

CMSC 471

Propositional and First-Order Logic

KMA Solaiman
ksolaima@umbc.edu

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, has-color, occurs-after, owns, visits, precedes, ...
 - Properties: blue, oval, even, large, ...
 - Functions: father-of, best-friend, more-than ...

Quantifiers: \forall and \exists

- **Universal quantification**

- $(\forall x)P(X)$ means P holds for **all** values of X in the domain associated with variable¹
- E.g., $(\forall X) \text{dolphin}(X) \rightarrow \text{mammal}(X)$

- **Existential quantification**

- $(\exists x)P(X)$ means P holds for **some** value of X in domain associated with variable
- E.g., $(\exists X) \text{mammal}(X) \wedge \text{lays_eggs}(X)$
- 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: $(\forall X) \text{ student}(X) \rightarrow \text{smart}(X)$

Means: All students are smart

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

Logic: $(\forall X) \text{ student}(X) \wedge \text{smart}(X)$

Means: Everything is a student and is smart

Existential Quantifier: \exists

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

Logic: $(\exists X) \text{ student}(X) \wedge \text{ smart}(X)$

Meaning: There is a student who is smart

- Common mistake: represent this in FOL as:

Logic: $(\exists X) \text{ student}(X) \rightarrow \text{ smart}(X)$

Meaning: ?

Existential Quantifier: \exists

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

Logic: $(\exists X) \text{ student}(X) \wedge \text{ smart}(X)$

Meaning: There is a student who is smart

- Common mistake: represent this in FOL as:

Logic: $(\exists X) \text{ student}(X) \rightarrow \text{ smart}(X)$

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

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

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

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

- Scope of x
- Scope of 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: $(\forall X)(\exists Y) \text{ likes}(X,Y)$
 - Someone is liked by everyone: $(\exists Y)(\forall X) \text{ likes}(X,Y)$

Procedural example 1 (Illustrative only!)

```
def verify1():  
    # Everyone likes someone: (  $\forall x$ )( $\exists y$ ) likes(x,y)  
    for p1 in people():  
        foundLike = False  
        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 2 (Illustrative only!)

```
def verify2():
```

```
    # Someone is liked by everyone:  $(\exists y)(\forall x) \text{likes}(x,y)$ 
```

```
    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.

Connections between \forall and \exists

- We can relate sentences involving \forall and \exists 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. $\neg(\exists x) P(x) \leftrightarrow (\forall x) \neg P(x)$
- Examples
 1. All dogs don't like cats \leftrightarrow No dog likes cats
 2. Not all dogs bark \leftrightarrow There is a dog that doesn't bark
 3. All dogs sleep \leftrightarrow There is no dog that doesn't sleep
 4. There is a dog that talks \leftrightarrow Not all dogs can't talk

Notational differences

- **Different symbols** for *and*, *or*, *not*, *implies*, ...

- $\forall \exists \Rightarrow \Leftrightarrow \wedge \vee \neg \bullet \supset$

- $p \vee (q \wedge r)$

- $p + (q * r)$

- **Prolog**

$\text{cat}(X) \text{ :- furry}(X), \text{ meows}(X), \text{ has}(X, \text{claws})$

- **Lisp notations**

(forall ?x (implies (and (furry ?x)

(meows ?x)

(has ?x claws))

(cat ?x)))

Translating English to FOL

Every gardener likes the sun

All purple mushrooms are poisonous

Translating English to FOL

Every gardener likes the sun

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

All purple mushrooms are poisonous

Translating English to FOL

Every gardener likes the sun

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

All purple mushrooms are poisonous

$\forall x (\text{mushroom}(x) \wedge \text{purple}(x)) \rightarrow \text{poisonous}(x)$

Translating English to FOL

Every gardener likes the sun

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

All purple mushrooms are poisonous

$\forall x (\text{mushroom}(x) \wedge \text{purple}(x)) \rightarrow \text{poisonous}(x)$

No purple mushroom is poisonous (two ways)

Translating English to FOL

Every gardener likes the sun

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

All purple mushrooms are poisonous

$$\forall x (\text{mushroom}(x) \wedge \text{purple}(x)) \rightarrow \text{poisonous}(x)$$

No purple mushroom is poisonous (two ways)

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

Translating English to FOL

Every gardener likes the sun

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

All purple mushrooms are poisonous

$$\forall x (\text{mushroom}(x) \wedge \text{purple}(x)) \rightarrow \text{poisonous}(x)$$

No purple mushroom is poisonous (two ways)

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

$$\forall x (\text{mushroom}(x) \wedge \text{purple}(x)) \rightarrow \neg \text{poisonous}(x)$$



English to FOL: Counting

Use = predicate to identify different individuals

- There are **at least two** purple mushrooms

$$\exists x \exists y \text{ mushroom}(x) \wedge \text{purple}(x) \wedge \text{mushroom}(y) \wedge \text{purple}(y) \wedge \neg(\mathbf{x=y})$$

- There are **exactly two** purple mushrooms

$$\begin{aligned} &\exists x \exists y \text{ mushroom}(x) \wedge \text{purple}(x) \wedge \text{mushroom}(y) \wedge \\ &\text{purple}(y) \wedge \neg(\mathbf{x=y}) \wedge \\ &\forall z (\text{mushroom}(z) \wedge \text{purple}(z)) \rightarrow ((\mathbf{x=z}) \vee (\mathbf{y=z})) \end{aligned}$$

Saying there are 802 different Pokemon will be hard!

Translating English to FOL



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**

Translating English to FOL



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

* Most common interpretation, I think



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* = \leftrightarrow

Translating English to FOL



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

$\exists x \forall t \text{ person}(x) \wedge \text{time}(t) \rightarrow \text{canFool}(x, t)$

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

$\forall t \exists x \text{ person}(x) \wedge \text{time}(t) \rightarrow \text{canFool}(x, t)$

* Most common interpretation, I think

Translating English to FOL



You can fool all of the people some of the time

There's at least one time when you can fool everyone*

$$\exists t \forall x \text{time}(t) \wedge \text{person}(x) \rightarrow \text{canFool}(x, t)$$

Everybody can be fooled at some point in time

$$\forall x \exists t \text{person}(x) \wedge \text{time}(t) \rightarrow \text{canFool}(x, t)$$

* Most common interpretation, I think

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

$$\forall x (B(x) \wedge \neg(\text{Penguin}(x) \vee \text{Ostrich}(x) \vee \text{OtherSpecies}(x)) \rightarrow F(x))$$



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 triples used by modern knowledge graphs
 - 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.
 - $\text{spouse}(X,Y) \Rightarrow \sim X = Y$
 - $\text{spouse}(X,Y) \Rightarrow \text{person}(X), \text{person}(Y)$
- Add FOL sentences for inference, e.g.
 - $\text{spouse}(X,Y) \Leftrightarrow \text{spouse}(Y,X)$
 - $\text{man}(X) \Leftrightarrow \text{person}(X) \wedge \text{male}(X)$
- Add instance data
 - e.g., $\text{spouse}(\text{djt}, \text{mt})$

Example: A simple genealogy KB by FOL

Predicates:

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

Facts:

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

Example Axioms



$(\forall x,y)$ parent(x, y) \leftrightarrow child (y, x)

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

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

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

$(\forall x,y)$ spouse(x, y) \leftrightarrow spouse(y, x) ;*spouse relation is symmetric*

$(\forall x,y)$ parent(x, y) \rightarrow ancestor(x, y)

$(\forall x,y)(\exists z)$ parent(x, z) \wedge ancestor(z, y) \rightarrow ancestor(x, y)

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

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

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

$(\forall x,y)(\exists z)$ relative(z, x) \wedge relative(z, y) \rightarrow relative(x, y) ;*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 “ $p(X) \leftrightarrow \dots$ ” and can be decomposed into two parts
 - **Necessary** description: “ $p(x) \rightarrow \dots$ ”
 - **Sufficient** description “ $p(x) \leftarrow \dots$ ”
 - Some concepts have definitions (e.g., triangle) and some don't (e.g., person)

More on definitions

Example: define $\text{father}(x, y)$ by $\text{parent}(x, y)$ and $\text{male}(x)$

- **$\text{parent}(x, y)$** is a necessary (but not sufficient) description of $\text{father}(x, y)$
$$\text{father}(x, y) \rightarrow \text{parent}(x, y)$$
- **$\text{parent}(x, y) \wedge \text{male}(x) \wedge \text{age}(x, 35)$** is a sufficient (but not necessary) description of $\text{father}(x, y)$:
$$\text{father}(x, y) \leftarrow \text{parent}(x, y) \wedge \text{male}(x) \wedge \text{age}(x, 35)$$
- **$\text{parent}(x, y) \wedge \text{male}(x)$** is a necessary and sufficient description of $\text{father}(x, y)$
$$\text{parent}(x, y) \wedge \text{male}(x) \leftrightarrow \text{father}(x, y)$$

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)

$$\forall r \text{ transitive}(r) \rightarrow (\forall xyz) r(x,y) \wedge r(y,z) \rightarrow r(x,z)$$

- More expressive, but reasoning is undecidable, 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 $\text{king}(x)$ is true
 - $\exists x \text{ king}(x) \wedge \forall y (\text{king}(y) \rightarrow x=y)$
 - $\exists x \text{ king}(x) \wedge \neg \exists y (\text{king}(y) \wedge x \neq y)$
 - $\exists! x \text{ king}(x)$
- Every country has exactly one ruler
 - $\forall c \text{ country}(c) \rightarrow \exists! r \text{ ruler}(c,r)$
- Iota operator: $\iota x P(x)$ means “the unique x such that $p(x)$ is true”
 - The unique ruler of Freedonia is dead
 - $\text{dead}(\iota x \text{ ruler}(\text{freedonia},x))$





Examples of FOL in use

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



schema.org

Examples of FOL in use



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

- The semantics of schema.org is only defined in natural language text
- [Wikidata](https://www.wikidata.org)'s knowledge graph has a rich schema
 - Many constraint/logical violations are flagged with warnings
 - However, not all, see this [Wikidata query](#) 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

The screenshot shows a Google search for "what did marie curie discover". The search results include a link to "Marie Curie - Facts - NobelPrize.org" and a link to "Women who changed science | Marie Curie - The Nobel Prize". Below the search results is an infobox for Marie Curie, which includes a portrait and a brief biography. The infobox text reads: "Marie Salomea Skłodowska Curie, was a Polish and naturalized-French physicist and chemist who conducted pioneering research on radioactivity. Wikipedia". It also lists her birth date as "November 7, 1867, Warsaw, Poland" and her death date as "July 4, 1934, Passy, France". The infobox also mentions her spouse: "Pierre Curie (m. 1895–1906)".



Virtual assistants & search engines



Google **question** Sign in

[All](#) [News](#) [Images](#) [Books](#) [Videos](#) [More](#) [Tools](#)

Marie Curie / Discovered **answer**

 Radium  Polonium **Infobox**

<https://www.nobelprize.org/prizes/physics/facts> **Marie Curie - Facts - NobelPrize.org**
1911 Prize: After Marie and Pierre Curie first discovered the radioactive elements polonium and radium, Marie continued to investigate their properties.
Date of death: July 4, 1934 Born: November 7, 1867, Warsaw
[Questions and answers](#) · [Nobel Prize in Chemistry](#) · [Biographical](#)







<https://www.nobelprize.org/stories/marie-curie> **Women who changed science | Marie Curie - The Nobel Prize**
Indefatigable despite a career of physically demanding and ultimately fatal work, she discovered polonium and radium, championed the use of radiation in ...

People also ask

- What is Marie Curie most famous for? ▾
- What 3 things did Marie Curie discover? ▾
- Did Marie Curie discover penicillin? ▾
- What did Marie Curie get the Nobel Prize for? ▾

[Feedback](#)

[Feedback](#)

[More images](#)

Marie Curie [Share](#)

Polish-French physicist

Marie Salomea Skłodowska Curie, was a Polish and naturalized-French physicist and chemist who conducted pioneering research on radioactivity.
[Wikipedia](#)

Born: November 7, 1867, Warsaw, Poland
Died: July 4, 1934, Passy, France
Spouse: Pierre Curie (m. 1895–1906)

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., [OWL](#)) 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

Model Checking using the AIMA Code

```
python> python
```

```
Python ...
```

```
>>> from logic import *
```

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

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

```
>>> dp11_satisfiable(expr('P & P==>Q & ~P==>R'))
```

```
{R: True, P: True, Q: True}
```

```
>>> dp11_satisfiable(expr('P & P==>Q & ~P==>R & ~R'))
```

```
{R: False, P: True, Q: True}
```

```
>>> dp11_satisfiable(expr('P & P==>Q & ~P==>R & ~Q'))
```

```
False
```

```
>>>
```

Model Checking using the AIMA Code

```
python> python
```

```
Python ...
```

```
>>> from logic import *
```

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

```
((P & (P >> Q)) & (~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

Model Checking using the AIMA Code

```
python> python
```

```
Python ...
```

```
>>> from logic import *
```

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

```
((P & (P >> Q)) & (~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

dpll_satisfiable returns a model if satisfiable else False

Model Checking using the AIMA Code

```
python> python
```

```
Python ...
```

```
>>> from logic import *
```

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

```
((P & (P >> Q)) & (~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

dpll_satisfiable returns a model if satisfiable else False

The KB entails Q but does not entail R

Checking Validity

- Use the functions in aim's [logic.py](#) 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., $P \vee \neg P$

AIMA KB Class

```
>>> kb1 = PropKB()
>>> kb1.clauses
[]
>>> kb1.tell(expr('P==>Q & ~P==>R'))
>>> kb1.clauses
[(Q | ~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 | ~P), (R | P)]
>>> kb1.ask(expr('Q'))
False
```

AIMA KB Class

```
>>> kb1 = PropKB()
>>> kb1.clauses
[]
>>> kb1.tell(expr('P==>Q & ~P==>R'))
>>> kb1.clauses
[(Q | ~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 | ~P), (R | P)]
>>> kb1.ask(expr('Q'))
False
```

PropKB is a subclass

AIMA KB Class

```
>>> kb1 = PropKB()
>>> kb1.clauses
[]
>>> kb1.tell(expr('P==>Q & ~P==>R'))
>>> kb1.clauses
[(Q | ~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 | ~P), (R | P)]
>>> kb1.ask(expr('Q'))
False
```

PropKB is a subclass

A sentence is converted to CNF and the clauses added

AIMA KB Class

```
>>> kb1 = PropKB()
>>> kb1.clauses
[]
>>> kb1.tell(expr('P==>Q & ~P==>R'))
>>> kb1.clauses
[(Q | ~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 | ~P), (R | P)]
>>> kb1.ask(expr('Q'))
False
```

PropKB is a subclass

A sentence is converted to CNF and the clauses added

The KB does not entail Q

AIMA KB Class

```
>>> kb1 = PropKB()
>>> kb1.clauses
[]
>>> kb1.tell(expr('P==>Q & ~P==>R'))
>>> kb1.clauses
[(Q | ~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 | ~P), (R | P)]
>>> kb1.ask(expr('Q'))
False
```

PropKB is a subclass

A sentence is converted to CNF and the clauses added

The KB does not entail Q

After adding P the KB does entail Q

AIMA KB Class

```
>>> kb1 = PropKB()
>>> kb1.clauses
[]
>>> kb1.tell(expr('P==>Q & ~P==>R'))
>>> kb1.clauses
[(Q | ~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 | ~P), (R | P)]
>>> kb1.ask(expr('Q'))
False
```

PropKB is a subclass

A sentence is converted to CNF and the clauses added

The KB does not entail Q

After adding P the KB does entail Q

Retracting P removes it and the KB no longer entails Q

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