First-Order Logic

Outline

I. Syntax of FOL

II. Quantifiers

III. Model for FOL

IV. Assertions & queries in FOL

* Figures are from the textbook site unless a source is specifically cited.
I. Propositional Logic: Strength and Weakness

- Programming languages lack a general mechanism for deriving facts from other facts.

- They lack the expressiveness required to handle partial information.
I. Propositional Logic: Strength and Weakness

- Programming languages lack a general mechanism for deriving facts from other facts.

- They lack the expressiveness required to handle partial information.

- Propositional logic addresses the above issues.
I. Propositional Logic: Strength and Weakness

♦ Programming languages lack a general mechanism for deriving facts from other facts.

♦ They lack the expressiveness required to handle partial information.

♦ Propositional logic addresses the above issues.

♦ It also has *compositionality* – the meaning of a sentence is a function of the meanings of its parts.

\[-\text{rain} \lor \neg \text{outside} \lor \text{wet}\]
I. Propositional Logic: Strength and Weakness

ляем Programming languages lack a general mechanism for deriving facts from other facts.

♦ They lack the expressiveness required to handle partial information.

♦ Propositional logic addresses the above issues.

♦ It also has compositionality – the meaning of a sentence is a function of the meanings of its parts.

\[ \neg \text{rain} \lor \neg \text{outside} \lor \text{wet} \]

♦ It lacks the expressive power to describe an environment with many objects.
I. Propositional Logic: Strength and Weakness

- Programming languages lack a general mechanism for deriving facts from other facts.

- They lack the expressiveness required to handle partial information.

- Propositional logic addresses the above issues.

- It also has *compositionality* – the meaning of a sentence is a function of the meanings of its parts.

  $$\neg \text{rain} \lor \neg \text{outside} \lor \text{wet}$$

- It lacks the expressive power to describe an environment with *many objects*.

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

  // Squares adjacent to pits are breezy.
I. Propositional Logic: Strength and Weakness

- Programming languages lack a general mechanism for deriving facts from other facts.

- They lack the expressiveness required to handle partial information.

- Propositional logic addresses the above issues.

- It also has compositionality – the meaning of a sentence is a function of the meanings of its parts.

\[ \neg \text{rain} \lor \neg \text{outside} \lor \text{wet} \]

- It lacks the expressive power to describe an environment with many objects.

\[ B_{1,1} \iff (P_{1,2} \lor P_{2,1}) \]
\[ B_{1,2} \iff (P_{1,1} \lor P_{1,3} \lor P_{2,2}) \]
\[ \vdots \]

- Propositional logic assumes the world contains facts only.
Combining Formal and Natural Languages

First-order logic

- built around objects and relations
  - Objects: people, houses, cars, trees, colors, days, ...
  - Relations:
    - unary properties such as big, windy, ...
    - $n$-ary properties such as bigger than, parent of, on, owns, ...
  - Functions: square of, best friend, age, ...

- capable of expressing facts about some or all objects
# Formal Languages

<table>
<thead>
<tr>
<th>Language</th>
<th>Ontological Commitment (What exists in the world)</th>
<th>Epistemological Commitment (What an agent believes about facts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propositional logic</td>
<td>facts</td>
<td>true/false/unknown</td>
</tr>
<tr>
<td>First-order logic</td>
<td>facts, objects, relations</td>
<td>true/false/unknown</td>
</tr>
<tr>
<td>Temporal logic</td>
<td>facts, objects, relations, times</td>
<td>true/false/unknown</td>
</tr>
<tr>
<td>Probability theory</td>
<td>facts</td>
<td>degree of belief $\in [0, 1]$</td>
</tr>
<tr>
<td>Fuzzy logic</td>
<td>facts with degree of truth $\in [0, 1]$</td>
<td>known interval value</td>
</tr>
</tbody>
</table>
Alphabet of First-Order Logic

_logical symbols_

- connectives: \( \land, \lor, \Rightarrow, \Leftrightarrow, \neg \)
- parenthesis: (, ) and punctuation , 
- equality: =
- quantifiers: \( \forall \text{ (universal quantification)}, \exists \text{ (existential quantification)} \)
- variables: \( x, y, z, \ldots; x_1, x_2, \ldots \)
Alphabet of First-Order Logic

♦ Logical symbols
  • connectives: $\wedge, \vee, \Rightarrow, \Leftrightarrow, \neg$
  • parenthesis: (, ) and punctuation ,
  • equality: $=$
  • quantifiers: $\forall$ (universal quantification), $\exists$ (existential quantification)
  • variables: $x, y, z, \ldots; x_1, x_2, \ldots$

♦ Non-logical symbols
  • constants: Socrates, Turing, 1, earth, …
Alphabet of First-Order Logic

♦ Logical symbols
  • connectives: $\land$, $\lor$, $\Rightarrow$, $\Leftrightarrow$, $\neg$
  • parenthesis: (, ) and punctuation ,
  • equality: =
  • quantifiers: $\forall$ (universal quantification), $\exists$ (existential quantification)
  • variables: $x, y, z, ...$; $x_1, x_2, ...$

♦ Non-logical symbols
  • constants: Socrates, Turing, 1, earth, ...
  • predicate symbols: $true$, $false$
Alphabet of First-Order Logic

◆ Logical symbols
  • connectives: $\land$, $\lor$, $\Rightarrow$, $\Leftrightarrow$, $\neg$
  • parenthesis: (, ) and punctuation ,
  • equality: =
  • quantifiers: $\forall$ (universal quantification), $\exists$ (existential quantification)
  • variables: $x, y, z, \ldots; x_1, x_2, \ldots$

◆ Non-logical symbols
  • constants: Socrates, Turing, 1, earth, ...
  • predicate symbols: $true, false$
    $Father(x, y)$ // $x$ is father of $y$
    $Female(x)$ // $x$ is female
Alphabet of First-Order Logic

♦ Logical symbols
  • connectives: ∧, ∨, ⇒, ⇔, ¬
  • parenthesis: (, ) and punctuation ,
  • equality: =
  • quantifiers: ∀ (universal quantification), ∃ (existential quantification)
  • variables: x, y, z, ...; x₁, x₂, ...

♦ Non-logical symbols
  • constants: Socrates, Turing, 1, earth, ...
  • predicate symbols: true, false
    Father(x, y) // x is father of y
    Female(x) // x is female
  • function symbols: gcd(x, y) // greatest common divisor of x and y
    FatherOf(x) // father of x
Terms and Atomic Sentences

♦ Terms

- constants: Socrates, Turing, 1, earth, …
Terms and Atomic Sentences

Terms

- constants: Socrates, Turing, 1, earth, ...
- variables: $x, y, z, ...; x_1, x_2, ...$
Terms and Atomic Sentences

Terms

- constants: Socrates, Turing, 1, earth, ...
- variables: $x, y, z, ...$; $x_1, x_2, ...$
- functions: $gcd(x, y)$, $FatherOf(x)$, ...
Terms and Atomic Sentences

Terms

- constants: Socrates, Turing, 1, earth, ...
- variables: $x, y, z, ...; x_1, x_2, ...$
- functions: $gcd(x, y), FatherOf(x), ...$

$$f(x_1, x_2, ..., x_n)$$

function symbol terms
Terms and Atomic Sentences

◆ Terms

  • constants: Socrates, Turing, 1, earth, ...
  • variables: $x, y, z, ...; x_1, x_2, ...$
  • functions: $gcd(x, y)$, $FatherOf(x)$, ...

◆ Atomic sentences

  • predicates: $true, false$
Terms and Atomic Sentences

Terms

- constants: Socrates, Turing, 1, earth, ...
- variables: \( x, y, z, \ldots \); \( x_1, x_2, \ldots \)
- functions: \( \text{gcd}(x, y) \), \( \text{FatherOf}(x) \), ...

Atomic sentences

- predicates: \( \text{true} \), \( \text{false} \)
  
  \( \text{Mother}(\text{Aphrodite, Harmonia}) \)
  
  \( \text{Male}(\text{John}) \)

Terms and Atomic Sentences

- **Terms**
  - constants: Socrates, Turing, 1, earth, ...
  - variables: $x, y, z, ...; x_1, x_2, ...$
  - functions: $gcd(x, y), FatherOf(x), ...$

- **Atomic sentences**
  - predicates: $true$, $false$
    - $Mother(Aphrodite, Harmonia)$
    - $Male(John)$
  - term equalities
    - $FatherOf(Apollo) = Zeus$

Complex Sentences

- made of atomic sentences using logical connectives

\[ Father(x, y) \Rightarrow Male(x) \]
\[ Female(x) \lor \neg Mother(x, y) \]
\[ Likes(Mary, John) \iff Likes(John, Mary) \]
\[ (Parent(x, y) \land Parent(y, z)) \Rightarrow GrandParent(x, z) \]
Complex Sentences

• made of atomic sentences using logical connectives

\[ Father(x, y) \Rightarrow Male(x) \]
\[ Female(x) \lor \neg Mother(x, y) \]
\[ Likes(Mary, John) \iff Likes(John, Mary) \]
\[ (Parent(x, y) \land Parent(y, z)) \Rightarrow GrandParent(x, z) \]

• universal quantification

\[ \forall x \ Circle(x) \Rightarrow Ellipse(x) \quad // \text{Every circle is an ellipse.} \]
\[ \neg \forall x \ Likes(x, \text{sushi}) \quad // \text{Not everyone likes sushi.} \]
\[ \forall x \ Integer(x) \Rightarrow (\text{Even}(x) \lor \text{Odd}(x)) \quad // \text{Every integer is either even or odd.} \]
Complex Sentences

• made of atomic sentences using logical connectives

  \[Father(x, y) \Rightarrow Male(x)\]
  \[Female(x) \lor \neg Mother(x, y)\]
  \[Likes(Mary, John) \Leftrightarrow Likes(John, Mary)\]
  \[(Parent(x, y) \land Parent(y, z)) \Rightarrow GrandParent(x, z)\]

• universal quantification

  \[\forall x \ Circle(x) \Rightarrow Ellipse(x)\]  // Every circle is an ellipse.
  \[\neg \forall x \ Likes(x, \text{sushi})\]  // Not everyone likes sushi.
  \[\forall x \ Integer(x) \Rightarrow (Even(x) \lor Odd(x))\]  // Every integer is either even or odd.

• existential quantification

  \[\exists x \ Star(x) \land \neg (x = \text{Sun})\]  // There are stars other than the sun.
  \[\exists x \ Whale(x) \land (Age(x) = 200)\]  // Some whales live to 200 years.
Syntax of First-Order Logic

\[
\begin{align*}
\text{Sentence} & \rightarrow \text{AtomicSentence} \mid \text{ComplexSentence} \\
\text{AtomicSentence} & \rightarrow \text{Predicate} \mid \text{Predicate}(\text{Term}, \ldots) \mid \text{Term} = \text{Term} \\
\text{ComplexSentence} & \rightarrow (\text{Sentence}) \\
& \mid \lnot \text{Sentence} \\
& \mid \text{Sentence} \land \text{Sentence} \\
& \mid \text{Sentence} \lor \text{Sentence} \\
& \mid \text{Sentence} \implies \text{Sentence} \\
& \mid \text{Sentence} \iff \text{Sentence} \\
& \mid \text{Quantifier} \ \text{Variable}, \ldots \ \text{Sentence}
\end{align*}
\]

\[
\begin{align*}
\text{Term} & \rightarrow \text{Function}(\text{Term}, \ldots) \\
& \mid \text{Constant} \\
& \mid \text{Variable}
\end{align*}
\]

\[
\begin{align*}
\text{Quantifier} & \rightarrow \forall \mid \exists \\
\text{Constant} & \rightarrow a \mid x_1 \mid \text{John} \mid \cdots \\
\text{Variable} & \rightarrow a \mid x \mid s \mid \cdots \\
\text{Predicate} & \rightarrow \text{True} \mid \text{False} \mid \text{After} \mid \text{Loves} \mid \text{Raining} \mid \cdots \\
\text{Function} & \rightarrow \text{Mother} \mid \text{LeftLeg} \mid \cdots
\end{align*}
\]

\text{OPERATOR PRECEDENCE} : \neg, =, \land, \lor, \implies, \iff
II. Scope of a Quantifier

- Quantifiers $\forall$ and $\exists$ have the lowest precedence.

* In symbolic logic, $\forall$ and $\exists$ have the highest precedence.
II. Scope of a Quantifier

- Quantifiers $\forall$ and $\exists$ have the lowest precedence.

$$ \forall x \; P(x) \Rightarrow Q(x) \quad \equiv \quad \forall x \; (P(x) \Rightarrow Q(x)) $$

* In symbolic logic, $\forall$ and $\exists$ have the highest precedence.
II. Scope of a Quantifier

- Quantifiers $\forall$ and $\exists$ have the lowest precedence.

\[
\forall x \ P(x) \Rightarrow Q(x) \quad \equiv \quad \forall x \ (P(x) \Rightarrow Q(x))
\]

* In symbolic logic, $\forall$ and $\exists$ have the highest precedence.
II. Scope of a Quantifier

- Quantifiers $\forall$ and $\exists$ have the lowest precedence.

\[
\forall x \ P(x) \Rightarrow Q(x) \equiv \forall x \ (P(x) \Rightarrow Q(x))
\]

scope of $\forall$

\[
\forall x \ P(x) \Rightarrow Q(x) \lor \exists y \ R(x, y) \lor S(y) \land T(x, y)
\]

* In symbolic logic, $\forall$ and $\exists$ have the highest precedence.
II. Scope of a Quantifier

- Quantifiers ∀ and ∃ have the lowest precedence.

\[ \forall x \ P(x) \Rightarrow Q(x) \equiv \forall x \ (P(x) \Rightarrow Q(x)) \]

scope of ∀

\[ \forall x \ P(x) \Rightarrow Q(x) \lor \exists y \ R(x, y) \lor S(y) \land T(x, y) \]

\[ \equiv \forall x \ (P(x) \Rightarrow (Q(x) \lor \exists y \ (R(x, y) \lor (S(y) \land T(x, y))))) \]

* In symbolic logic, ∀ and ∃ have the highest precedence.
II. Scope of a Quantifier

♦ Quantifiers ∀ and ∃ have the lowest precedence.

\[ \forall x \; P(x) \Rightarrow Q(x) \equiv \forall x \; (P(x) \Rightarrow Q(x)) \]

scope of ∀

\[ \forall x \; P(x) \Rightarrow Q(x) \lor \exists y \; R(x, y) \lor S(y) \land T(x, y) \]

\[ \equiv \forall x \; (P(x) \Rightarrow (Q(x) \lor \exists y \; (R(x, y) \lor (S(y) \land T(x, y)))))) \]

* In symbolic logic, ∀ and ∃ have the highest precedence.
II. Scope of a Quantifier

- Quantifiers $\forall$ and $\exists$ have the lowest precedence.

\[
\forall x \ P(x) \Rightarrow Q(x) \equiv \forall x \ (P(x) \Rightarrow Q(x))
\]

\[
\forall x \ P(x) \Rightarrow Q(x) \lor \exists y \ R(x,y) \lor S(y) \land T(x,y)
\]

\[
\equiv \forall x \ (P(x) \Rightarrow (Q(x) \lor \exists y \ (R(x,y) \lor (S(y) \land T(x,y)))))
\]

* In symbolic logic, $\forall$ and $\exists$ have the highest precedence.
II. Scope of a Quantifier

- Quantifiers $\forall$ and $\exists$ have the lowest precedence.

\[
\forall x \ P(x) \Rightarrow Q(x) \equiv \forall x \ (P(x) \Rightarrow Q(x))
\]

- Each of $\forall$ and $\exists$ quantifies the remaining scope of the innermost pair of parentheses containing it.

* In symbolic logic, $\forall$ and $\exists$ have the highest precedence.
II. Scope of a Quantifier

- Quantifiers $\forall$ and $\exists$ have the lowest precedence.

\[
\forall x \ P(x) \Rightarrow Q(x) \equiv \forall x \ (P(x) \Rightarrow Q(x))
\]

- Each of $\forall$ and $\exists$ quantifies the remaining scope of the innermost pair of parentheses containing it.

\[
\forall x \ P(x) \Rightarrow Q(x) \lor \exists y \ R(x, y) \lor S(y) \land T(x, y)
\]

- In symbolic logic, $\forall$ and $\exists$ have the highest precedence.
Free and Bound Variables

A variable occurrence is *bound* in a formula if it is quantified.
A variable occurrence is *free* in a formula if it is not quantified.
A variable occurrence is *bound* in a formula if it is quantified.
A variable occurrence is *free* in a formula if it is not quantified.

\[ \forall x \ Father(x, y) \Rightarrow Male(x) \]

\( x \) is bound while \( y \) is free.
A variable occurrence is *bound* in a formula if it is quantified. A variable occurrence is *free* in a formula if it is not quantified.

\[ \forall x \ Father(x, y) \Rightarrow Male(x) \]  
\[ x \text{ is bound while } y \text{ is free.} \]

\[ \neg \forall x \exists y \exists z \forall s \forall t \ P(x, y, z, s, t) \]  
\[ x, y, z, s, t \text{ are all bound} \]
Free and Bound Variables

A variable occurrence is *bound* in a formula if it is quantified. A variable occurrence is *free* in a formula if it is not quantified.

\[
\forall x \ Father(x, y) \Rightarrow Male(x)
\]

\(x\) is bound while \(y\) is free.

\[
\neg \forall x \exists y \exists z \forall s \forall t \ P(x, y, z, s, t)
\]

\(x, y, z, s, t\) are all bound

\[
\forall x \ \forall y \ (P(x) \Rightarrow Q(x, f(y), z))
\]

\(x, y\) are bound while \(z\) is free.
Free and Bound Variables

A variable occurrence is *bound* in a formula if it is quantified.
A variable occurrence is *free* in a formula if it is not quantified.

\[
\forall x \ Father(x, y) \Rightarrow Male(x)
\]

\[
\neg \forall x \exists y \exists z \forall s \forall t \ P(x, y, z, s, t)
\]

\[
\forall x \forall y \ (P(x) \Rightarrow Q(x, f(y), z))
\]

Free and bound variables can have the same name.

\[
P(x) \Rightarrow \exists x \ Q(x)
\]
Free and Bound Variables

A variable occurrence is *bound* in a formula if it is quantified.
A variable occurrence is *free* in a formula if it is not quantified.

\[ \forall x \ Father(x, y) \Rightarrow Male(x) \]

- \( x \) is bound while \( y \) is free.

\[ \neg \forall x \exists y \exists z \forall s \forall t \ P(x, y, z, s, t) \]

- \( x, y, z, s, t \) are all bound

\[ \forall x \forall y \ (P(x) \Rightarrow Q(x, f(y), z)) \]

- \( x, y \) are bound while \( z \) is free.

Free and bound variables can have the same name.

\[ P(x) \Rightarrow \exists x \ Q(x) \]

- \( x \) is free
Free and Bound Variables

A variable occurrence is *bound* in a formula if it is quantified. A variable occurrence is *free* in a formula if it is not quantified.

\[ \forall x \ Father(x, y) \Rightarrow Male(x) \]

\( x \) is bound while \( y \) is free.

\[ \neg \forall x \exists y \exists z \forall s \forall t \ P(x, y, z, s, t) \]

\( x, y, z, s, t \) are all bound

\[ \forall x \forall y \ (P(x) \Rightarrow Q(x, f(y), z)) \]

\( x, y \) are bound while \( z \) is free.

Free and bound variables can have the same name.

\[ P(x) \Rightarrow \exists x \ Q(x) \]

free \hspace{1cm} bound
A variable occurrence is *bound* in a formula if it is quantified. A variable occurrence is *free* in a formula if it is not quantified.

- $\forall x \, \text{Father}(x, y) \Rightarrow \text{Male}(x)$  
  $x$ is bound while $y$ is free.

- $\neg \forall x \exists y \exists z \forall s \forall t \, P(x, y, z, s, t)$  
  $x, y, z, s, t$ are all bound

- $\forall x \, \forall y \, (P(x) \Rightarrow Q(x, f(y), z))$  
  $x, y$ are bound while $z$ is free.

Free and bound variables can have the same name.

- $P(x) \Rightarrow \exists x \, Q(x)$  
  free

- $P(x) \Rightarrow (\exists x \, Q(x)) \land R(x)$  
  bound
Free and Bound Variables

A variable occurrence is *bound* in a formula if it is quantified.
A variable occurrence is *free* in a formula if it is not quantified.

\[
\forall x \; \text{Father}(x, y) \Rightarrow \text{Male}(x)
\]
x is bound while \(y\) is free.

\[
\neg \forall x \exists y \exists z \forall s \forall t \; P(x, y, z, s, t)
\]
x, \(y\), \(z\), \(s\), \(t\) are all bound

\[
\forall x \; \forall y \; (P(x) \Rightarrow Q(x, f(y), z))
\]
x, \(y\) are bound while \(z\) is free.

Free and bound variables can have the same name.

\[
P(x) \Rightarrow \exists x \; Q(x)
\]
free

\[
P(x) \Rightarrow (\exists x \; Q(x)) \land R(x)
\]
binding

same free variable
**Free and Bound Variables**

A variable occurrence is *bound* in a formula if it is quantified. A variable occurrence is *free* in a formula if it is not quantified.

\[ \forall x \ Father(x, y) \Rightarrow Male(x) \]

\[ \neg \forall x \exists y \exists z \forall s \forall t \ P(x, y, z, s, t) \]

\[ \forall x \forall y \ (P(x) \Rightarrow Q(x, f(y), z)) \]

Free and bound variables can have the same name.

\[ P(x) \Rightarrow \exists x \ Q(x) \]

\[ P(x) \Rightarrow (\exists x \ Q(x)) \land R(x) \]

\[ x \text{ is bound while } y \text{ is free.} \]

\[ x, y, z, s, t \text{ are all bound} \]

\[ x, y \text{ are bound while } z \text{ is free.} \]
Nested Quantifiers

∀x∃y Student(x) ∧ Course(y) ∧ Enrolled(x, y)

∀x∃y Brother(x, y) ⇒ Sibling(x, y)

Order matters for quantifiers of different types:

∀x∃y Loves(x, y)  // Everybody (x) loves somebody (y)
∃x∀y Loves(y, x)  // There is someone (x) whom everyone (y) loves.
Nested Quantifiers

\[ \forall x \exists y \text{Student}(x) \land \text{Course}(y) \land \text{Enrolled}(x, y) \]

\[ \forall x \exists y \text{Brother}(x, y) \Rightarrow \text{Sibling}(x, y) \]

Order matters for quantifiers of different types:

\[ \forall x \exists y \text{Loves}(x, y) \quad \text{// Everybody (x) loves somebody (y)} \]

\[ \exists x \forall y \text{Loves}(y, x) \quad \text{// There is someone (x) whom everyone (y) loves.} \]

but not for those of the same type and appearing next to each other:

\[ \exists x \forall y \text{Loves}(x, y) \equiv \exists y \forall x \text{Loves}(x, y) \]

\[ \forall x \forall y (\text{Brother}(x, y) \Rightarrow \text{Sibling}(x, y)) \]

\[ \equiv \forall y \forall x (\text{Brother}(x, y) \Rightarrow \text{Sibling}(x, y)) \]
Connections Between $\forall$ and $\exists$ Through $\neg$

$$\forall x \, \neg Likes(x, Parsnips) \equiv \neg \exists x \, Likes(x, Parsnips)$$

$$\forall x \, Likes(x, Icecream) \equiv \neg \exists x \, \neg Likes(x, Icecream)$$
Connections Between $\forall$ and $\exists$ Through $\neg$

\[
\forall x \ \neg \text{Likes}(x, \text{Parsnips}) \equiv \neg \exists x \ \text{Likes}(x, \text{Parsnips})
\]

\[
\forall x \ \text{Likes}(x, \text{Icecream}) \equiv \neg \exists x \ \neg \text{Likes}(x, \text{Icecream})
\]

De Morgan’s rules still apply:

\[
\neg \forall x \ P(x) \equiv \exists x \ \neg P(x)
\]

\[
\neg \exists x \ P(x) \equiv \forall x \ \neg P(x)
\]
Connections Between ∀ and ∃ Through ¬

∀x ¬Likes(x, Parsnips)  ≡  ¬ ∃x Likes(x, Parsnips)
∀x Likes(x, Icecream)  ≡  ¬ ∃x ¬Likes(x, Icecream)

De Morgan’s rules still apply:

¬∀x P(x)  ≡  ∃x ¬P(x)
¬∃x P(x)  ≡  ∀x ¬P(x)

Move negation inward, flipping the quantifiers:

¬∀x∃y∃z∀s∀t P(x, y, z, s, t)
Connections Between $\forall$ and $\exists$ Through $\neg$

\[ \forall x \; \neg \text{Likes}(x, \text{Parsnips}) \equiv \neg \exists x \; \text{Likes}(x, \text{Parsnips}) \]
\[ \forall x \; \text{Likes}(x, \text{Icecream}) \equiv \neg \exists x \; \neg \text{Likes}(x, \text{Icecream}) \]

De Morgan’s rules still apply:

\[ \neg \forall x \; P(x) \equiv \exists x \; \neg P(x) \]
\[ \neg \exists x \; P(x) \equiv \forall x \; \neg P(x) \]

Move negation inward, flipping the quantifiers:

\[ \neg \forall x \exists y \exists z \forall s \forall t \; P(x, y, z, s, t) \equiv \exists x \neg \exists y \exists z \forall s \forall t \; P(x, y, z, s, t) \]
Connections Between \( \forall \) and \( \exists \) Through \( \neg \)

\[
\forall x \; \neg \text{Likes}(x, \text{Parsnips}) \equiv \neg \exists x \; \text{Likes}(x, \text{Parsnips})
\]

\[
\forall x \; \text{ Likes}(x, \text{Icecream}) \equiv \neg \exists x \; \neg \text{Likes}(x, \text{Icecream})
\]

De Morgan’s rules still apply:

\[
\neg \forall x \; P(x) \equiv \exists x \; \neg P(x)
\]

\[
\neg \exists x \; P(x) \equiv \forall x \; \neg P(x)
\]

Move negation inward, flipping the quantifiers:

\[
\neg \forall x \; \exists y \; \exists z \; \forall s \; \forall t \; P(x, y, z, s, t) \equiv \exists x \; \exists y \; \exists z \; \forall s \; \forall t \; P(x, y, z, s, t)
\]

\[
\equiv \exists x \; \forall y \; \exists z \; \forall s \; \forall t \; P(x, y, z, s, t)
\]
Connections Between ∀ and ∃ Through ¬

∀x ¬Likes(x, Parsnips) ≡ ¬∃x Likes(x, Parsnips)
∀x Likes(x, Icecream) ≡ ¬∃x ¬Likes(x, Icecream)

De Morgan’s rules still apply:

¬∀x P(x) ≡ ∃x ¬P(x)
¬∃x P(x) ≡ ∀x ¬P(x)

Move negation inward, flipping the quantifiers:

¬∀x∃y∃z∀s∀t P(x, y, z, s, t) ≡ ∃x¬∃y∃z∀s∀t P(x, y, z, s, t)
≡ ∃x∀y¬∃z∀s∀t P(x, y, z, s, t)
≡ ∃x∀y∀z¬∀s∀t P(x, y, z, s, t)
Connections Between $\forall$ and $\exists$ Through $\neg$

$$\forall x \neg Likes(x,\text{Parsnips}) \equiv \neg \exists x Likes(x,\text{Parsnips})$$
$$\forall x Likes(x,\text{Icecream}) \equiv \neg \exists x \neg Likes(x,\text{Icecream})$$

De Morgan’s rules still apply:

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

Move negation inward, flipping the quantifiers:

$$\neg \forall x \forall y \exists z \forall s \forall t \ P(x, y, z, s, t) \equiv \exists x \forall y \exists z \forall s \forall t \ P(x, y, z, s, t)$$
$$\equiv \exists x \forall y \forall z \neg \forall s \forall t \ P(x, y, z, s, t)$$
$$\equiv \exists x \forall y \forall z \exists s \neg \forall t \ P(x, y, z, s, t)$$
Connections Between $\forall$ and $\exists$ Through $\neg$

$\forall x \neg \text{Likes}(x, \text{Parsnips}) \equiv \neg \exists x \text{Likes}(x, \text{Parsnips})$

$\forall x \text{Likes}(x, \text{Icecream}) \equiv \neg \exists x \neg \text{Likes}(x, \text{Icecream})$

De Morgan’s rules still apply:

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

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

Move negation inward, flipping the quantifiers:

$\neg \forall x \exists y \exists z \forall s \forall t \ P(x, y, z, s, t) \equiv \exists x \exists y \exists z \forall s \forall t \ P(x, y, z, s, t)$

$\equiv \exists x \forall y \neg \exists z \forall s \forall t \ P(x, y, z, s, t)$

$\equiv \exists x \forall y \forall z \neg \forall s \forall t \ P(x, y, z, s, t)$

$\equiv \exists x \forall y \forall z \exists s \neg \forall t \ P(x, y, z, s, t)$

$\equiv \exists x \forall y \forall z \exists s \exists t \neg P(x, y, z, s, t)$
Equality

- It states that two terms refer to the same object.

\[ \text{Father}(\text{Zeus}) = \text{Cronus} \]
\[ \text{Father}(\text{Cronus}) = \text{Uranus} \]
Equality

• It states that two terms refer to the same object.

\[ Father(Zeus) = \text{Cronus} \]
\[ Father(\text{Cronus}) = \text{Uranus} \]

• The symbol can also be used to state that two terms are not the same object.
Equality

- It states that two terms refer to the same object.

\[ \text{Father}(\text{Zeus}) = \text{Cronus} \]
\[ \text{Father}(\text{Cronus}) = \text{Uranus} \]

- The symbol can also be used to state that two terms are not the same object.

// Zeus has exactly two brothers
Equality

- It states that two terms refer to the same object.

\[
\text{Father}(Zeus) = \text{Cronus}
\]
\[
\text{Father}(\text{Cronus}) = \text{Uranus}
\]

- The symbol can also be used to state that two terms are not the same object.

// Zeus has exactly two brothers

\[
\exists x, y \ \text{Brother}(x, \text{Zeus}) \land \text{Brother}(y, \text{Zeus}) \land \neg(x = y) \\
\land (\forall z \ \text{Brother}(z, \text{Zeus}) \Rightarrow (z = x) \lor (z = y))
\]
Equality

- It states that two terms refer to the same object.

\[ \text{Father}(Zeus) = \text{Cronus} \]
\[ \text{Father}(\text{Cronus}) = \text{Uranus} \]

- The symbol can also be used to state that two terms are not the same object.

// Zeus has exactly two brothers
// (x \equiv \text{Poseidon} and y \equiv \text{Hades}, or x \equiv \text{Hades} and y \equiv \text{Poseidon})

\[ \exists x, y \ \text{Brother}(x, \text{Zeus}) \land \text{Brother}(y, \text{Zeus}) \land \lnot(x = y) \]
\[ \land (\forall z \ \text{Brother}(z, \text{Zeus}) \Rightarrow (z = x) \lor (z = y)) \]
III. Model for First-Order Logic

Sentences are true with respect to a model $M$. 
III. Model for First-Order Logic

- Sentences are true with respect to a model $M$.

- The model $M$ contains objects (called *domain elements*) and interpretations of symbols.
III. Model for First-Order Logic

- Sentences are true with respect to a model $M$.
- The model $M$ contains objects (called *domain elements*) and interpretations of symbols.
  - constant symbols $\rightarrow$ objects in domain $D$
III. Model for First-Order Logic

- Sentences are true with respect to a model $M$.

- The model $M$ contains objects (called domain elements) and interpretations of symbols.
  - constant symbols $\rightarrow$ objects in domain $D$
  - predicate symbols $\rightarrow$ relations
III. Model for First-Order Logic

_sentences are true with respect to a model \( M \).

-The model \( M \) contains objects (called **domain elements**) and interpretations of symbols.
  - constant symbols \( \rightarrow \) objects in domain \( D \)
  - predicate symbols \( \rightarrow \) relations
  - function symbols \( \rightarrow \) functional relations
III. Model for First-Order Logic

Sentences are true with respect to a model $M$.

The model $M$ contains objects (called domain elements) and interpretations of symbols.

- constant symbols $\rightarrow$ objects in domain $D$
- predicate symbols $\rightarrow$ relations
- function symbols $\rightarrow$ functional relations

Each predicate $P(x_1, \ldots, x_k)$ is mapped to a relation, which is a set of $k$-tuples over $D$. 
III. Model for First-Order Logic

- Sentences are true with respect to a model $M$.

- The model $M$ contains objects (called domain elements) and interpretations of symbols.
  
  - constant symbols $\rightarrow$ objects in domain $D$
  - predicate symbols $\rightarrow$ relations
  - function symbols $\rightarrow$ functional relations

- Each predicate $P(x_1, \ldots, x_k)$ is mapped to a relation, which is a set of $k$-tuples over $D$.

- Each function $f(x_1, \ldots, x_k)$ is mapped to a function from $D^k$ to $D \cup \{o\}$, where $o$ is some invisible object.
Model Example

Model for the family relationships of the Greek gods (incomplete).

- **Father**: (Zeus, Hermes)
- **Mother**: (Hera, Ares)
- **Mother**: (Aphrodite, Harmonia)
- **Father**: (Zeus, Athena)

**Predicates**
- **Weapon**: (Zeus) // ≡ Thunderbolt
- **Weapon**: (Apollo) // ≡ BowAndArrows
- **Carry**: (Hermes) // ≡ Flute
- **Carry**: (Aphrodite) // ≡ Apple

**Domain**: $D$

- Zeus
- Ares
- Hera
- Harmonia
- Demeter
- Dionysus
- Hermes
- Apollo
- Poseidon
- Athena
- Artemis
- Persephone
- Hephaestus

**Functions**
A predicate $P(t_1, ..., t_k)$ is true if the objects referred to by the terms $t_1, ..., t_k$ are in the relation referred to by the predicate.
A predicate $P(t_1, \ldots, t_k)$ is true if the objects referred to by the terms $t_1, \ldots, t_k$ are in the relation referred to by the predicate.

$t_1 = t_2$ is true if the two terms $t_1$ and $t_2$ refer to the same object.
Truth in First-Order Logic

- A predicate $P(t_1, \ldots, t_k)$ is true if the objects referred to by the terms $t_1, \ldots, t_k$ are in the relation referred to by the predicate.

- $t_1 = t_2$ is true if the two terms $t_1$ and $t_2$ refer to the same object.

- The semantics of sentences formed with logical connectives are identical to those in propositional logic.
Truth in First-Order Logic

♦ A predicate $P(t_1, ..., t_k)$ is true if the objects referred to by the terms $t_1, ..., t_k$ are in the relation referred to by the predicate.

♦ $t_1 = t_2$ is true if the two terms $t_1$ and $t_2$ refer to the same object.

♦ The semantics of sentences formed with logical connectives are identical to those in propositional logic.

Quantifiers allow us to express properties of a collection of objects instead of enumerating them by name.

∀ (universal): “for all”

∃ (existential): “there exists”
Truths with Quantifications

- $\forall x \ P(x)$ is true in a model $M$ iff $P(x)$ is true with $x$ assuming every object in the model

$$\forall x \ Father(x, y) \Rightarrow Male(x)$$  true (in every model)

$$\forall x \ Ellipse(x) \Rightarrow Circle(x)$$  true (in every model)
Truths with Quantifications

- \( \forall x \ P(x) \) is true in a model \( M \) iff \( P(x) \) is true with \( x \) assuming every object in the model.

\[
\forall x \ Father(x, y) \Rightarrow Male(x)
\]  
\( true \) (in every model)

\[
\forall x \ Ellipse(x) \Rightarrow Circle(x)
\]  
\( true \) (in every model)

- \( \exists x \ P(x) \) is true in a model \( M \) iff \( P(x) \) is true with \( x \) assuming some object in the model.

\[
\exists x \neg Likes(x, \text{sushi})
\]  
\( true \) (in a model that includes all the people in the world)
Truths with Quantifications

- $\forall x \ P(x)$ is true in a model $M$ iff $P(x)$ is true with $x$ assuming every object in the model.

  $\forall x \ Father(x, y) \Rightarrow Male(x)$  
  $\forall x \ Ellipse(x) \Rightarrow Circle(x)$  
  \text{true (in every model)}

- $\exists x \ P(x)$ is true in a model $M$ iff $P(x)$ is true with $x$ assuming some object in the model.

  $\exists x \ \neg Likes(x, \text{sushi})$  
  $\exists x \ Mother(x, \text{Ares}) \land Mother(x, \text{Harmonia})$  
  \text{true (in a model that includes all the people in the world)}  
  \text{false (in the model of Greek mythology)}
IV. Knowledge Engineering

A field of AI dedicated to representing information about the world in a form that can be utilized by a computer to solve complex tasks such as:

- medical diagnosis
- dialog in a natural language
- etc.

♦ Knowledge representation (logical rules, semantic nets, frames, etc.)
♦ Automated reasoning (inference engines, theorem provers, etc.)
IV. Knowledge Engineering

A field of AI dedicated to representing information about the world in a form that can be utilized by a computer to solve complex tasks such as:

- medical diagnosis
- dialog in a natural language
- etc.

- Knowledge representation (logical rules, semantic nets, frames, etc.)
- Automated reasoning (inference engines, theorem provers, etc.)

A *domain* is some part of the world about which we wish to express some knowledge.
Assertions and Queries in FOL

- Add sentences, called assertions, to a KB using TELL.

  TELL(KB, Likes(John, Icecream))
  TELL(KB, Father(Zeus, Athena))
  TELL(KB, ∀x∃y Brother(x, y) ⇒ Sibling(x, y))
Add sentences, called **assertions**, to a KB using TELL.

\[
\text{TELL}(KB, \text{Likes}(John, \text{Icecream})) \\
\text{TELL}(KB, \text{Father}(Zeus, Athena)) \\
\text{TELL}(KB, \forall x \exists y \text{ Brother}(x, y) \Rightarrow \text{Sibling}(x, y))
\]

Ask the KB questions using ASK.

\[
\text{ASK}(KB, \text{Likes}(John, \text{Icecream}))
\]
Assertions and Queries in FOL

- Add sentences, called **assertions**, to a KB using **TELL**.

\[
\text{TELL}(KB, Likes(\text{John, Icecream}))
\]
\[
\text{TELL}(KB, \text{Father}(\text{Zeus, Athena}))
\]
\[
\text{TELL}(KB, \forall x \exists y \ Brother(x, y) \Rightarrow \text{Sibling}(x, y))
\]

- Ask the KB questions using **ASK**.

\[
\text{ASK}(KB, Likes(\text{John, Icecream}))
\]

**Query**: question asked
Assertions and Queries in FOL

- Add sentences, called assertions, to a KB using TELL.

  \[
  \text{TELL}(KB, Likes(John, Icecream))
  \]
  \[
  \text{TELL}(KB, Father(Zeus, Athena))
  \]
  \[
  \text{TELL}(KB, \forall x \exists y \text{ Brother}(x, y) \Rightarrow \text{Sibling}(x, y))
  \]

- Ask the KB questions using ASK.

  \[
  \text{ASK}(KB, Likes(John, Icecream))
  \]
  \[
  \underline{Query}: \text{question asked}
  \]

Any query is entailed by the KB should be answered affirmatively.
Suppose another KB has the following predicates:

Bird(Swan), Bird(Crane), Bird(Parrot),

- Quantified query
  \text{ASK}(KB, \exists x \text{ Bird}(x))
Suppose another KB has the following predicates:

\[ \text{Bird(Swan)}, \text{Bird(Crane)}, \text{Bird(Parrot)}, \]

- Quantified query

\[ \text{Ask}(KB, \exists x \text{ Bird}(x)) \quad \text{returns} \quad true \]
Suppose another KB has the following predicates:

\( Bird(Swan), Bird(Crane), Bird(Parrot), \)

- Quantified query
  \[ \text{ASK}(KB, \exists x \: Bird(x)) \] returns \text{true} \n
- To know what values of \( x \) make the sentence true
  \[ \text{ASKVARS}(KB, Bird(x)) \]
Suppose another KB has the following predicates:

\[ Bird(Swan), Bird(Crane), Bird(Parrot), \]

- **Quantified query**
  \[ \text{ASK}(KB, \exists x \ Bird(x)) \]
  returns \text{true}

- To know what values of \( x \) make the sentence true
  \[ \text{ASK\_VARS}(KB, Bird(x)) \]
  The query returns
  \[ \{x / Swan\}, \{x / Crane\}, \text{and} \{x / Parrot\} \]
Substitution

Suppose another KB has the following predicates:

\[ \text{Bird(Swan)}, \text{Bird(Crane)}, \text{Bird(Parrot)}, \]

- Quantified query
  \[ \text{ASK}(\text{KB}, \exists x \text{ Bird}(x)) \] returns true

- To know what values of \( x \) make the sentence true
  \[ \text{ASKVARS}(\text{KB}, \text{Bird}(x)) \]

The query returns

\{x / Swan\}, \{x / Crane\}, and \{x / Parrot\}

substitution or binding list
The Kinship Domain

Kinship relations are represented by binary predicates.

// One’s mother is the person’s female parent.
\[ \forall m, c \ Mother(c) = m \iff Female(m) \land Parent(m, c) \]
The Kinship Domain

Kinship relations are represented by binary predicates.

// One’s mother is the person’s female parent.
∀m, c Mother(c) = m ⇔ Female(m) ∧ Parent(m, c)

// One’s husband is the person’s male spouse.
∀w, h Husband(h, w) ⇔ Male(h) ∧ Spouse(h, w)
Kinship relations are represented by binary predicates.

// One’s mother is the person’s female parent.
∀m, c \( \text{Mother}(c) = m \) ⇔ \( \text{Female}(m) \land \text{Parent}(m, c) \)

// One’s husband is the person’s male spouse.
∀w, h \( \text{Husband}(h, w) \) ⇔ \( \text{Male}(h) \land \text{Spouse}(h, w) \)

// Parent and child are inverse relations.
∀p, c \( \text{Parent}(p, c) \) ⇔ \( \text{Child}(c, p) \)
The Kinship Domain

Kinship relations are represented by binary predicates.

// One's mother is the person's female parent.
∀m, c  Mother(c) = m ⇔ Female(m) ∧ Parent(m, c)

// One's husband is the person's male spouse.
∀w, h  Husband(h, w) ⇔ Male(h) ∧ Spouse(h, w)

// Parent and child are inverse relations.
∀p, c  Parent(p, c) ⇔ Child(c, p)

// A grand parent is a parent of one's parent
∀g, c  GrandParent(g, c) ⇔ ∃p (Parent(g, p) ∧ Parent(p, c))
The Kinship Domain

Kinship relations are represented by binary predicates.

\[
\forall m, c \quad \text{Mother}(c) = m \iff \text{Female}(m) \land \text{Parent}(m, c)
\]

// One’s mother is the person’s female parent.

\[
\forall w, h \quad \text{Husband}(h, w) \iff \text{Male}(h) \land \text{Spouse}(h, w)
\]

// One’s husband is the person’s male spouse.

\[
\forall p, c \quad \text{Parent}(p, c) \iff \text{Child}(c, p)
\]

// Parent and child are inverse relations.

\[
\forall g, c \quad \text{GrandParent}(g, c) \iff \exists p \ (\text{Parent}(g, p) \land \text{Parent}(p, c))
\]

// A grand parent is a parent of one’s parent

\[
\forall x, s \quad \text{Sibling}(x, s) \iff x \neq s \land \exists p \ (\text{Parent}(p, x) \land \text{Parent}(p, s))
\]

// A sibling is another child of one’s parent
The Kinship Domain

Kinship relations are represented by binary predicates.

- **Axioms**
  - One’s mother is the person’s female parent.
    \[
    \forall m, c \quad Mother(c) = m \iff Female(m) \land Parent(m, c)
    \]
  - One’s husband is the person’s male spouse.
    \[
    \forall w, h \quad Husband(h, w) \iff Male(h) \land Spouse(h, w)
    \]
  - Parent and child are inverse relations.
    \[
    \forall p, c \quad Parent(p, c) \iff Child(c, p)
    \]
  - A grand parent is a parent of one’s parent
    \[
    \forall g, c \quad GrandParent(g, c) \iff \exists p \ (Parent(g, p) \land Parent(p, c))
    \]
  - A sibling is another child of one’s parent
    \[
    \forall x, s \quad Sibling(x, s) \iff x \neq s \land \exists p \ (Parent(p, x) \land Parent(p, s))
    \]
The Kinship Domain

Kinship relations are represented by binary predicates.

\[ \forall m, c \text{ Mother}(c) = m \iff \text{Female}(m) \land \text{Parent}(m, c) \]

\[ \forall w, h \text{ Husband}(h, w) \iff \text{Male}(h) \land \text{Spouse}(h, w) \]

\[ \forall p, c \text{ Parent}(p, c) \iff \text{Child}(c, p) \]

\[ \forall g, c \text{ GrandParent}(g, c) \iff \exists p (\text{Parent}(g, p) \land \text{Parent}(p, c)) \]

\[ \forall x, s \text{ Sibling}(x, s) \iff x \neq s \land \exists p (\text{Parent}(p, x) \land \text{Parent}(p, s)) \]

These are definitions in the form of \( \forall x, y P(x, y) \iff \ldots \) and built upon a basic set of predicates \text{Child}, \text{Male}, \text{Female}, etc.
Axioms and Theorems

- **Axioms** in a domain are logical sentences that are taken to be true without being derived.

- **Theorems** are logical sentences entailed by axioms.
Axioms and Theorems

- **Axioms** in a domain are logical sentences that are taken to be true without being derived.

- **Theorems** are logical sentences entailed by axioms.

\[ \forall x, y \, \text{Sibling}(x, y) \iff \text{Sibling}(y, x) \]

// entailed by

// \[ \forall x, s \, \text{Sibling}(x, s) \iff x \neq s \land \exists p \left( \text{Parent}(p, x) \land \text{Parent}(p, s) \right) \]
Axioms and Theorems

- **Axioms** in a domain are logical sentences that are taken to be true without being derived.

- **Theorems** are logical sentences entailed by axioms.

\[ \forall x, y \ Sibling(x, y) \iff Sibling(y, x) \]

// entailed by
// \forall x, s \ Sibling(x, s) \iff x \neq s \land \exists p \ (Parent(p, x) \land Parent(p, s))

\[ \text{Ask}(KB, \forall x, y \ Sibling(x, y) \iff Sibling(y, x)) \text{ should return } \text{true}. \]
Axioms and Theorems

أخلاقات ونظرية

♦ **Axioms** in a domain are logical sentences that are taken to be true without being derived.

♦ **Theorems** are logical sentences entailed by axioms.

\[ \forall x, y \text{ Sibling}(x, y) \leftrightarrow \text{Sibling}(y, x) \]

// entailed by

\[ \forall x, s \text{ Sibling}(x, s) \iff x \neq s \land \exists p (\text{Parent}(p, x) \land \text{Parent}(p, s)) \]

\[ \text{Ask}(KB, \forall x, y \text{ Sibling}(x, y) \leftrightarrow \text{Sibling}(y, x)) \text{ should return } \text{true}. \]

♦ Some axioms are not definitions.

\[ \forall x \text{ Person}(x) \leftrightarrow \cdots \]  // no clear way to define

\[ \forall x, y \text{ Sibling}(x, y) \leftrightarrow \text{Sibling}(y, x) \]
Axioms and Theorems

- **Axioms** in a domain are logical sentences that are taken to be true without being derived.

- **Theorems** are logical sentences entailed by axioms.

\[ \forall x, y \ Sibling(x, y) \Leftrightarrow Sibling(y, x) \]
// entailed by
// \[ \forall x, s \ Sibling(x, s) \Leftrightarrow x \neq s \land \exists p \ (\text{Parent}(p, x) \land \text{Parent}(p, s)) \]

\text{Ask}(KB, \forall x, y \ Sibling(x, y) \Leftrightarrow Sibling(y, x)) \text{ should return } true.

- Some axioms are not definitions.

\[ \forall x \ \text{Person}(x) \Leftrightarrow \cdots \] // no clear way to define

- Some predicates have no complete definitions.
Axioms and Theorems

Axioms in a domain are logical sentences that are taken to be true without being derived.

Theorems are logical sentences entailed by axioms.

\[ \forall x, y \ Sibling(x, y) \iff Sibling(y, x) \]
// entailed by
// \[ \forall x, s \ Sibling(x, s) \iff x \neq s \land \exists p \ (Parent(p, x) \land Parent(p, s)) \]

\text{Ask}(KB, \forall x, y \ Sibling(x, y) \iff Sibling(y, x)) \text{ should return } \text{true.}

Some axioms are not definitions.

\[ \forall x \ Person(x) \iff \ldots \] // no clear way to define

Some predicates have no complete definitions.

\[ \forall x \ Person(x) \implies \ldots \] // partial specification of
\[ \forall x \ldots \implies Person(x) \] // properties