The Foundations: Logic and Proofs

Chapter 1, Part II: Predicate Logic

With Question/Answer Animations

Chapter Summary

- Propositional Logic
  - The Language of Propositions
  - Applications
  - Logical Equivalences
- Predicate Logic
  - The Language of Quantifiers
  - Logical Equivalences
  - Nested Quantifiers
- Proofs
  - Rules of Inference
  - Proof Methods
  - Proof Strategy

Summary

- Predicate Logic (First-Order Logic (FOL), Predicate Calculus)
  - The Language of Quantifiers
  - Logical Equivalences
  - Nested Quantifiers
  - Translation from Predicate Logic to English
  - Translation from English to Predicate Logic

Predicates and Quantifiers

Section 1.4

Section Summary

- Predicates
- Variables
- Quantifiers
  - Universal Quantifier
  - Existential Quantifier
- Negating Quantifiers
  - De Morgan's Laws for Quantifiers
- Translating English to Logic
- Logic Programming (*optional*)

Propositional Logic Not Enough

If we have:

"All men are mortal."

"Socrates is a man."

- Does it follow that "Socrates is mortal?"
- Can't be represented in propositional logic. Need a language that talks about objects, their properties, and their relations.
- Later we'll see how to draw inferences.

Introducing Predicate Logic

Predicate logic uses the following new features:

- Variables: x, y, z
- Predicates: $P(x)$, $M(x)$
- Quantifiers (*to be covered in a few slides*):

- *Propositional functions* are a generalization of propositions.
  - They contain variables and a predicate, e.g., $P(x)$
  - Variables can be replaced by elements from their *domain*.

Propositional Functions

Propositional functions become propositions (and have truth values) when variables are replaced by values from *domain* (or *bound* by a quantifier, as we will see later).

- The statement *P(x)* is said to be the value of the propositional function *P* at *x*.
- For example, let *P(x)* denote "$x > 0$" and the domain be the integers. Then:

  P(-3) is false.

  P(0) is false.

  P(3) is true.
- Often domain is denoted by *U*.
- So in this example *U* is Z = integers.

Examples of Propositional Functions

- Let "$x + y = z$" be denoted by $R(x, y, z)$ and U (for all three variables) be the integers. Find these truth values:

  R(2,-1,5)

  Solution: F

  R(3,4,7)

  Solution: T

  R(x, 3, z)

  Solution: Not a Proposition

- Now let "$x - y = z$" be denoted by $Q(x, y, z)$, with U as the integers. Find these truth values:

  Q(2,-1,3)

  Solution: T

  Q(3,4,7)

  Solution: F

  Q(x, 3, z)

  Solution: Not a Proposition

Compound Expressions

Connectives carry over to predicate logic.

- If $P(x)$ denotes "$x > 0$," find these truth values:

  $P(3) \vee P(-1)$ **T**

  $P(3) \wedge P(-1)$ **F**

  $P(3) \rightarrow P(-1)$ **F**

  $P(3) \rightarrow \neg P(-1)$ **T**

- Expressions with variables are not propositions and therefore do not have truth values. For example,

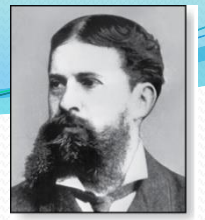
  $P(3) \wedge P(y)$

  $P(x) \rightarrow P(y)$

- When used with quantifiers (introduced next), propositional functions become propositions.

Quantifiers

Charles Peirce (1839-1914)



We need *quantifiers* to express the meaning of English words including *all* and *some*:

- "All men are Mortal."
- "Some cats do not have fur."

- The two most important quantifiers are:
  - *Universal Quantifier,* "For all," symbol: $\forall$
  - *Existential Quantifier*, "There exists," symbol: $\exists$
- We write as in $\forall x\, P(x)$ and $\exists x\, P(x)$.
  - $\forall x\, P(x)$ asserts $P(x)$ is true for <u>every</u> x in the *domain*.
  - $\exists x\, P(x)$ asserts $P(x)$ is true for <u>some</u> x in the *domain*.
- The quantifiers **bind** the variable x in these expressions.

Universal Quantifier

$\forall x\, P(x)$: "For all x, P(x)" or "For every x, P(x)"

Examples:

1) If $P(x)$ denotes "$x > 0$" and $U = \mathrm{Z}$ (integers), then $\forall x\, P(x)$ is false.
2) If $P(x)$ denotes "$x > 0$" and $U = \mathrm{Z}^+$ (positive integers), then $\forall x\, P(x)$ is true.
3) If $P(x)$ denotes "x is even" and $U = \mathrm{Z}$, then $\forall\, x\, P(x)$ is false.

Existential Quantifier

$\exists x\, P(x)$: "For some x, P(x)", "There is an x s.t. P(x)," or "For at least one x, P(x)."

Examples:

1. If $P(x)$ denotes "$x > 0$" and $U = Z$, then $\exists x\, P(x) \equiv$ T.
2. If $P(x)$ denotes "$x < 0$" and $U = Z^+$, then $\exists x\, P(x) \equiv$ F.
3. If $P(x)$ denotes "x is even" & $U =$ Z, then $\exists x\, P(x) \equiv$ T.

Uniqueness Quantifier

$\exists!x\, P(x)$ means that $P(x)$ is true for <u>one and only one</u> x in the universe of discourse.

- This is commonly expressed in English in the following equivalent ways:
  - "There is a unique x such that $P(x)$."
  - "There is one and only one x such that $P(x)$"
- Examples (U = Z):

1. If $P(x)$ denotes "$x + 1 = 0$", then $\exists!x\, P(x) \equiv$ T.
2. If $P(x)$ denotes "$x > 0$", then $\exists!x\, P(x) \equiv$ F.

Thinking about Quantifiers

When the domain of discourse is finite, we can think of quantification as looping through the elements of the domain.

- To evaluate $\forall x\, P(x)$ loop through all x in the domain.
  - If at every step $P(x)$ is true, then $\forall x\, P(x) \equiv T$.
  - If at a step $P(x)$ is false, then $\forall x\, P(x) \equiv F$ and loop terminates.
- To evaluate $\exists x\, P(x)$ loop through all x in the domain.
  - If at a step $P(x)$ is true, then $\exists x\, P(x) \equiv T$ and loop terminates.
  - If loop ends without finding x for which $P(x) = T$, then $\exists x\, P(x) \equiv F$
- Even if domains are infinite, we can still think of quantifiers in this fashion, but the loops will not terminate in some cases.

More Examples

The truth value of $\exists x\, P(x)$ and $\forall x\, P(x)$ depend on both the propositional function $P(x)$ and on the domain U.

- **Examples**:
  1. If U is Z^+ (positive integers) and $P(x)$ is the statement "$x < 2$", then $\exists x\, P(x)$ is true, but $\forall x\, P(x)$ is false.
  2. If U is Z^- (negative integers) and $P(x)$ is the statement "$x < 2$", then both $\exists x\, P(x)$ and $\forall x\, P(x)$ are true.
  3. If $U = \{3, 4, 5\}$ and $P(x)$ is the statement "$x > 2$", then both $\exists x\, P(x)$ and $\forall x\, P(x)$ are true.
  4. If $U = \{3, 4, 5\}$ and $P(x)$ is the statement "$x < 2$", then both $\exists x\, P(x)$ and $\forall x\, P(x)$ are false.

Precedence of Quantifiers

The quantifiers $\forall$ and $\exists$ have higher precedence than all the logical operators.

- For example, $\forall x\, P(x) \vee Q(x)$ means $(\forall x\, P(x)) \vee Q(x)$
- $\forall x\, (P(x) \vee Q(x))$ means something different.
- Unfortunately, often people write $\forall x\, P(x) \vee Q(x)$ when they mean $\forall x\, (P(x) \vee Q(x))$.

Translating from English to Logic

Example 1: Translate into predicate logic:

"Every student in this class has taken a course in Java."

Solution: First decide on the domain *U*.

Solution 1: If *U* is all students in class, define a propositional function J(*x*) denoting "x has taken a course in Java" and translate as $\forall x\, J(x)$.

Solution 2: But if *U* is all people, also define a propositional function S(x) denoting "x is a student in this class" and translate as $\forall x\, (S(x) \rightarrow J(x))$.

Exercise: $\forall x\, (S(x) \wedge J(x))$ is not correct. What does it mean?

Translating from English to Logic

Example 2: Translate into predicate logic:

"Some student in this class has taken a course in Java."

Solution: First decide on the domain U.

Solution 1: If U is all students in class, translate as $\exists x\, J(x)$

Solution 2: If U is all people, then translate as $\exists x\, (S(x) \wedge J(x))$

Exercise: $\exists x\, (S(x) \rightarrow J(x))$ is not correct. What does it mean?

Rewrite as a statement without the implication.

Returning to the Socrates Example

- Introduce propositional functions:
  - *Man(x)* denoting "*x* is a man"
  - *Mortal(x)* denoting "*x* is mortal."
- Specify the domain as all people.
- The two premises are: $\forall x(Man(x) \rightarrow Mortal(x))$

  $Man(Socrates)$
- The conclusion is: $Mortal(Socrates)$

Equivalences in Predicate Logic

- Statements involving predicates and quantifiers are *logically equivalent* if and only if they have the same truth value:
  - for every predicate substituted into these statements;
  - for every domain for the variables in the expressions.

(Recall: $S \equiv T$ indicates that S and T are logically equivalent.)

- **Example**: $\forall x \neg\neg S(x) \equiv \forall x\, S(x)$

Quantifiers can be replaced with Conjunctions and Disjunctions

If the domain is finite, e.g., $U = \{1, 2, 3\}$:

- a universally quantified proposition ≡ conjunction of propositions $\forall x P(x) \equiv P(1) \wedge P(2) \wedge P(3)$
- an existentially quantified proposition ≡ disjunction of propositions $\exists x P(x) \equiv P(1) \vee P(2) \vee P(3)$

Even if domain is infinite, can still think of quantifiers in this fashion, but the expressions will be infinitely long.

Negating Quantified Expressions

Consider: “Every student in class has taken a course in Java.”

Symbolically: $\forall x\, J(x)$ where:

- $J(x)$ is “x has taken course in Java”;
- domain is "students in class".

Negating the original statement gives:

“It is not the case that every student in class has taken Java.”

- i.e., “There is a student in class who has not taken Java.”
- Symbolically: $\neg \forall x\, J(x) \equiv \exists x\, \neg J(x)$

Negating Quantified Expressions (*cont.*)

Now Consider :

"There is a student in this class who has taken a course in Java."

i.e., $\exists x\, J(x)$ where $J(x)$ is "x has taken a course in Java."

- Negating the original statement gives:

"It is not the case that there is a student in class who has taken Java."

- i.e., "Every student in this class has not taken Java"
- Symbolically: $\neg\exists x\, J(x) \equiv \forall x\, \neg J(x)$
- The results for last 2 slides are summarized in the next slide.

De Morgan's Laws for (negating) Quantifiers

TABLE 2 De Morgan's Laws for Quantifiers.

Negation	*Equivalent Statement*	*When Is Negation True?*	*When False?*
$\neg\exists x\, P(x)$	$\forall x \neg P(x)$	For every x, $P(x)$ is false.	There is an x for which $P(x)$ is true.
$\neg\forall x\, P(x)$	$\exists x \neg P(x)$	There is an x for which $P(x)$ is false.	$P(x)$ is true for every x.

- Upshot from the table:

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

Translation from English to Logic

Examples (Let U be all people):

1. "Some student in this class has visited Mexico."

   Solution: Let $M(x)$ denote "x has visited Mexico" and $S(x)$ denote "x is a student in this class".

   $$\exists x \ (S(x) \wedge M(x))$$

2. "Every student in class has visited Canada or Mexico."

   Solution: Add $C(x)$ denoting "x has visited Canada."

   $$\forall x \ (S(x) \rightarrow (M(x) \vee C(x)))$$

Some Fun with Translating

U = {fleegles, snurds, thingamabobs}

$F(x)$: x is a fleegle

$S(x)$: x is a snurd

$T(x)$: x is a thingamabob

- "Everything is a fleegle".

  Solution: $\forall x\, F(x)$

- "Nothing is a snurd."

  Solution: $\neg\exists x\, S(x)$ What is this equivalent to?

  Solution: $\forall x \neg S(x)$

Some Fun with Translating (cont)

U = {fleegles, snurds, thingamabobs}

F(x): *x* is a fleegle

S(x): *x* is a snurd

T(x): *x* is a thingamabob

- "All fleegles are snurds."

Solution: $\forall x\,(F(x) \rightarrow S(x))$

- "Some fleegles are thingamabobs."

Solution: $\exists x\,(F(x) \wedge T(x))$

Some Fun with Translating (cont)

U = {fleegles, snurds, thingamabobs}

$F(x)$: x is a fleegle

$S(x)$: x is a snurd

$T(x)$: x is a thingamabob

- "No snurd is a thingamabob."

Solution: $\neg\exists x\,(S(x) \wedge T(x))$ What is this equivalent to?

Solution: $\forall x\,(\neg S(x) \vee \neg T(x))$

- "If any fleegle is a snurd then it is also a thingamabob."

Solution: $\forall x\,((F(x) \wedge S(x)) \rightarrow T(x))$

System Specification Examples

Translate into predicate logic:

1. "Every mail message larger than 1 megabyte will be compressed."

Decide on predicates & domains (left implicit here) for variables:

- Let $L(m, y)$ be "Mail message m is larger than y megabytes."
- Let $C(m)$ denote "Mail message m will be compressed."

$$\forall m(L(m, 1) \rightarrow C(m))$$

2. "If a user is active, at least one network link will be available."

- Let $A(u)$ represent "User u is active."
- Let $S(n, x)$ represent "Network link n is state x.

$$\exists u\, A(u) \rightarrow \exists n\, S(n, available)$$

Charles Dodgson
(AKA Lewis Caroll)
(1832-1898)

Lewis Carroll Example

1. "All lions are fierce."
2. "Some lions do not drink coffee."
3. "Some fierce creatures do not drink coffee."

The first two are *premises* and the third is the *conclusion*.

Let P(x), Q(x), R(x) be "x is a lion", "x is fierce", "x drinks coffee".

1. $\forall x\,(P(x) \rightarrow Q(x))$
2. $\exists x\,(P(x) \wedge \neg R(x))$
3. $\exists x\,(Q(x) \wedge \neg R(x))$

Later we will show how the conclusion follows from the premises.

Some Predicate Calculus (*optional*)

- An assertion involving predicates and quantifiers is *valid* if it is true
  - for all domains
  - every propositional function substituted for the predicates in the assertion.

  Example: $\forall x \neg S(x) \leftrightarrow \neg \exists x S(x)$
- An assertion involving predicates is *satisfiable* if it is true
  - for some domains
  - some propositional functions that can be substituted for the predicates in the assertion.

  Otherwise it is *unsatisfiable*.

  Example: $\forall x(F(x) \leftrightarrow T(x))$ not valid but satisfiable

  Example: $\forall x(F(x) \wedge \neg F(x))$ unsatisfiable

More Predicate Calculus (*optional*)

- The *scope* of a quantifier is the part of an assertion in which variables are bound by the quantifier.

  Example: $\forall x(F(x) \vee S(x))$ *x* has wide scope

  Example: $\forall x(F(x)) \vee \forall y(S(y))$ *x* has narrow scope

Logic Programming (optional)

- Prolog (from *Pro*gramming in *Log*ic) is a programming language developed in the 1970s by researchers in artificial intelligence (AI).
- Prolog programs include *Prolog facts* and *Prolog rules*.
- As an example of a set of Prolog facts consider the following:

```
instructor(chan, math273).
instructor(patel, ee222).
instructor(grossman, cs301).
enrolled(kevin, math273).
enrolled(juana, ee222).
enrolled(juana, cs301).
enrolled(kiko, math273).
enrolled(kiko, cs301).
```

- Here the predicates *instructor(p,c)* and *enrolled(s,c)* represent that professor *p* is the instructor of course *c* and that student *s* is enrolled in course *c*.

Logic Programming (cont)

- In Prolog, names beginning with an uppercase letter are variables.
- If we have a predicate *teaches(p,s)* representing "professor *p* teaches student *s*," we can write the rule:

```
teaches(P,S) :- instructor(P,C), enrolled(S,C).
```

- This Prolog rule can be viewed as equivalent to the following statement in logic (using our conventions for logical statements).

$$\forall p\ \forall c\ \forall s(I(p,c) \wedge E(s,c)) \rightarrow T(p,s))$$

Logic Programming (cont)

- Prolog programs are loaded into a *Prolog interpreter*. The interpreter receives *queries* and returns answers using the Prolog program.
- For example, using our program, the following query may be given:

```
?enrolled(kevin,math273).
```

- Prolog produces the response:

```
yes
```

- Note that the `?` is the prompt given by the Prolog interpreter indicating that it is ready to receive a query.

Logic Programming (cont)

- The query:

```
?enrolled(X,math273).
```

produces the response:

```
X = kevin;
X = kiko;
no
```

- The query:

```
?teaches(X,juana).
```

produces the response:

```
X = patel;
X = grossman;
no
```

The Prolog interpreter tries to find an instantiation for X. It does so and returns `X = kevin.` Then the user types the `;` indicating a request for another answer. When Prolog is unable to find another answer it returns `no`.

Logic Programming (cont)

- The query:

```
?teaches(chan,X).
```

  produces the response:

```
X = kevin;
X = kiko;
no
```

- A number of very good Prolog texts are available. *Learn Prolog Now!* is one such text with a free online version at http://www.learnprolognow.org/
- There is much more to Prolog and to the entire field of logic programming.