Preliminary Examinations

2nd Semester, S.Y. 2024 – 2025
Transcribed by: Aerith Claude Catap
Instructor: Dr. Mary Jane C. Rabena

HAU School of Computing
First Year | Bachelor of Science in Computer Science

Syllabus:

1. **Propositional Logic**
   1.1. Propositional Variables
2. **Logical Connectives**
   2.1. Negation Operator
   2.2. Conjunction Operator
   2.3. Disjunction Operator
   2.4. Exclusive OR Operator
   2.5. Implication Operator
   2.6. Biconditional Operator
3. **Logical and Bit Operations**
4. **Compound Propositions and Truth Tables**
   4.1. Translating Logical Expressions to English Statements
   4.2. Translating English Statements to Logical Expressions
5. **Logical Equivalences**
   5.1. Propositional Equivalences

**Should there be any mistakes, inconsistencies, or inaccuracies in this material, please feel free to reach out to the transcriber. Good luck and study well for your exams!*

Last updated: January 6, 2025 at 5:00 PM

Propositional Logic

- **Logic** is the study of reasoning.
- It is the discipline that considers the methods of reasoning used to prove results. The rules of logic specify the meaning of mathematical statements, these are used to distinguish between valid and invalid mathematical arguments. Focuses on the relationship among statements, not on the content of any particular statement.
- Logic has numerous applications in computer science, such as the design of computer circuits, construction of computer programs, verification of the correctness of programs and in many other ways.
- **Propositional Logic** is the logic that deals with statements (propositions) and compound statements built from simpler statements using so-called **Boolean/Logical connectives.**

A **proposition** is simply:

- A statement (a declarative sentence)
  - with some definite meaning, (not vague or ambiguous)
- Having a truth value that is either **true** or **false**
  - It is **never** both, neither, nor somewhere in between
  - However, you might not know the actual truth value, and the truth value might depend on the situation or context.

Examples of propositions:

1. The Philippines is in Asia.
2. 1 + 1 = 2
3. 4 is an integer
4. √5 is an integer
5. 7 is an even integer
6. It's raining.

- The first three statements are all true, statements 4 and 5 are both false, while the last statement's truth value will depend on the given situation, but will only be either true or false, never both, at any given instance. Therefore, all given statements are considered propositions.

Consider the following statements:

1. What time is it?
2. Enjoy the weather!
3. 1 + x = 2
4. x + y = z

- The first statement is not a proposition since it's **not declarative** and requires a specific response that is neither true nor false. The second one is also a **non-declarative** sentence, rather it's imperative.
- For statements 3 and 4, x, y and z are not assigned any values, which does not allow us to determine whether the statement is true or if it is false. In statement 3, if x=1, it will be true but if x=2 then it will be false. Same goes for the last statement, different combinations of values for x, y and z will provide a truth value of true or false.

Propositional Variables

- These are variables that **represent propositions.** The common notations used are p, q, r and s.

Examples:
Let p be the proposition, Year 2020 is a leap year.
Let q be the proposition, Year 2020 has 366 days.

- Using the variables, the propositions can now be represented using p and q. As such, they also have truth values of either true or false. We can now say that p = true and q = true.

Logical Connectives

- Also known as **Boolean Operators**, are used to form new propositions from one, two or more existing propositions.
- There are six (6) operators use to negate or combine multiple propositions for the purpose of creating a new one.

Negation Operator (*not*, ¬)

- A **unary** operator, operates on a single value, known as the "not" operator.
- It is represented by the symbol ¬ before the propositional variable or a line on top of the propositional variable. *(e.g. ¬p or p, read as "not of p".)*

Let *p* be a proposition:

- The negation of p, denoted by ¬p, is the statement *"It is not the case that p."*
- The proposition ¬*p* is read "not *p*."
- The truth value of the negation of *p*, **<u>¬*p* is the opposite of the truth value of *p*.</u>**
  - If *p* is true, ¬*p* is false.
  - If *p* is false, ¬*p* is true.

Truth Table: **Negation** (not)	
p	**¬*p***
T	F
F	T

Preliminary Examinations

2nd Semester, S.Y. 2024 – 2025
Transcribed by: Aerith Claude Catap
Instructor: Dr. Mary Jane C. Rabena

HAU School of Computing
First Year | Bachelor of Science in Computer Science

Conjunction Operator (*and,* ∧)

- A **binary** operator, operates on two propositions at a time, known as the "and" operator.
- It is represented by the symbol ∧. *(e.g. p ∧ q, read as p and q.)*

Let *p* and *q* be propositions:
- The conjunction of *p* and *q*, denoted by ***p* ∧ *q***, is the proposition "*p* and *q*".
- The conjunction *p* ∧ *q* is **true when both p and q are true and is false otherwise.**

Truth Table: **Conjunction** (and)		
p	***q***	***p* ∧ *q***
T	T	T
T	F	F
F	T	F
F	F	F

Disjunction Operator (*or,* ∨)

- A **binary** operator known as the "or" operator or more specifically "inclusive or".
- It is represented by the symbol ∨**.** *(e.g. p V q, read as p or q)*

Let *p* and *q* be propositions:
- The disjunction of *p* and *q*, denoted by *p* ∨ *q*, is the proposition "*p* or *q*".
- The conjunction p ∨ q is **false when both *p* and *q* are false and is true otherwise.**

Truth Table: **Disjunction/Inclusive OR**		
p	***q***	***p* ∨ *q***
T	T	T
T	F	T
F	T	T
F	F	F

Exclusive-OR Operator (*xor,* ⊕)

- Also a **binary** operator, known as "exclusive or". It is represented by the symbol ⊕.

Let *p* and *q* be propositions:
- The exclusive or of *p* and *q*, denoted by *p* ⊕ *q*, is the proposition that is **true when exactly one of *p* and *q* is true and is false otherwise.**

Truth Table: **Exclusive OR**		
p	***q***	***p* ⊕ *q***
T	T	F
T	F	T
F	T	T
F	F	F

Implication Operator (*if…then,* →)

- A **binary** operator, also known as conditional statement, is denoted by the arrow symbol, →.
- The proposition *p*→*q* is read as ***p* implies *q***, where *p* is the **hypothesis** and *q* is the **conclusion**.
- Unlike other operators, the order of the propositions in an implication is important and will yield a different result when presented in different order.

Let *p* and *q* be propositions:
- The conditional statement *p* → *q*, is the proposition "**if p, then q**." Other ways to express implication:
  - if p, q
  - p only if q
  - p is sufficient for q
  - q if p
  - q when p
  - q whenever p
  - q unless ¬p
- The conditional statement **is false when p is true and q is false, and true otherwise**.

Truth Table: **Implication**		
p	***q***	***p* → *q***
T	T	T
T	F	F
F	T	T
F	F	T

Biconditional Operator (*iff,* ↔)

- A **binary** operator that is also called bi-implications, denoted by a double headed arrow, ↔.

Let *p* and *q* be propositions:
- The biconditional statement *p* ↔ *q* is the proposition "**p if and only if q**." If and only if can be shortened as *iff.*
- The biconditional statement *p* ↔ *q* is **true when p and q have the same truth values, and is false otherwise.**

Truth Table: **Biconditional**		
p	***q***	***p* ↔ *q***
T	T	T
T	F	F
F	T	F
F	F	T

Logical and Bit Operations

- Computers represent information using bits.
- A bit is a symbol with two possible values, 0 and 1.
- By convention, **1 represents T (true)** and **0 represents F (false).**
- A variable is called a **Boolean** variable if its value is either true or false.

Truth Table: **Bit Operators *(or, and, xor)***				
x	***y***	***x* ∨ *y***	***x* ∧ *y***	***x* ⊕ *y***

Preliminary Examinations

2nd Semester, S.Y. 2024 – 2025
Transcribed by: Aerith Claude Catap
Instructor: Dr. Mary Jane C. Rabena

HAU School of Computing
First Year | Bachelor of Science in Computer Science

0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

- A **bit string** is a sequence of zero or more bits. The length of this string is the number of bits in the string.

Compound Propositions and Truth Tables

- A **compound proposition** is built using logical connectives involving any number of propositional variables.
- **Parentheses** are used to specify the order in which logical operators in a compound proposition are to be applied.
- To reduce the number of parentheses, the precedence order is defined for logical operators.

Operator Precedence	
1st	¬
2nd	∧
3rd (Leftmost operator)	∨, ⊕
4th	→
5th	↔

Examples:
Construct the truth tables for each compound proposition.

1. **p ∨ q → ¬p ∧ ¬q**

p	q	¬ p	¬ q	(¬p ∧ ¬q)	(p ∨ q)	GIVEN
T	T	F	F	F	T	F
T	F	F	T	F	T	F
F	T	T	F	F	T	F
F	F	T	T	T	F	T

2. **(p →q) ⊕ (¬ p ↔ q) ∧ ¬ q**

p	q	¬ p	¬ q	(p →q)	(¬p ↔ q)	(¬p ↔ q) ∧ ¬ q	GIVEN
T	T	F	F	T	F	F	T
T	F	F	T	F	T	T	T
F	T	T	F	T	T	F	T
F	F	T	T	T	F	F	T

3. **(p ↔ ¬ q) → ¬ (q ∨ r)**

p	q	r	¬ q	(p ↔ ¬q)	(q ∨ r)	¬(q ∧ r)	GIVEN
T	T	T	F	F	T	F	T
T	T	F	F	F	T	F	T
T	F	T	T	T	T	F	F
T	F	F	T	T	F	T	T
F	T	T	F	T	T	F	F
F	T	F	F	T	T	F	F
F	F	T	T	F	T	F	T
F	F	F	T	F	F	T	T

Translating Logical Expression to English Statement

- Translating logical expressions into Boolean expressions means converting the logical connectives and propositions into mathematical operations.
- Below are examples of how different logical expressions are mapped to their corresponding Boolean expressions. Consider the following propositions:

Given: p = Mr. Lorenzo is a Math teacher.
q = Mr. Lorenzo is a LET passer.

1. $p \wedge q$ = Mr. Lorenzo is a Math teacher and he is a LET passer.
2. $p \vee q$ = Mr. Lorenzo is a Math teacher or he is a LET passer.
3. $p \rightarrow q$ = If Mr. Lorenzo is a Math teacher, then he is a LET passer.
4. $p \leftrightarrow q$ = Mr. Lorenzo is a Math teacher if and only if he is a LET passer.
5. $\neg p$ = Mr. Lorenzo is not a Math teacher.
6. $\neg q$ = Mr Lorenzo is not a LET passer.
7. $\neg p \wedge \neg q$ = Mr. Lorenzo is a not Math teacher and he is not a LET passer.
8. $\neg p \rightarrow q$ = If Mr. Lorenzo is a not Math teacher, then he is a LET passer.
9. $q \leftrightarrow \neg p$ = Mr. Lorenzo is a LET passer if and only if he is not a Math teacher.
10. $q \vee \neg p$ = Mr. Lorenzo is a LET passer or he is not a Math teacher.
11. $\neg(p \wedge q)$ = Mr. Lorenzo is not a Math teacher and he is not a LET passer.
    $\neg(p \wedge q) = \neg p \wedge \neg q$
12. $\neg(\neg p \rightarrow q)$ = If Mr. Lorenzo is a Math teacher, then he is not a LET passer.
    $\neg(\neg p \rightarrow q) = p \rightarrow \neg q$
13. $\neg(\neg p \rightarrow \neg q)$ = If Mr. Lorenzo is a Math teacher, then he is a LET passer.
    $\neg(\neg p \rightarrow \neg q) = p \rightarrow q$

Translating English Statement to Logical Expression

- It is often necessary to translate a natural language (e.g. English) sentence into logical notation. It
- makes it easier to do logical manipulations and makes you know the real meaning without any of the missing/hidden details of natural languages, which are often ambiguous.
- Some guidelines to translation:

1. If the sentence you are translating is a simple declarative sentence, then you only need to equate an atomic proposition to the sentence we want it to correspond to. **Atomic propositions** correspond to **simple, declarative** sentences.

Preliminary Examinations

2nd Semester, S.Y. 2024 – 2025
Transcribed by: Aerith Claude Catap
Instructor: Dr. Mary Jane C. Rabena

HAU School of Computing
First Year | Bachelor of Science in Computer Science

2. Assuming the sentence you have to translate is a compound sentence, the first step is to identify the simple, declarative sentences in it.
3. Once you've identified the simple, declarative sentences, translate these into atomic propositions. Assign each to a propositional variable, they can be written like this: "*p* = ..." or "Let *p* = ..."
4. Next you need to identify those words that string the simple sentences together in English, (*e.g., and, but, or, if...then, if and only if, just in case, it is not the case that, unless, only if, when, etc.*)
5. Once you've identified these words, translate them into the appropriate connectives.
6. Lastly, you need to determine the order the atomic propositions combine in with the connectives. Remember to use (but not abuse) the parentheses. They make a difference.

Given:
p = It is very cold.
q = It is raining.

1. It is very cold, and it is raining. = $\boldsymbol{p \wedge q}$
2. It is very cold, or it is raining. = $\boldsymbol{p \vee q}$
3. If it is very cold, then it is raining. = $\boldsymbol{p \rightarrow q}$
4. If it is not raining, then it is not very cold. = $\boldsymbol{\neg q \rightarrow \neg p}$, which can also be written as $\boldsymbol{\neg (q \rightarrow p)}$
5. If it is raining, then it is very cold. = $\boldsymbol{q \rightarrow p}$
6. It is not very cold if and only if it is not raining. = $\boldsymbol{\neg p \leftrightarrow \neg q}$, which can also be written as $\boldsymbol{\neg(p \leftrightarrow q)}$
7. It is not raining, or it is not very cold. = $\boldsymbol{\neg q \vee \neg p}$, (*which can also be written as* $\neg(q \vee p)$.)
8. It is not raining if and only if it is very cold. = $\boldsymbol{\neg q \leftrightarrow p}$
9. It is very cold, and it is not raining. = $\boldsymbol{p \wedge \neg q}$
10. It is not raining if and only if it is not very cold. = $\boldsymbol{\neg q \leftrightarrow \neg p}$ *(which can also be written as* $\boldsymbol{\neg(q \leftrightarrow p)}$*.)*

Logical Equivalences

1. A compound proposition that is always true, no matter what the truth values of the propositional variables that occur in it, is called a **tautology.**
2. A compound proposition that is always false is called a **contradiction**.
3. A compound proposition that is neither a tautology nor a contradiction is called a **contingency.**

- Compound propositions that have the same truth values in all possible cases are called **logically equivalent.**
- The notation $\boldsymbol{p \equiv q}$ denotes that *p* and *q* are logically equivalent.
  - The symbol ≡ is not a logical connective, and *p* ≡ *q* is not a compound proposition but rather is the statement that p ↔ q is a tautology.
  - The symbol ⇔ is sometimes used instead of ≡ to denote logical equivalence.

Propositional Equivalences

- The compound propositions *p* and *q* are called **logically equivalent** if p ↔ q is a **tautology**.
- One way to determine whether two compound propositions are equivalent is to use a **truth table.**

For example:
Assess if the propositions $\boldsymbol{\neg(p \vee q)}$ and $\boldsymbol{(\neg p \wedge \neg q)}$ are logically equivalent.

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

Therefore, since the resulting truth values for each proposition is exactly the same for each combination of T and F values (as highlighted), then the two propositions are logically equivalent. So, $\boldsymbol{\neg(p \vee q) \equiv \neg p \wedge \neg q}$.

- Another means to prove equivalence between propositions instead of determining all possible truth values using a table, is the use of **laws of logical equivalences**.
  - These are propositions that are proven to be equivalent with the application of existing mathematical and logical laws.
  - Through the laws, compound propositions can be simplified by replacing them with equivalent propositions, to prove logical equivalence.

See the laws on the fifth page of this material.

Preliminary Examinations

2nd Semester, S.Y. 2024 – 2025
Transcribed by: Aerith Claude Catap
Instructor: Dr. Mary Jane C. Rabena

HAU School of Computing
First Year | Bachelor of Science in Computer Science

Summary

Negation (*not*, ¬)
Reverses the truth value of p.

p	**¬*p***
T	**F**
F	**T**

Conjunction (*and*, ∧)
A conjunction is true only when both p and q are true.

p	***q***	***p* ∧ *q***
T	T	**T**
T	F	**F**
F	T	**F**
F	F	**F**

Disjunction/Inclusive OR (*or*, ∨)
A disjunction is true when at least one proposition is true.

p	***q***	***p* ∨ *q***
T	T	**T**
T	F	**T**
F	T	**T**
F	F	**F**

Exclusive OR (*xor*, ⊕)
An exclusive disjunction is true if both propositions have different truth values.

p	***q***	***p* ⊕ *q***
T	T	**F**
T	F	**T**
F	T	**T**
F	F	**F**

Implication (*if…then*, →)
An implication is true except when p = true and q = false.

p	***q***	***p* → *q***
T	T	**T**
T	F	**F**
F	T	**T**
F	F	**T**

Converse	***q* → *p***
Inverse	**¬*p* → ¬*q***
Contrapositive	**¬*q* → ¬*p***

Biconditional (*iff*, ↔)
A biconditional statement is true when both propositions have the same truth value.

p	***q***	***p* ↔ *q***
T	T	T
T	F	F
F	T	F
F	F	T

Operator Precedence

1st	Negation	¬
2nd	Conjunction	∧
3rd (Leftmost operator)	Disjunction	∨, ⊕
4th	Implication	→
5th	Biconditional	↔

Laws of Logical Equivalences

p ∧ **T** ≡ p p ∨ **F** ≡ p	Identity laws
p ∨ **T** ≡ **T** p ∧ **F** ≡ **F**	Domination laws
p ∨ p ≡ p p ∧ p ≡ p	Idempotent laws
¬(¬p) ≡ p	Double negation law
p ∨ q ≡ q ∨ p p ∧ q ≡ q ∧ p	Commutative law
(p ∨ q) ∨ r ≡ p ∨ (q ∨ r) (p ∧ q) ∧ r ≡ p ∧ (q ∧ r)	Associative laws
p ∨ (q ∧ r) ≡ (p ∨ q) ∧ (p ∨ r) p ∧ (q ∨ r) ≡ (p ∧ q) ∨ (p ∧ r)	Distributive laws
¬ (p ∧ q) ≡ ¬p ∨ ¬q ¬ (p ∨ q) ≡ ¬p ∧ ¬q	De Morgan's laws
p ∨ (p ∧ q) ≡ p p ∧ (p ∨ q) ≡ p	Absorption laws
p ∨ ¬p ≡ **T** p ∧ ¬p ≡ **T**	Negation laws

Bit Operators *(or, and, xor)*
0 = False; 1 = True

x	***y***	***x* ∨ *y***	***x* ∧ *y***	***x* ⊕ *y***
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0