

Example Axioms

 $(\forall x,y) \text{ parent}(x, y) \leftrightarrow \text{child } (y, x)$

 $(\forall x,y)$ father(x, y) \leftrightarrow parent(x, y) \land male(x) ;similar for mother(x, y)

 $(\forall x,y)$ daughter(x, y) \leftrightarrow child(x, y) \land female(x) ;similar for son(x, y)

 $(\forall x,y)$ husband(x, y) \leftrightarrow spouse(x, y) \land male(x) ;similar for wife(x, y)

(∀x,y) spouse(x, y) ↔ spouse(y, x) ;spouse relation is symmetric

 $(\forall x,y) \text{ parent}(x, y) \rightarrow \text{ancestor}(x, y)$

 $(\forall x,y)(\exists z) \text{ parent}(x, z) \land \text{ancestor}(z, y) \rightarrow \text{ancestor}(x, y)$

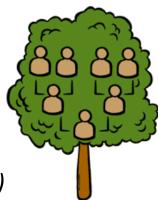
 $(\forall x,y)$ descendant $(x, y) \leftrightarrow$ ancestor(y, x)

 $(\forall x,y)(\exists z)$ ancestor $(z, x) \land ancestor(z, y) \rightarrow relative(x, y)$

 $(\forall x,y)$ spouse(x, y) \rightarrow relative(x, y) ;related by marriage

 $(\forall x,y)(\exists z) \text{ relative}(z, x) \land \text{ relative}(z, y) \rightarrow \text{ relative}(x, y) \text{ ; transitive}$

 $(\forall x,y)$ relative $(x, y) \leftrightarrow$ relative(y, x) ;symmetric



Axioms, definitions and theorems

- Axioms: facts and rules that capture (important) facts & concepts in a domain; axioms are used to prove theorems
 - Mathematicians dislike unnecessary (dependent) axioms, i.e. ones that can be derived from others
 - Dependent axioms can make reasoning faster, however
 - Choosing a good set of axioms is a design problem
- A definition of a predicate is of the form "p(X) ↔ …" and can be decomposed into two parts
 - Necessary description: " $p(x) \rightarrow ...$ "
 - Sufficient description "p(x) ← …"
 - Some concepts have definitions (e.g., triangle) and some don't (e.g., person)

More on definitions

Example: define father(x, y) by parent(x, y) and male(x)

- parent(x, y) is a necessary (but not sufficient) description of father(x, y) father(x, y) → parent(x, y)
- parent(x, y) ^ male(x) ^ age(x, 35) is a sufficient (but not necessary) description of father(x, y):
 father(x, y) ← parent(x, y) ^ male(x) ^ age(x, 35)
- parent(x, y) ^ male(x) is a necessary and sufficient description of father(x, y) parent(x, y) ^ male(x) ↔ father(x, y)

Higher-order logic

- FOL only lets us quantify over variables, and variables can only range over objects
- HOL allows us to quantify over relations, e.g. "two functions are equal iff they produce the same value for all arguments"

 $\forall f \forall g (f = g) \leftrightarrow (\forall x f(x) = g(x))$

- E.g.: (quantify over predicates) $\forall r \text{ transitive}(r) \rightarrow (\forall xyz) r(x,y) \land r(y,z) \rightarrow r(x,z))$
- More expressive, but reasoning is undecideable, in general

Expressing uniqueness

- Often want to say that there is a single, unique object that satisfies a condition
- There exists a unique x such that king(x) is true
 - $\exists x \text{ king}(x) \land \forall y \text{ (king}(y) \rightarrow x=y)$
 - $\exists x \text{ king}(x) \land \neg \exists y \text{ (king}(y) \land x \neq y)$
 - ∃! x king(x)
- Every country has exactly one ruler
 - $\forall c \text{ country}(c) \rightarrow \exists ! r \text{ ruler}(c,r)$
- lota operator: ι x P(x) means "the unique x such that p(x) is true"
 - The unique ruler of Freedonia is dead
 - dead(1 x ruler(freedonia,x))



Examples of FOL in use



- Semantics of W3C's <u>Semantic Web</u> stack (RDF, RDFS, OWL) is defined in FOL
- <u>OWL</u> Full is equivalent to FOL
- Other OWL profiles support a subset of FOL and are more efficient
- The semantics of <u>schema.org</u> is only defined in natural language text
- <u>Wikidata</u>'s knowledge graph (and Google's) has a richer schema



Examples of FOL in use



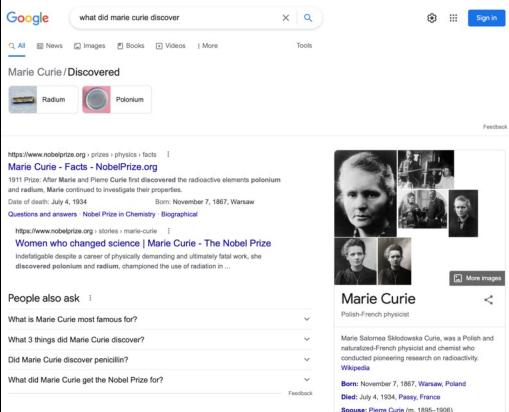
Many practical approaches embrace the approach that "some data is better than none"

- The semantics of <u>schema.org</u> is only defined in natural language text
- <u>Wikidata</u>'s knowledge graph has a rich schema
 - Many constraint/logical violations are flagged with warnings
 - However, not all, see this <u>Wikidata query</u> that finds people who are their own grandfather

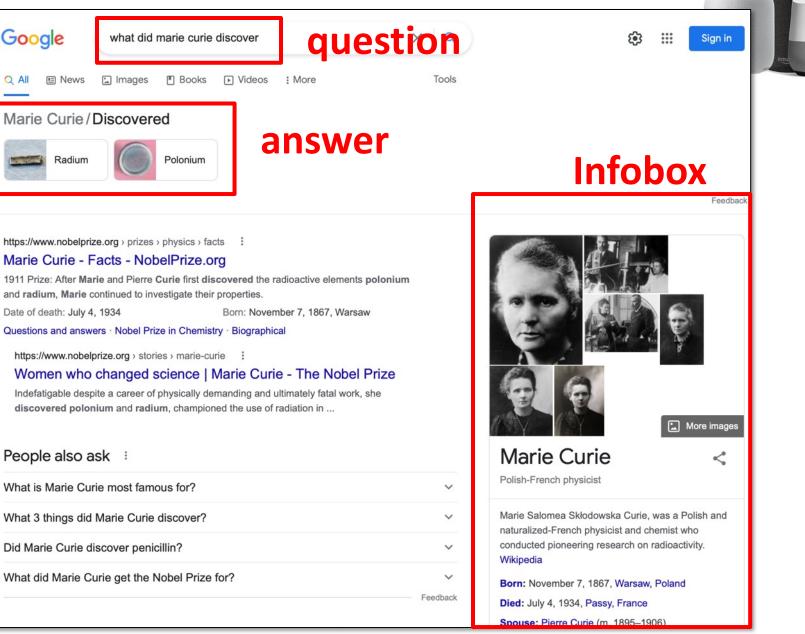
Virtual assistants and Infoboxes



- Web search engines and virtual assistants like Alexa use custom **knowledge graphs** to
 - help understand queries and content of web pages & documents
 - Answer questions
 - Show infoboxes
- Wikidata shares roots with these
- All draw on the similar knowledge, like the ~300 Wikipedia & Wikimedia sites



Virtual assistants & search engines



FOL Summary

- First order logic (FOL) introduces predicates, functions and quantifiers
- More expressive, but reasoning more complex
 - Reasoning in propositional logic is NP hard, FOL is semi-decidable
- Common AI knowledge representation language
 - Other KR languages (e.g., <u>OWL</u>) are often defined by mapping them to FOL
- FOL variables range over objects
 - HOL variables range over functions, predicates or sentences

Logical Inference: Overview

- Model checking for propositional logic
- Rule based reasoning in first-order logic
 - Inference rules and generalized modes ponens
 - Forward chaining
 - Backward chaining
- Resolution-based reasoning in first-order logic
 - Clausal form
 - Unification
 - Resolution as search

python> python
Python ...
>>> from logic import *
>>> expr('P & P==>Q & ~P==>R')

 $((P \& (P >> Q)) \& (\sim P >> R))$

>>> dpll_satisfiable(expr('P & P==>Q & ~P==>R'))
{R: True, P: True, Q: True}

>>> dpll_satisfiable(expr('P & P==>Q & ~P==>R & ~R'))
{R: False, P: True, Q: True}

>>> dpll_satisfiable(expr('P & P==>Q & ~P==>R & ~Q')) False

expr parses a string, and returns a logical expression

python> python

Python ...

>>> from logic import *

>>> expr('P & P = >Q & ~P = >R')

 $((P \& (P >> Q)) \& (\sim P >> R))$

>>> dpll_satisfiable(expr('P & P==>Q & ~P==>R'))
{R: True, P: True, Q: True}

>>> dpll_satisfiable(expr('P & P==>Q & ~P==>R & ~R'))
{R: False, P: True, Q: True}

>>> dpll_satisfiable(expr('P & P==>Q & ~P==>R & ~Q')) False

>>>

python> python

Python ...

>>> from logic import *

>>> expr('P & P==>Q & ~P==>R')

((P & (P >> Q)) & (~P >> R))

expr parses a string, and returns a logical expression

dpll_satisfiable returns a model if satisfiable else False

>>> dpll_satisfiable(expr('P & P==>Q & ~P==>R'))
{R: True, P: True, Q: True}

>>> dpll_satisfiable(expr('P & P==>Q & ~P==>R & ~R'))
{R: False, P: True, Q: True}

>>> dpll_satisfiable(expr('P & P==>Q & ~P==>R & ~Q')) False

python> python

Python ...

>>> from logic import *

>>> expr('P & P==>Q & ~P==>R')

 $((P \& (P >> Q)) \& (\sim P >> R))$

expr parses a string, and returns a logical expression

dpll_satisfiable returns a model if satisfiable else False

>>> dpll_satisfiable(expr('P & P==>Q & ~P==>R'))
{R: True, P: True, Q: True}

>>> dpll_satisfiable(expr('P & P==>Q & ~P==>R & ~R'))
{R: False, P: True, Q: True}

>>> dpll_satisfiable(expr('P & P==>Q & ~P==>R & ~Q'))
False

The KB entails Q but does not entail R 50

>>>

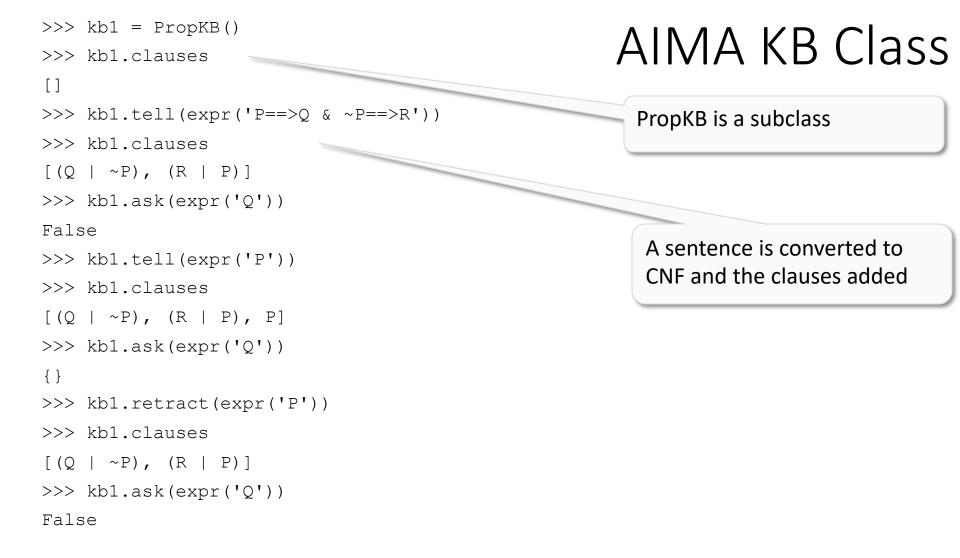
Checking Validity

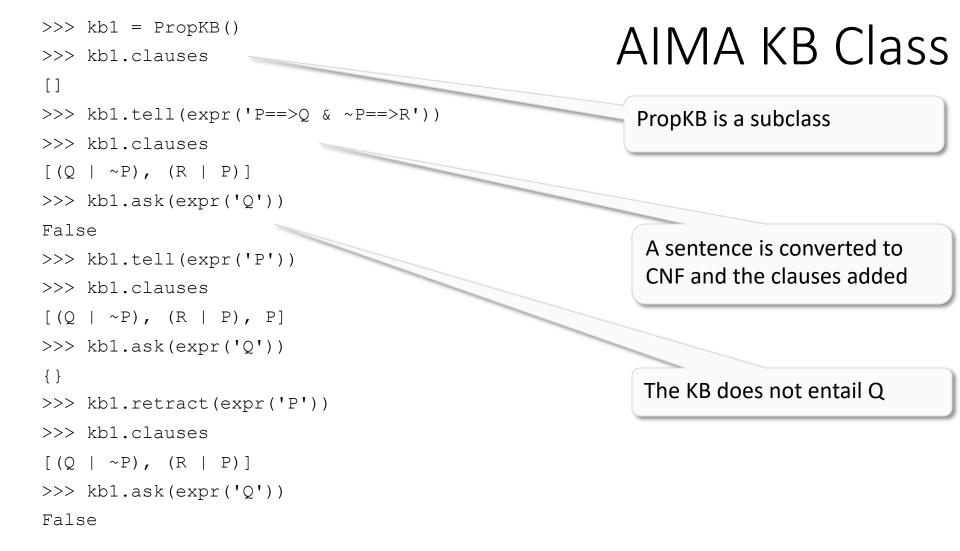
- Use the functions in aima's <u>logic.py</u> to see which of the following are valid, i.e., true in every model.
- convert these sentences to the appropriate string form that the python code uses
- use the expr() function in logic.py to turn each into an Expr object
- use the tt_true () function to check for validity.
- tt_true() checks an expression object to see if it is valid, i.e., true in all possible models.
- A valid sentence is true for all possible assignments of true and false to its variables, i.e., PV-P

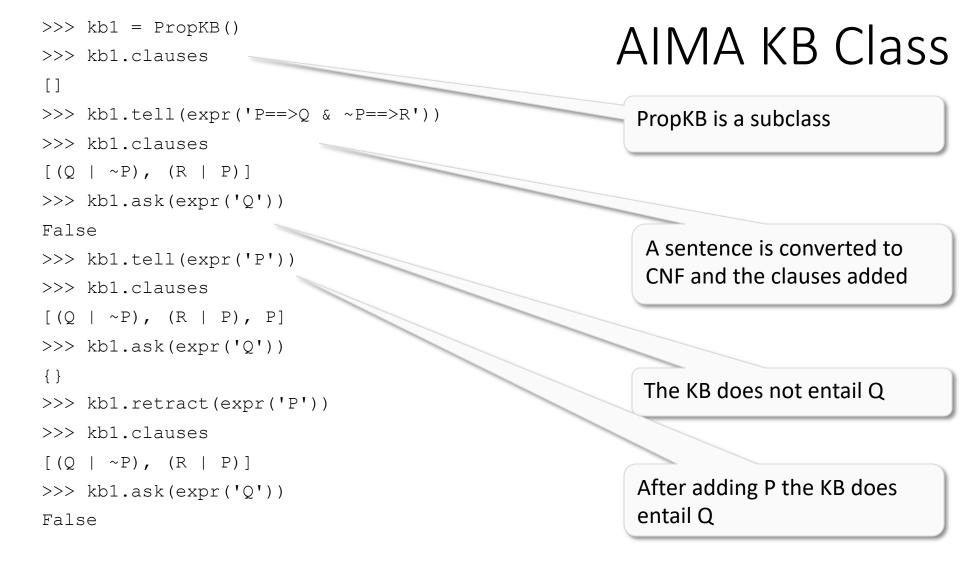
```
>>> kb1 = PropKB()
>>> kb1.clauses
[]
>>> kb1.tell(expr('P==>Q & ~P==>R'))
>>> kb1.clauses
[(Q | \sim P), (R | P)]
>>> kb1.ask(expr('Q'))
False
>>> kbl.tell(expr('P'))
>>> kb1.clauses
[(Q | ~P), (R | P), P]
>>> kb1.ask(expr('Q'))
{ }
>>> kb1.retract(expr('P'))
>>> kb1.clauses
[(Q | \sim P), (R | P)]
>>> kb1.ask(expr('Q'))
False
```

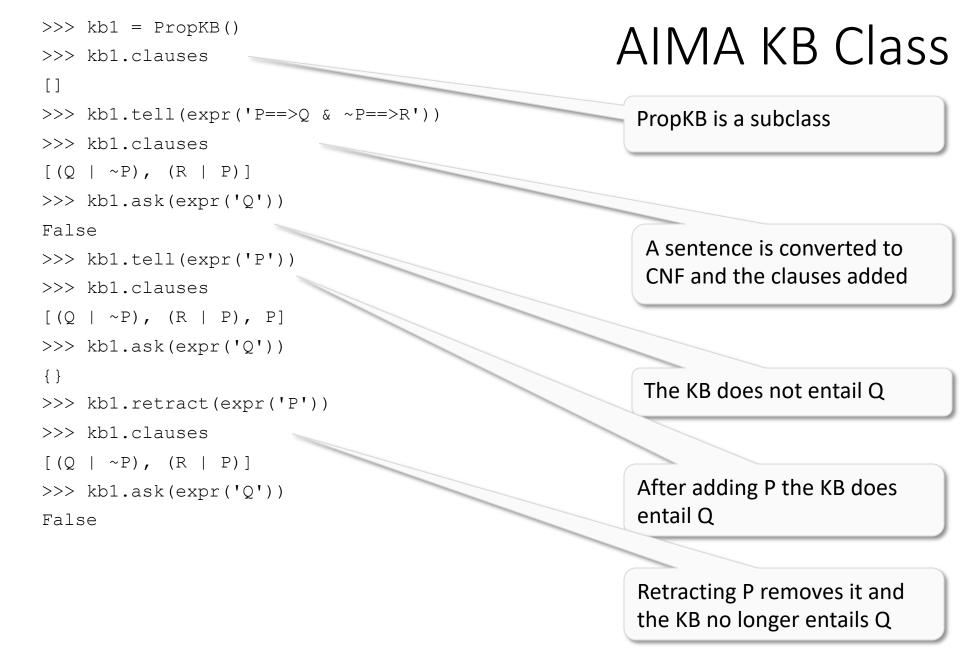
AIMA KB Class

```
>>> kb1 = PropKB()
                                                     AIMA KB Class
>>> kb1.clauses
[]
>>> kb1.tell(expr('P==>Q & ~P==>R'))
                                                       PropKB is a subclass
>>> kb1.clauses
[(Q | \sim P), (R | P)]
>>> kb1.ask(expr('Q'))
False
>>> kb1.tell(expr('P'))
>>> kb1.clauses
[(Q | ~P), (R | P), P]
>>> kb1.ask(expr('Q'))
{ }
>>> kb1.retract(expr('P'))
>>> kb1.clauses
[(Q | \sim P), (R | P)]
>>> kb1.ask(expr('Q'))
False
```









Logic Summary

- Propositional logic
 - Problems with propositional logic
- First-order logic
 - Properties, relations, functions, quantifiers, ...
 - Terms, sentences, wffs, axioms, theories, proofs, ...
 - Variations and extensions to first-order logic

Logical agents

- Reflex agents
- Representing change: situation calculus, frame problem
- Preferences on actions
- Goal-based agents