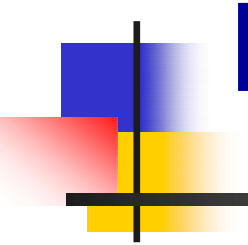
Discrete Mathematics and Applications

Dept. Information Technology,
CBIT



Lecture 1

Course Overview

Chapter 1. The Foundations

1.1 Propositional Logic

Resources

- Textbook: *Discrete Mathematics and Its Applications* (*6th Edition*), by Kenneth H. Rosen, McGraw-Hill
- Web site:
  - 1. https://onlinecourses.nptel.ac.in/noc18_cs53/
  - 2. https://www.coursera.org/learn/discrete-mathematics

What is Mathematics, really?

- It's *not* just about numbers!
- Mathematics is *much* more than that:

> Mathematics is, most generally, the study of any and all *absolutely certain* truths about any and all *perfectly well-defined* concepts.

- These concepts can be *about* numbers, symbols, objects, images, sounds, *anything*!
- It is a way to interpret the world around you.

So, what's *this* class about?

What are "discrete structures" anyway?

- "***Discrete***" - Composed of distinct, separable parts. (Opposite of *continuous*.)

discrete:*continuous* :: *digital*:*analog*

- "***Structures***" - Objects built up from simpler objects according to some definite pattern.
- "***Discrete Mathematics***" - The study of discrete, mathematical (i.e. well-defined conceptual) objects and structures.

Discrete Objects/Concepts and Structures We Study

- DM PART I
  - Propositions
  - Predicates
  - Proofs
  - Sets
  - Functions
  - Algorithms
  - Integers
  - Summations
  - Sequences
  - Strings
  - Permutations
  - Combinations
  - Probability
- DM PART II
  - Relations
  - Graphs
  - Trees
  - Boolean Functions / Logic Circuits

Why to Study Discrete Math?

- The basis of all of digital information processing is: *Discrete manipulations of discrete structures represented in memory.*
- It's the basic language and conceptual foundation for all of computer science.
- Discrete math concepts are also widely used throughout math, science, engineering, economics, biology, *etc.*, …

Why to Study Discrete Math?

- To learn a set of Mathematical facts and how to apply them.
- How to think logically and mathematically.
- A generally useful tool for rational thought!
- Discrete Mathematics: Is a part of mathematics to study of discrete objects.

Problems solved

- How many ways are there to choose a valid password on a computer system?
- What is the probability of winning a lottery?
- Is there a link between two computers in a network?
- How can I identify spam e-mail messages?
- How can I encrypt a message so that no unintended recipient can read it?
- What is the shortest path between two cities using a transportation system?

Uses of Discrete Math

- Discrete Mathematics is used whenever objects are counted, when relationships between finite sets are studied.
- Develops ability to understand and create mathematical arguments.
- Discrete mathematics is the gateway to more advanced courses in all parts of the mathematical sciences.

Uses for Discrete Math in Computer Science

- Advanced algorithms & data structures
- Programming language compilers & interpreters
- Computer networks
- Operating systems
- Computer architecture
- Database management systems
- Cryptography
- Error correction codes
- Graphics & animation algorithms, game engines, *etc.*…
- *i.e.*, the whole field!

LOGIC

- Rules of logic specify meaning of mathematical statements.
- logic has practical applications to design computing machines.
- Proof –makes a correct mathematical argument.
- Theorem- mathematical statement is True.
- To learn a mathematical topic, we need to construct mathematical argument on that topic.
- Proofs are used to verfiy that computer programs and algorithms produce correct result.

1.1 Propositional Logic

- Logic
  - Study of reasoning.
  - Specifically concerned with whether reasoning is correct.
  - Focuses on the relationship among statements, not on the content of any particular statement.
  - Gives precise meaning to mathematical statements.
- ***Propositional Logic*** is the logic that deals with statements (propositions) and compound statements built from simpler statements using so-called *Boolean connectives.*
- Some applications in computer science:
  - Design of digital electronic circuits.
  - Expressing conditions in programs.
  - Queries to databases & search engines.

Definition of a *Proposition*

Definition: A ***proposition*** (denoted *p*, *q*, *r*, …) is simply:

- a *statement* (*i.e.*, a declarative sentence which declares a fact)
  - *with some definite meaning*, (not vague or ambiguous)
- having a *truth value* that's either *true* (**T**) or *false* (**F**)
  - it is **never** both, neither, or somewhere "in between!"
    - However, you might not *know* the actual truth value,
    - and, the truth value might *depend* on the situation or context.
- Later, we will study *probability theory,* in which we assign *degrees of certainty* ("between" **T** and **F**) to propositions.
  - But for now: think True/False only! (or in terms of **1** and **0**)

Examples of Propositions

- It is raining. (In a given situation)
- Beijing is the capital of China. (T)
- 2 + 2 = 5. (F)
- 1 + 2 = 3. (T)
- A fact-based declaration is a proposition, even if no one knows whether it is true
  - 11213 is prime.
  - There exists an odd perfect number.

Examples of Non-Proposition

The following are **NOT** propositions:

- Who's there? (interrogative, question)
- Just do it! (imperative, command)
- jhhhggggg. (meaningless interjection)
- Yeah, I sorta dunno, whatever... (vague)
- 1 + 2 (expression with a non-true/false value)
- x + 2 = 5 (declaration about semantic tokens of non-constant value)

Truth Tables

- An *operator* or *connective* combines one or more *operand* expressions into a larger expression. (*e.g.*, "+" in numeric expressions.)
- ***Unary*** operators take *one* operand (*e.g.,* −3); ***Binary*** operators take *two* operands (*e.g.* 3×4).
- ***Propositional*** or ***Boolean operators*** operate on propositions (or their truth values) instead of on numbers.
- The ***Boolean domain*** is the set {T, F}. Either of its elements is called a ***Boolean value***.

An n-tuple $(p_1,\ldots,p_n)$ of Boolean values is called a ***Boolean n-tuple***.

- An n-operand truth table is a table that assigns a Boolean value to the set of all Boolean n-tuples.

Some Popular Boolean Operators

Formal Name	Nickname	Arity	Symbol
Negation operator	NOT	Unary	$\neg$
Conjunction operator	AND	Binary	$\wedge$
Disjunction operator	OR	Binary	$\vee$
Exclusive-OR operator	XOR	Binary	$\oplus$
Implication operator	IMPLIES	Binary	$\rightarrow$
Biconditional operator	IFF	Binary	$\leftrightarrow$

The Negation Operator

- The unary ***negation*** *operator* "¬" (*NOT*) transforms a proposition into its logical *negation*.
- *E.g.* If *p* = "I have brown hair."

then ¬*p* = "It is not the case that I have brown hair" or "I do **not** have brown hair."

- The *truth table* for NOT:

p	$\neg p$
T	F
F	T

Operand column

Result column

The Conjunction Operator

- The binary ***conjunction*** *operator* "$\wedge$" (*AND*) combines two propositions to form their logical *conjunction*.

E.g. If p = "Rebecca's PC has more than 16 GB free hard disk space."

and

q = "The processor in Rebecca's PC runs faster than 1 GHz."

then, $p \wedge q$ = "Rebecca's PC has more than 16 GB free hard disk space, and the processor in Rebecca's PC runs faster than 1 GHz."

"Rebecca's PC has more than 16 GB free hard disk space, and its processor runs faster than 1 GHz."

Conjunction Truth Table

Operand columns

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

- Note that a conjunction $p_1 \wedge p_2 \wedge \ldots \wedge p_n$ of n propositions will have 2^n rows in its truth table

The Disjunction Operator

- The binary ***disjunction*** *operator* "$\vee$" (*OR*) combines two propositions to form their logical *disjunction*.

- *E.g.* If *p* = "My car has a bad engine." and *q* = "My car has a bad carburetor."

then, $p \vee q$ = "My car has a bad engine, **or** my car has a bad carburetor."

Meaning is like "and/or" in informal English.

Disjunction Truth Table

p	q	$p \vee q$
T	T	T
T	F	**T**
F	T	**T**
F	F	F

Note difference from AND

- Note that $p \vee q$ means that p is true, or q is true, **or both** are true!
- So, this operation is also called ***inclusive or***, because it **includes** the possibility that both p and q are true.

The Exclusive-Or Operator

- The binary ***exclusive-or*** *operator* "⊕" (*XOR*) combines two propositions to form their logical "exclusive or"

- *E.g.* If *p* = "I will earn an A in this course." and
  q = "I will drop this course.", then

p ⊕ *q* = "I will **either** earn an A in this course,
or I will drop it (**but not both**!)"

Exclusive-Or Truth Table

p	q	$p \oplus q$
T	T	**F**
T	F	T
F	T	T
F	F	F

Note difference from OR.

- Note that $p \oplus q$ means that p is true, or q is true, but **not both**!
- This operation is called ***exclusive or***, because it **excludes** the possibility that both p and q are true.

Natural Language is Ambiguous

- Note that the English "or" can be ambiguous regarding the "both" case!
- "Pat is a singer or Pat is a writer." - $\vee$
- "Pat is a man or Pat is a woman." - $\oplus$

p	q	p "or" q
T	T	?
T	F	T
F	T	T
F	F	F

- Need context to disambiguate the meaning!
- For this class, assume "or" means inclusive ($\vee$).

The Implication Operator

- The conditional statement (aka ***implication***)

$p \rightarrow q$ states that *p* implies *q.*

- *I.e.*, If *p* is true, then *q* is true; but if *p* is not true, then *q* could be either true or false.
- *E.g.*, let *p* = "You study hard."

q = "You will get a good grade."

$p \rightarrow q$ = "If you study hard, then you will get a good grade." (else, it could go either way)

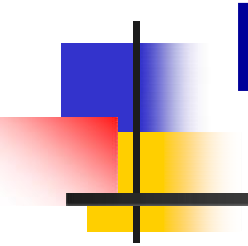
  - *p*: *hypothesis* or *antecedent* or *premise*
  - *q*: *conclusion* or *consequence*

Implication Truth Table

p	q	$p \rightarrow q$	
T	T	T	
T	F	**F**	The only False case!
F	T	T	
F	T	T	

- $p \rightarrow q$ is **false** only when p is true but q is **not** true.
- $p \rightarrow q$ does **not** require that p or q **are ever true**!
- *E.g.* "(1=0) $\rightarrow$ pigs can fly" is TRUE!

Discrete Mathematics for Computer Science



Lecture 2

Chapter 1. The Foundations

1.1 Propositional Logic

Review: The Implication Operator

- The conditional statement (***implication***) $p \rightarrow q$ states that *p* implies *q.*
- *I.e.*, If *p* is true, then *q* is true; but if *p* is not true, then *q* could be either true or false.
- *E.g.*, let *p* = "You study hard."
  q = "You will get a good grade."
  $p \rightarrow q$ = "If you study hard, then you will get a good grade." (else, it could go either way)
  - *p*: *hypothesis* or *antecedent* or *premise*
  - *q*: *conclusion* or *consequence*

Review: Implication Truth Table

p	q	$p \rightarrow q$
T	T	T
T	F	**F**
F	T	T
F	F	T

The only False case!

- $p \rightarrow q$ is **false** only when p is true but q is **not** true.
- $p \rightarrow q$ does **not** require that p or q **are ever true**!
  - *E.g.* "(1=0) $\rightarrow$ pigs can fly" is TRUE!

$p \rightarrow q$ English Phrases

- "*p* implies *q*"
- "if *p*, then *q*"
- "if *p*, *q*"
- "when *p*, *q*"
- "whenever *p*, *q*"
- "*q* if *p*"
- "*q* when *p*"
- "*q* whenever *p*"
- "*p* only if *q*"
- "*p* is sufficient for *q*"
- "*q* is necessary for *p*"
- "*q* follows from *p*"
- "*q* is implied by *p*"

Converse, Inverse, *Contrapositive*

- Some terminology, for an implication $p \rightarrow q$:
- Its ***converse*** is: $q \rightarrow p$.
- Its ***inverse*** is: $\neg p \rightarrow \neg q$.
- Its ***contrapositive***: $\neg q \rightarrow \neg p$.

p	q	$p \rightarrow q$	$q \rightarrow p$	$\neg p \rightarrow \neg q$	$\neg q \rightarrow \neg p$
T	T	T	T	T	T
T	F	F	T	T	F
F	T	T	F	F	T
F	F	T	T	T	T

- One of these three has the *same meaning* (same truth table) as $p \rightarrow q$. Can you figure out which?

Contrapositive

Examples

- p: Today is Easter
  q: Tomorrow is Monday

- $p \rightarrow q$:
  If today is Easter then tomorrow is Monday.
- ***Converse***: $q \rightarrow p$
  If tomorrow is Monday then today is Easter.
- ***Inverse***: $\neg p \rightarrow \neg q$
  If today is not Easter then tomorrow is not Monday.
- ***Contrapositive***: $\neg q \rightarrow \neg p$
  If tomorrow is not Monday then today is not Easter.

The Biconditional Operator

- The ***biconditional*** statement $p \leftrightarrow q$ states that *p* ***if and only if*** (*iff*) *q*.
- *p* = "It is below freezing."

q = "It is snowing."

$p \leftrightarrow q$ = "It is below freezing if and only if it is snowing."

or

= "That it is below freezing is necessary and sufficient for it to be snowing"

Biconditional Truth Table

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

- p is necessary and sufficient for q
- If p then q, and conversely
- p iff q

- $p \leftrightarrow q$ is equivalent to $(p \rightarrow q) \wedge (q \rightarrow p)$.
- $p \leftrightarrow q$ means that p and q have the **same** truth value.
- $p \leftrightarrow q$ does **not** imply that p and q are true.
- Note this truth table is the exact **opposite** of $\oplus$'s! Thus, $p \leftrightarrow q$ means $\neg(p \oplus q)$.

Boolean Operations

- Conjunction: $p \wedge q$, (read p and q), “discrete math is a required course and I am a computer science major”.
- Disjunction: , $p \vee q$, (read p or q), “discrete math is a required course or I am a computer science major”.
- Exclusive or: $p \oplus q$, “discrete math is a required course or I am a computer science major but not both”.
- Implication: $p \rightarrow q$, “if discrete math is a required course then I am a computer science major”.
- Biconditional: $p \leftrightarrow q$, “discrete math is a required course if and only if I am a computer science major”.

Boolean Operations

- We have seen 1 unary operator and 5 binary operators. What are they? Their truth tables are below.

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \oplus q$	$p \rightarrow q$	$p \leftrightarrow q$
T	T	F	T	T	F	T	T
T	F	F	F	T	T	F	F
F	T	T	F	T	T	T	F
F	F	T	F	F	F	T	T

- For an implication $p \rightarrow q$
- Its ***converse*** is: $q \rightarrow p$
- Its ***inverse*** is: $\neg p \rightarrow \neg q$
- Its ***contrapositive***: $\neg q \rightarrow \neg p$

Compound Propositions

- A ***propositional variable*** is a variable such as p, q, r (possibly subscripted, e.g. p_j) over the Boolean domain.
- An ***atomic proposition*** is either Boolean constant or a propositional variable: e.g. T, F, p
- A ***compound proposition*** is derived from atomic propositions by application of propositional operators:

e.g. $\neg p$, $p \vee q$, $(p \vee \neg q) \rightarrow q$

- Precedence of logical operators: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$
- Precedence also can be indicated by parentheses.
  - e.g. $\neg p \wedge q$ means $(\neg p) \wedge q$, not $\neg(p \wedge q)$

An Exercise

- Any compound proposition can be evaluated by a truth table
- $(p \vee \neg q) \rightarrow q$

p	q	$\neg q$	$p \vee \neg q$	$(p \vee \neg q) \rightarrow q$
T	T	F	T	T
T	F	T	T	F
F	T	F	F	T
F	F	T	T	F

Translating English Sentence

- Let p = "It rained last night",
  q = "The sprinklers came on last night,"
  r = "The lawn was wet this morning."

  Translate each of the following into English:

  $\neg p$ = "It didn't rain last night."

  $r \wedge \neg p$ = "The lawn was wet this morning, and it didn't rain last night."

  $\neg r \vee p \vee q$ = "The lawn wasn't wet this morning, or it rained last night, or the sprinklers came on last night."

Another Example

- Find the converse of the following statement.
  - "Raining tomorrow is a sufficient condition for my not going to town."
- **Step 1**: Assign propositional variables to component propositions.
  - *p*: It will rain tomorrow
  - *q*: I will not go to town
- **Step 2**: Symbolize the assertion: $p \rightarrow q$
- **Step 3**: Symbolize the converse: $q \rightarrow p$
- **Step 4**: Convert the symbols back into words.
  - "If I don't go to town then it will rain tomorrow" or
  - "Raining tomorrow is a *necessary condition* for my not going to town."

Logic and Bit Operations

- A ***bit*** is a binary (base 2) digit: 0 or 1.
- Bits may be used to represent truth values.
  - By convention:

0 represents "False"; 1 represents "True".

- A ***bit string of length n*** is an ordered sequence of $n \geq 0$ bits.
- By convention, bit strings are (sometimes) written left to right:
  - e.g. the "first" bit of the bit string "1001101010" is 1.
  - What is the length of the above bit string?

Bitwise Operations

- Boolean operations can be extended to operate on bit strings as well as single bits.

- Example:

01 1011 0110

11 0001 1101

11 1011 1111 Bit-wise OR

01 0001 0100 Bit-wise AND

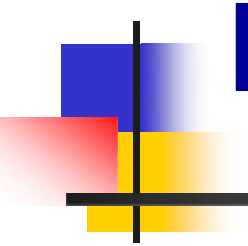
10 1010 1011 Bit-wise XOR

End of 1.1

You have learned about:

- Propositions: what they are
- Propositional logic operators'
  - symbolic notations, truth tables, English equivalents, logical meaning
- Atomic vs. compound propositions
- Bits, bit strings, and bit operations

- Next section:
  - Propositional equivalences
  - Equivalence laws
  - Proving propositional equivalences



Lecture 3

Chapter 1. The Foundations

2. Propositional Equivalences
3. Predicates and Quantifiers

1.2 Propositional Equivalence

- A ***tautology*** is a compound proposition that is **true** *no matter what* the truth values of its atomic propositions are!
  - *e.g.* $p \vee \neg p$ ("Today the sun will shine or today the sun will not shine.") [What is its truth table?]
- A ***contradiction*** is a compound proposition that is **False.**
  - *e.g.* $p \wedge \neg p$ ("Today is Wednesday and today is not Wednesday.") [Truth table?]
- A ***contingency*** is a compound proposition that is neither a tautology nor a contradiction.
  - *e.g.* $(p \vee q) \rightarrow \neg r$

Logical Equivalence

- Compound proposition p is ***logically equivalent*** to compound proposition q, written $p \equiv q$ or $p \Leftrightarrow q$, **iff** the compound proposition $p \leftrightarrow q$ is a tautology.

- Compound propositions p and q are logically equivalent to each other **iff** p and q contain the same truth values as each other in <u>all</u> corresponding rows of their truth tables.

Proving Equivalence via Truth Tables

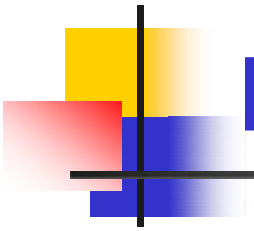
- Prove that $\neg(p \wedge q) \equiv \neg p \vee \neg q$. (De Morgan's law)

p	q	$p \wedge q$	$\neg p$	$\neg q$	$\neg p \vee \neg q$	$\neg(p \wedge q)$
T	T	T	F	F	F	F
T	F	F	F	T	T	T
F	T	F	T	F	T	T
F	F	F	T	T	T	T

- Show that

Check out the solution in the textbook!

  - $\neg(p \vee q) \equiv \neg p \wedge \neg q$ (De Morgan's law)
  - $p \rightarrow q \equiv \neg p \vee q$
  - $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$ (distributive law)



Equivalence Laws

- These are similar to the arithmetic identities you may have learned in algebra, but for propositional equivalences instead.

- They provide a pattern or template that can be used to match part of a much more complicated proposition and to find an equivalence for it and possibly simplify it.

Equivalence Laws

- *Identity*: $p \wedge \mathbf{T} \equiv p$ $p \vee \mathbf{F} \equiv p$
- *Domination*: $p \vee \mathbf{T} \equiv \mathbf{T}$ $p \wedge \mathbf{F} \equiv \mathbf{F}$
- *Idempotent*: $p \vee p \equiv p$ $p \wedge p \equiv p$
- *Double negation:* $\neg\neg p \equiv p$
- *Commutative:* $p \vee q \equiv q \vee p$ $p \wedge q \equiv q \wedge p$
- *Associative:* $(p \vee q) \vee r \equiv p \vee (q \vee r)$
  $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$

More Equivalence Laws

- *Distributive*: $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$
  $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
- *De Morgan's*:
  $\neg(p \wedge q) \equiv \neg p \vee \neg q$
  $\neg(p \vee q) \equiv \neg p \wedge \neg q$
- *Absorption*
  $p \vee (p \wedge q) \equiv p$ $p \wedge (p \vee q) \equiv p$
- *Trivial tautology/contradiction*:
  $p \vee \neg p \equiv \mathbf{T}$ $p \wedge \neg p \equiv \mathbf{F}$

See Table 6, 7, and 8 of Section 1.2

Defining Operators via Equivalences

Using equivalences, we can *define* operators in terms of other operators.

Exclusive or: $p \oplus q \equiv (p \wedge \neg q) \vee (\neg p \wedge q)$

$p \oplus q \equiv (p \vee q) \wedge \neg(p \wedge q)$

- Implies: $p \rightarrow q \equiv \neg p \vee q$
- Biconditional: $p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$

$p \leftrightarrow q \equiv \neg(p \oplus q)$

$p \leftrightarrow q \equiv \neg p \leftrightarrow \neg q$

$p \leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$

$\neg(p \leftrightarrow q) \equiv p \leftrightarrow \neg q$

This way we can “normalize” propositions

Logical Equivalences Involving Conditional Statements.

- $p \rightarrow q \equiv \neg p \vee q$
- $p \rightarrow q \equiv \neg q \rightarrow \neg p$
- $p \vee q \equiv \neg p \rightarrow q$
- $p \wedge q \equiv \neg(p \rightarrow \neg q)$
- $\neg(p \rightarrow q) \equiv p \wedge \neg q$
- $(p \rightarrow q) \wedge (p \rightarrow r) \equiv p \rightarrow (q \wedge r)$
- $(p \rightarrow r) \wedge (q \rightarrow r) \equiv (p \vee q) \rightarrow r$
- $(p \rightarrow q) \vee (p \rightarrow r) \equiv p \rightarrow (q \vee r)$
- $(p \rightarrow r) \vee (q \rightarrow r) \equiv (p \wedge q) \rightarrow r$

An Example Problem

- Show that $\neg(p \rightarrow q)$ and $p \wedge \neg q$ are logically equivalent.

$\neg(p \rightarrow q)$ [Expand definition of $\rightarrow$]

$\equiv \neg(\neg p \vee q)$ [DeMorgan's Law]

$\equiv \neg(\neg p) \wedge \neg q$ [Double Negation]

$\equiv p \wedge \neg q$

Another Example Problem

- Check using a symbolic derivation whether

$$(p \wedge \neg q) \rightarrow (p \oplus r) \equiv \neg p \vee q \vee \neg r$$

$(p \wedge \neg q) \rightarrow (p \oplus r)$ [Expand definition of $\rightarrow$]

$\equiv \neg(p \wedge \neg q) \vee (p \oplus r)$ [Expand definition of $\oplus$]

$\equiv \neg(p \wedge \neg q) \vee ((p \vee r) \wedge \neg(p \wedge r))$

[DeMorgan's Law]

$\equiv (\neg p \vee q) \vee ((p \vee r) \wedge \neg(p \wedge r))$

cont.

Example Continued...

$$(p \wedge \neg q) \rightarrow (p \oplus r) \equiv \neg p \vee q \vee \neg r$$

$(\neg p \vee q) \vee ((p \vee r) \wedge \neg(p \wedge r))$ [$\vee$ Commutative]

$\equiv (q \vee \neg p) \vee ((p \vee r) \wedge \neg(p \wedge r))$ [$\vee$ Associative]

$\equiv q \vee (\neg p \vee ((p \vee r) \wedge \neg(p \wedge r)))$ [Distribute $\vee$ over $\wedge$]

$\equiv q \vee (((\neg p \vee (p \vee r)) \wedge (\neg p \vee \neg(p \wedge r))))$ [$\vee$ Assoc.]

$\equiv q \vee (((\neg p \vee p) \vee r) \wedge (\neg p \vee \neg(p \wedge r))))$ [Trivial taut.]

$\equiv q \vee ((\mathbf{T} \vee r) \wedge (\neg p \vee \neg(p \wedge r)))$ [Domination]

$\equiv q \vee (\mathbf{T} \wedge (\neg p \vee \neg(p \wedge r)))$ [Identity]

$\equiv q \vee (\neg p \vee \neg(p \wedge r))$

cont.

End of Long Example

$$(p \wedge \neg q) \rightarrow (p \oplus r) \equiv \neg p \vee q \vee \neg r$$

$q \vee (\neg p \vee \neg(p \wedge r))$ [DeMorgan's Law]

$\equiv q \vee (\neg p \vee (\neg p \vee \neg r))$ [$\vee$ Associative]

$\equiv q \vee ((\neg p \vee \neg p) \vee \neg r)$ [Idempotent]

$\equiv q \vee (\neg p \vee \neg r)$ [Associative]

$\equiv (q \vee \neg p) \vee \neg r$ [$\vee$ Commutative]

$\equiv \neg p \vee q \vee \neg r$ ■

Review: Propositional Logic

- Atomic propositions: p, q, r, …
- Boolean operators: $\neg \wedge \vee \oplus \rightarrow \leftrightarrow$
- Compound propositions: $(p \wedge \neg q) \vee r$
- Equivalences: $p \wedge \neg q \Leftrightarrow \equiv \neg(p \rightarrow q)$
- Proving equivalences using:
  - Truth tables
  - Symbolic derivations (series of logical equivalences) $p \equiv q \equiv r \equiv \cdots$

1.3 Predicate Logic

- Consider the sentence

  "For every x, $x > 0$"

  If this were a true statement about the positive integers, it could not be adequately symbolized using only statement letters, parentheses and logical connectives.

  The sentence contains two new features: a ***predicate*** *and a* ***quantifier***

Subjects and Predicates

- In the sentence “The dog is sleeping”:
  - The phrase “the dog” denotes the ***subject*** – the *object* or *entity* that the sentence is about.
  - The phrase “is sleeping” denotes the ***predicate*** – a property that the subject of the statement can have.
- In predicate logic, a ***predicate*** is modeled as a ***proposional function P(·)*** from subjects to propositions.
  - $P(x)$ = “x is sleeping” (where x is any subject).
  - P(The cat) = “*The cat* is sleeping” (proposition!)

More About Predicates

- Convention: Lowercase variables *x*, *y*, *z...* denote subjects; uppercase variables *P*, *Q*, *R*… denote propositional functions (or predicates).
- Keep in mind that the *result of applying* a predicate *P* to a value of subject *x* is the *proposition*. But the predicate *P,* or the statement *P*(*x*) **itself** (*e.g. P* = "is sleeping" or *P*(*x*) = "*x* is sleeping") is **not** a proposition.
  - *e.g.* if *P*(*x*) = "*x* is a prime number",

    P(3) is the *proposition* "3 is a prime number."

Propositional Functions

- Predicate logic *generalizes* the grammatical notion of a predicate to also include propositional functions of **any** number of arguments, each of which may take **any** grammatical role that a noun can take.
  - *e.g.*:

    let $P(x,y,z)$ = "*x* gave *y* the grade *z*"

    then if

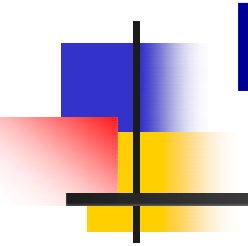
    x = "Mike", y = "Mary", z = "A",

    then

    $P(x,y,z)$ = "**Mike** gave **Mary** the grade **A**."

Examples

- Let $P(x)$: $x > 3$. Then
  - $P(4)$ is TRUE/FALSE $4 > 3$
  - $P(2)$ is TRUE/FALSE $2 > 3$
- Let $Q(x, y)$: x is the capital of y. Then
  - Q(Washington D.C., U.S.A.) is TRUE
  - Q(Hilo, Hawaii) is FALSE
  - Q(Massachusetts, Boston) is FALSE
  - Q(Denver, Colorado) is TRUE
  - Q(New York, New York) is FALSE
- Read EXAMPLE 6
  - If **x > 0** then x:= x + 1 (in a computer program)



Lecture 4

Chapter 1. The Foundations

1.3 Introduction of Logic Circuits,Predicates and Quantifiers

Logic Circuits

- A logic circuit (or digital circuit) receives input signals p1, p2, . . . , pn, each a bit [either 0 (off) or 1 (on)], and produces output signals s1, s2, . . . , sn, each a bit.

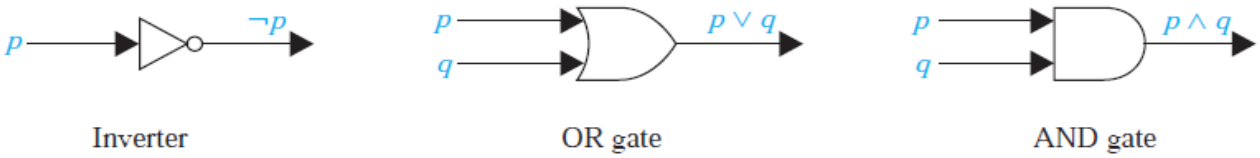


FIGURE 1 **Basic logic gates.**

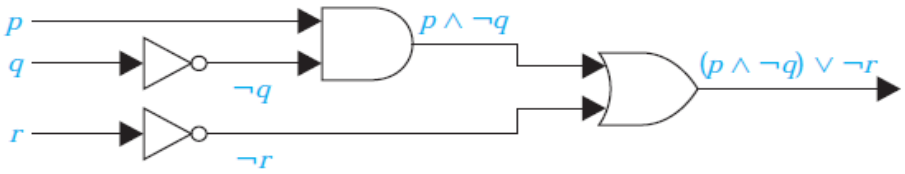


FIGURE 2 **A combinatorial circuit.**

EXAMPLE

Build a digital circuit that produces the output $(p \vee \neg r) \wedge (\neg p \vee (q \vee \neg r))$ when given input bits p, q, and r.

Solution:

To construct the desired circuit, we build separate circuits for $p \vee \neg r$ and for $\neg p \vee (q \vee \neg r)$ and combine them using an AND gate.

To construct a circuit for $p \vee \neg r$, we use an inverter to produce $\neg r$ from the input r. Then, we use an OR gate to combine p and $\neg r$.

To build a circuit for $\neg p \vee (q \vee \neg r)$, we first use an inverter to obtain $\neg r$.

Then we use an OR gate with inputs q and $\neg r$ to obtain $q \vee \neg r$.

Finally, we use another inverter and an OR gate to get $\neg p \vee (q \vee \neg r)$ from the inputs p and $q \vee \neg r$.

To complete the construction, we employ a final AND gate, with inputs $p \vee \neg r$ and $\neg p \vee q \vee \neg r$).

SOLUTION

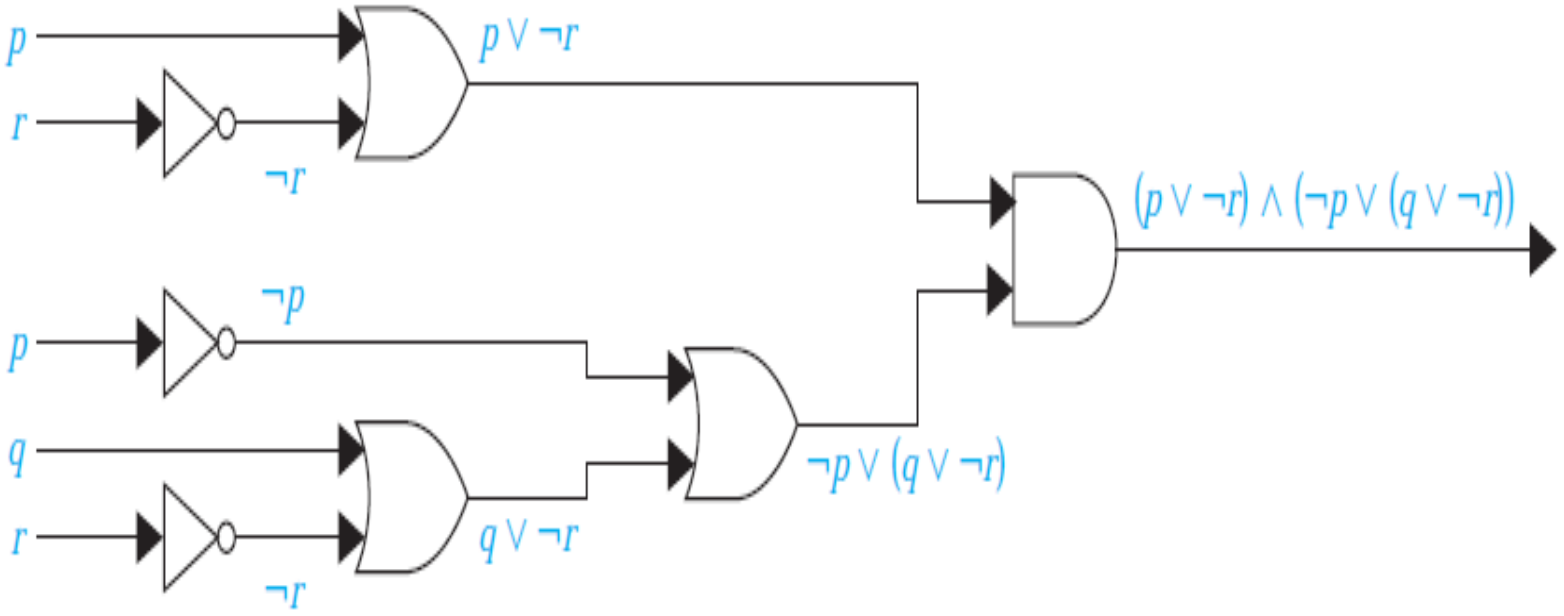


FIGURE 3 The circuit for $(p \vee \neg r) \wedge (\neg p \vee (q \vee \neg r))$.

Propositional Satisfiability

- A compound proposition is **satisfiable** if there is an assignment of truth values to its variables that makes it true.

- compound proposition is **unsatisfiable** if and only if its negation is true for all assignments of truth values to the variables, that is, if and only if its negation is a **tautology**.

EXAMPLE

Determine whether each of the compound propositions
$(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p)$,
$(p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$, and
$(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p) \wedge (p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$
is satisfiable.

SOL:

- $(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p)$ is true when the three variable p, q, and r have the same truth value.
- Hence, it is satisfiable as there is at least one assignment of truth values for p, q, and r that makes it true.
- $(p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$ is true when at least one of p, q, and r is true and at least one is false.
- Hence, $(p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$ is satisfiable,

as there is at least one assignment of truth values for p, q, and r that makes it true.

EXAMPLE con...

$(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p) \wedge (p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$ to be true, $(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p)$ and $(p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$ must both be true.

For the first to be true, the three variables must have the same truth values,

for the second to be true, at least one of three variables must be true and at least one must be false.

these conditions are contradictory. From these observations we conclude that no assignment of truth values to p, q, and r makes $(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p) \wedge (p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$ true. Hence, it is unsatisfiable.

Predicates

- Propositional logic, cannot adequately express the meaning of all statements in mathematics and in natural language.
- For example, suppose that we know that

"Every computer connected to the university network is functioning properly."

Statements involving variables, such as

"*x* > 3," "*x* = *y* + 3," "*x* + *y* = *z*,"

and

"computer *x is under attack by an intruder,*"

and

"computer *x is functioning properly,*"

These statements are neither true nor false when the values of the variables are not specified.

propositions can be produced from such statements.

Predicates

The statement "*x is greater than 3" has two parts.*

The first part, the variable x, is the subject of the statement.

The second part—the predicate, "is greater than 3"—refers to a property that the subject of the statement can have.

We can denote the statement "*x is greater than 3" by P(x),*

where *P denotes the predicate "is greater than 3" and x is the variable. The statement P(x) is* also said to be the value of the propositional function *P at x.*

Once a value has been assigned to the variable *x, the statement P(x) becomes a proposition and has a truth value.*

EXAMPLE

Let *P(x) denote the statement "x > 3." What are the truth values of P(4) and P(2)?*

EXAMPLE

Let A(x) denote the statement "Computer x is under attack by an intruder." Suppose that of the computers on campus, only CS2 and MATH1 are currently under attack by intruders. What are truth values of A(CS1), A(CS2), and A(MATH1)?

Previously…

- In predicate logic, a ***predicate*** is modeled as a ***proposional function P(·)*** from subjects to propositions.
  - *P*(*x*): "*x* is a prime number" (*x*: any subject)
  - *P*(3): "3 is a prime number." (proposition!)
- Propositional functions of **any** number of arguments, each of which may take **any** grammatical role that a noun can take
  - *P*(*x*,*y*,*z*): "***x*** gave ***y*** the grade ***z***"
  - *P*(Mike,Mary,A): "**Mike** gave **Mary** the grade **A**."

Quantifiers

- Quantification expresses the extent to which a predicate is true over a range of elements. In English, the words all, some, many, none, and few are used in quantifications.
- universal quantification- predicate is true for every element under consideration.
- existential quantification- there is one or more element under consideration for which the predicate is true.

Universe of Discourse (U.D.)

- The power of distinguishing subjects from predicates is that it lets you state things about *many* objects at once.
- e.g., let $P(x)$ = "$x + 1 > x$". We can then say, "For ***any*** number x, $P(x)$ is true" instead of $(\mathbf{0} + 1 > \mathbf{0}) \wedge (\mathbf{1} + 1 > \mathbf{1}) \wedge (\mathbf{2} + 1 > \mathbf{2}) \wedge \ldots$

- The collection of values that a variable x can take is called x's ***universe of discourse*** or the ***domain of discourse***.

Quantifier Expressions

- ***Quantifiers*** provide a notation that allows us to *quantify* (count) *how many* objects in the universe of discourse satisfy the given predicate.

- "∀" is the FOR ALL or ***universal*** quantifier.

∀*x P*(*x*) means *for all* *x* in the domain, *P*(*x*).

- "∃" is the EXISTS or ***existential*** quantifier.

∃*x P*(*x*) means there *exists* an *x* in the domain (that is, 1 or more) such that *P*(*x*).

The Universal Quantifier ∀

- ***∀x P(x): For all x in the domain, P(x).***
- ∀*x P*(*x*) is
  - *true* if *P*(*x*) is true for every *x* in *D* (*D: domain of discourse*)
  - *false* if *P*(*x*) is false for at least one *x* in *D*
    - For every real number *x*, $x^2 \geq 0$ TRUE
    - For every real number *x*, $x^2 - 1 > 0$ FALSE
    - A ***counterexample*** to the statement ∀*x P*(*x*) is a value *x* in the domain *D* that makes *P*(*x*) false
    - What is the truth value of ∀*x P*(*x*) when the domain is empty? TRUE

The Universal Quantifier ∀

- If all the elements in the domain can be listed as x_1, x_2,…, x_n then, $\forall x\, P(x)$ is the same as the conjunction:

$P(x_1) \wedge P(x_2) \wedge \cdots \wedge P(x_n)$

- Example: Let the domain of x be parking spaces at UH. Let $P(x)$ be the statement "x is full." Then the ***universal quantification*** of $P(x)$, $\forall x\, P(x)$, is the *proposition:*
  - "All parking spaces at UH are full."
  - or "Every parking space at UH is full."
  - or "For each parking space at UH, that space is full."

The Existential Quantifier ∃

- ***∃x P(x): There exists an x in the domain (that is, 1 or more) such that P(x).***
- $\exists x\, P(x)$ is
  - *true* if $P(x)$ is true for at least one x in the domain
  - *false* if $P(x)$ is false for every x in the domain
- What is the truth value of $\exists x\, P(x)$ when the domain is empty? FALSE
- If all the elements in the domain can be listed as $x_1, x_2, \ldots, x_n$ then, $\exists x\, P(x)$ is the same as the disjunction:

$$P(x_1) \vee P(x_2) \vee \cdots \vee P(x_n)$$

The Existential Quantifier ∃

- Example:

Let the domain of x be parking spaces at UH.

Let $P(x)$ be the statement "x is full."

Then the ***existential quantification*** of $P(x)$, $\exists x\, P(x)$, is the *proposition*:

- "Some parking spaces at UH are full."
- or "There is a parking space at UH that is full."
- or "At least one parking space at UH is full."

Free and Bound Variables

- An expression like $P(x)$ is said to have a ***free variable x*** (meaning, x is undefined).

- A quantifier (either $\forall$ or $\exists$) *operates* on an expression having one or more free variables, and ***binds*** one or more of those variables, to produce an expression having one or more ***bound variables***.

Example of Binding

- $P(x,y)$ has 2 free variables, x and y.
- $\forall x\, P(x,y)$ has 1 free variable  , and one bound variable  . [Which is which?]
  - "$P(x)$, where $x = 3$" is another way to bind x.
  - An expression with <u>zero</u> free variables is a bona-fide (actual) proposition.
  - An expression with <u>one or more</u> free variables is not a proposition:

  e.g. $\forall x\, P(x,y) = Q(y)$

Scope

- The statement $\exists x(x + y = 1)$, *the variable x is bound by the existential quantification* $\exists x$, *but* the variable *y is free because it is not bound by a quantifier and no value is assigned to this* variable. This illustrates that in the statement $\exists x(x + y = 1)$, *x is bound, but y is free.*

Scope

$\exists x(P(x) \wedge Q(x)) \vee \forall x R(x)$- all variables are bound

The scope of the first quantifier, $\exists x$, *is the expression* $P(x) \wedge Q(x)$ *because* $\exists x$ *is applied only to* $P(x) \wedge Q(x)$, *and* not to the rest of the statement. Similarly, the scope of the second quantifier, $\forall x$, *is the expression* $R(x)$.

Quantifiers with Restricted Domains

- An abbreviated notation is often used to restrict the domain of a quantifier, a condition a variable must satisfy is included after the quantifier.

- $\forall x{>}0\ P(x)$ is shorthand for

"For all x that are greater than zero, $P(x)$."

$= \forall x\ (x > 0 \rightarrow P(x))$

- $\exists x{>}0\ P(x)$ is shorthand for

"There is an x greater than zero such that $P(x)$."

$= \exists x\ (x > 0 \wedge P(x))$

Example

What do the statements $\forall x < 0\ (x^2 > 0)$, $\forall y = 0\ (y^3 = 0)$, and $\exists z > 0 (z^2 = 2)$ *mean, where* the domain in each case consists of the real numbers?

Solution: *The statement* $\forall x < 0\ (x^2 > 0)$ *states that for every real number* x *with* $x < 0$, $x^2 > 0$.

That is, it states "The square of a negative real number is positive." This statement is the same as $\forall x(x < 0 \rightarrow x^2 > 0)$.

The statement $\forall y = 0\ (y^3 = 0)$ *states that for every real number* y *with* $y = 0$, *we have* $y^3 = 0$. *That is, it states "The cube of every nonzero real number is nonzero."*

Note that this statement is equivalent to $\forall y(y = 0 \rightarrow y^3 = 0)$.

Finally, the statement $\exists z > 0\ (z^2 = 2)$ *states that there exists a real number* z *with* $z > 0$ such that $z^2 = 2$. *That is, it states "There is a positive square root of 2." This statement is* equivalent to $\exists z(z > 0 \wedge z^2 = 2)$.

Translating from English

- Express the statement "*Every student in this class has studied calculus*" using predicates and quantifiers.
  - Let $C(x)$ be the statement: "x has studied calculus."
  - If domain for x consists of the students in this class, then
  - it can be translated as $\forall x\, C(x)$

or

  - If domain for x consists of all people
  - Let $S(x)$ be the predicate: "x is in this class"
  - Translation: $\forall x\, (S(x) \rightarrow C(x))$

Translating from English

- Express the statement "*Some students in this class has visited Mexico*" using predicates and quantifiers.
  - Let $M(x)$ be the statement: "x has visited Mexico"
  - If domain for x consists of the students in this class, then
  - it can be translated as $\exists x\, M(x)$ or
  - If domain for x consists of all people
  - Let $S(x)$ be the statement: "x is in this class"
  - Then, the translation is $\exists x\, (S(x) \wedge M(x))$

Translating from English

- Express the statement "*Every student in this class has visited either Canada or Mexico*" using predicates and quantifiers.
  - Let $C(x)$ be the statement: "x has visited Canada" and $M(x)$ be the statement: "x has visited Mexico"
  - If domain for x consists of the students in this class, then
  - it can be translated as $\forall x\,(C(x) \vee M(x))$

Example

Q. Use predicates and quantifiers to express the system specifications "Every mail message larger than one megabyte will be compressed" and "If a user is active, at least one network link will be available."

***Solution**: Let S(m, y) be "Mail message m is larger than y megabytes," where the variable x has* the domain of all mail messages and the variable *y is a positive real number, and let C(m) denote* "Mail message *m will be compressed." Then the specification "Every mail message larger than* one megabyte will be compressed" can be represented as $\forall m(S(m, 1) \rightarrow C(m))$.

Let *A(u) represent "User u is active," where the variable u has the domain of all users,*

let *S(n, x) denote "Network link n is in state x," where n has the domain of all network* links and *x has the domain of all possible states for a network link.*

Then the specification

"If a user is active, at least one network link will be available" can be represented by $\exists u A(u) \rightarrow \exists n S(n, available)$.

Example

Q.Consider these statements. The first two are called *premises and the third is called the conclusion.*

The entire set is called an *argument.*

“All lions are fierce.”

“Some lions do not drink coffee.”

“Some fierce creatures do not drink coffee.”

Solution*: We can express these statements as:*

$\forall x(P(x) \rightarrow Q(x))$.

$\exists x(P(x) \wedge \neg R(x))$.

$\exists x(Q(x) \wedge \neg R(x))$.

Notice that the second statement cannot be written as $\exists x(P(x) \rightarrow \neg R(x))$. *The reason is that* $P(x) \rightarrow \neg R(x)$ *is true whenever x is not a lion, so that* $\exists x(P(x) \rightarrow \neg R(x))$ *is true as long as* there is at least one creature that is not a lion, even if every lion drinks coffee. Similarly, the third statement cannot be written as $\exists x(Q(x) \rightarrow \neg R(x))$.

Example

Consider these statements, of which the first three are premises and the fourth is a valid conclusion.

"All hummingbirds are richly colored."

"No large birds live on honey."

"Birds that do not live on honey are dull in color."

"Hummingbirds are small."

Let *P(x), Q(x), R(x), and S(x) be the statements "x is a hummingbird," "x is large," "x lives on* honey," and "*x is richly colored," respectively. Assuming that the domain consists of all birds,* express the statements in the argument using quantifiers and *P(x), Q(x), R(x), and S(x).*

Solution:We can express the statements in the argument as

$\forall x(P(x) \rightarrow S(x))$.

$\neg\exists x(Q(x) \wedge R(x))$.

$\forall x(\neg R(x) \rightarrow \neg S(x))$.

$\forall x(P(x) \rightarrow \neg Q(x))$.

Negations of Quantifiers

- $\forall x\, P(x)$: "Every student in the class has taken a course in calculus" ($P(x)$: "x has taken a course in calculus")
  - "Not every student in the class … calculus"

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

- Consider $\exists x\, P(x)$: "There is a student in the class who has taken a course in calculus"
  - "There is no student in the class who has taken a course in calculus"

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

Negations of Quantifiers

- Definitions of quantifiers: If the domain = {a, b, c,…}
  - $\forall x\, P(x) \equiv P(a) \wedge P(b) \wedge P(c) \wedge \cdots$
  - $\exists x\, P(x) \equiv P(a) \vee P(b) \vee P(c) \vee \cdots$
  - From those, we can prove the laws:

- $\neg\forall x\, P(x) \equiv \neg(P(a) \wedge P(b) \wedge P(c) \wedge \cdots)$

$\equiv \neg P(a) \vee \neg P(b) \vee \neg P(c) \vee \cdots$

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

- $\neg\exists x\, P(x) \equiv \neg(P(a) \vee P(b) \vee P(c) \vee \cdots)$

$\equiv \neg P(a) \wedge \neg P(b) \wedge \neg P(c) \wedge \cdots$

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

- Which *propositional* equivalence law was used to prove this?

DeMorgan's

Negations of Quantifiers

Theorem:

- **Generalized De Morgan's laws for logic**

1. $\neg\forall x\, P(x) \equiv \exists x\, \neg P(x)$

2. $\neg\forall x\, P(x) \equiv \exists x\, \neg P(x)$

Negations: Examples

- What are the negations of the statements $\forall x\,(x^2 > x)$ and $\exists x\,(x^2 = 2)$?
  - $\neg\forall x\,(x^2 > x) \equiv \exists x\,\neg(x^2 > x) \equiv \exists x\,(x^2 \leq x)$
  - $\neg\exists x\,(x^2 = 2) \equiv \forall x\,\neg(x^2 = 2) \equiv \forall x\,(x^2 \neq 2)$

- Show that $\neg\forall x(P(x) \rightarrow Q(x))$ and $\exists x(P(x) \wedge \neg Q(x))$ are logically equivalent.
  - $\neg\forall x(P(x) \rightarrow Q(x)) \equiv \exists x\,\neg(P(x) \rightarrow Q(x))$

$$\equiv \exists x\,\neg(\neg P(x) \vee Q(x))$$

$$\equiv \exists x\,(P(x) \wedge \neg Q(x))$$

Summary

TABLE 1 Quantifiers.

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

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.

Logic Programming

- A programming language is designed to reason using the rules of predicate logic. Prolog (from *Programming in Logic)*
- Prolog programs include a set of declarations consisting of two types of statements.
- **Prolog facts and Prolog rules. Prolog facts define predicates by specifying the elements that satisfy these predicates.**
- Prolog rules are used to define new predicates using those already defined by Prolog facts.

Example

Consider a Prolog program given facts telling it the instructor of each class and in which classes students are enrolled. The program uses these facts to answer queries concerning the professors who teach particular students. Such a program could use the predicates *instructor(p, c) and enrolled(s, c) to represent that professor p is the instructor of course c and that student s* is enrolled in course *c, respectively. For example, the Prolog facts in such a program might* include:

instructor(chan,math273)

instructor(patel,ee222)

instructor(grossman,cs301)

enrolled(kevin,math273)

enrolled(juana,ee222)

enrolled(juana,cs301)

enrolled(kiko,math273)

enrolled(kiko,cs301)

Example

Lowercase letters have been used for entries because Prolog considers names beginning with an uppercase letter to be variables.

A new predicate *teaches(p, s), representing that professor p teaches student s, can be* defined using the Prolog rule

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

teaches(p, s) is true if there exists a class c such that professor p is the instructor of class *c and student s is enrolled in class c.*

(Note that a comma is used to represent a conjunction of predicates in Prolog. Similarly, a semicolon is used to represent a disjunction

of predicates.)

Prolog answers queries using the facts and rules it is given. For example, using the facts and rules listed, the query ?enrolled(kevin,math273)

produces the response yes

Example

?enrolled(X,math273)

kevin

Kiko

To find all the professors who are instructors in classes being taken by Juana, we use the query

?teaches(X,juana)

This query returns

patel

grossman