Lecture 2

- Propositional Equivalence

Tautologies, Contradictions, and Contingencies

- A tautology is a proposition which is always **true**.
  - Example: $p \vee \neg p$; That cat is a mammal
  - $(p \wedge q) \rightarrow (p \vee q)$ is a tautology
- A *contradiction* is a proposition which is always **false**.
  - Example: $p \wedge \neg p$; That cat is a reptile.
- A *contingency* is a proposition which is **neither a tautology nor a contradiction**, such as p

p	¬p	p ∨¬p	p ∧¬p
T	F	T	F
F	T	T	F

Contingency

- A contingency is a proposition that is neither necessarily true nor necessarily false. This means that the truth value of a contingent proposition can vary depending on the circumstances or the way the world happens to be. In other words, a contingent statement can be true in some situations and false in others.
- For example, the statement "It is raining" is contingent because its truth depends on the weather at a particular time and place. If it's raining outside right now, then the statement is true. If it's not raining, the statement is false. The statement is not necessarily true (like a tautology, which is true in all possible worlds) nor is it necessarily false (like a contradiction, which is false in all possible worlds).
- Contingent statements are common in everyday language and situations, as they describe facts about the world that could be otherwise.

Logical Equivalence

- When the truth value of two statements is identical, in all possible cases, are logically equivalent.
- Denoted by $\equiv$ or $\Leftrightarrow$
- Example $p \equiv \neg(\neg p)$
- They provide a pattern or template that can be used to match all or part of a much more complicated proposition and to find an equivalence for it.

Example: Evaluate the equivalence using truth table $\neg(p \wedge q) \Leftrightarrow \neg p \vee \neg q$

Logical Equivalence

p	q	¬(p∧q)	¬p∨¬q	¬(p∧q)↔(¬p∨¬q)
True	True	False	False	True
True	False	True	True	True
False	True	True	True	True
False	False	True	True	True

Defining Operators via equivalence

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

  - Exclusive or: $p \oplus q \Leftrightarrow (p \vee q) \wedge (\neg p \vee \neg q)$
    $p \oplus q \Leftrightarrow (p \wedge \neg q) \vee (q \wedge \neg p)$
    $p \oplus q \Leftrightarrow (p \vee r) \wedge \neg(p \wedge r)$
  - **Implies:** $\boldsymbol{p \rightarrow q \Leftrightarrow \neg p \vee q}$* ***(Conditional disjunction equivalence)***
  - Biconditional: $p \leftrightarrow q \Leftrightarrow (p \rightarrow q) \wedge (q \rightarrow p)$
    $p \leftrightarrow q \Leftrightarrow \neg(p \oplus q)$

Equivalence Laws

Equivalence	Name
$p \wedge T \Leftrightarrow p$ $p \vee F \Leftrightarrow p$	Identity laws
$p \vee T \Leftrightarrow T$ $p \wedge F \Leftrightarrow F$	Domination laws
$p \vee p \Leftrightarrow p$ $p \wedge p \Leftrightarrow p$	Idempotent laws
$\neg(\neg p) \Leftrightarrow p$	Double negation law
$p \vee q \Leftrightarrow q \vee p$ $p \wedge q \Leftrightarrow q \wedge p$	Commutative laws
$(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$ $(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$	Associative laws
$p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$ $p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$	Distributive laws
$\neg(p \wedge q) \Leftrightarrow \neg p \vee \neg q$ $\neg(p \vee q) \Leftrightarrow \neg p \wedge \neg q$	De Morgan's laws
$p \vee (p \wedge q) \Leftrightarrow p$ $p \wedge (p \vee q) \Leftrightarrow p$	Absorption Law
$p \vee \neg p \Leftrightarrow T$ $p \wedge \neg p \Leftrightarrow F$	Negation Laws

TABLE 7 Logical Equivalences Involving Conditional Statements.

$$p \to q \equiv \neg p \vee q$$

$$p \to q \equiv \neg q \to \neg p$$

$$p \vee q \equiv \neg p \to q$$

$$p \wedge q \equiv \neg(p \to \neg q)$$

$$\neg(p \to q) \equiv p \wedge \neg q$$

$$(p \to q) \wedge (p \to r) \equiv p \to (q \wedge r)$$

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

$$(p \to q) \vee (p \to r) \equiv p \to (q \vee r)$$

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

TABLE 8 Logical Equivalences Involving Biconditional Statements.

$$p \leftrightarrow q \equiv (p \to q) \wedge (q \to p)$$

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

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

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

De Morgan's Laws

- Shows how to negate conjunction and disjunction.
- Negation of disjunction is formed by taking the conjunction of the negations of the compound proposition.

  $\neg (p \vee q) \Leftrightarrow \neg p \wedge \neg q$
- Negation of conjunction is formed by taking the disjunction of the negations of the compound proposition.

  $\neg (p \wedge q) \Leftrightarrow \neg p \vee \neg q$

Example: Use De Morgan's Law to express the negation of "Michael has a cellphone, and he has a laptop computer" and "Heather will go to the concert or Steve will go to the concert"

Applying De Morgan's Laws

Write negations for each of the following statements:

a. John is 6 feet tall, and he weighs at least 200 pounds.

b. The bus was late, or Tom's watch was slow.

Solutions

- John is not 6 feet tall or he weighs less than 200 pounds.
- The bus was not late and Tom's watch was not slow.
- Since the statement "neither p nor q" means the same as "~p and ~q" an alternative answer for (b) is "Neither was the bus late nor was Tom's watch slow."

- Using De Morgan's Laws, let's express the negation of the two given statements:

1. "Michael has a cellphone, and he has a laptop computer."
   - Let p represent "Michael has a cellphone" and q represent "Michael has a laptop computer."
   - The negation of the conjunction using De Morgan's Law is: $\neg(p \wedge q) \equiv (\neg p) \vee (\neg q)$
   - Which translates to: "Michael does not have a cellphone or he does not have a laptop computer."
2. "Heather will go to the concert or Steve will go to the concert."
   - Let r represent "Heather will go to the concert" and s represent "Steve will go to the concert."
   - The negation of the disjunction using De Morgan's Law is: $\neg(r \vee s) \equiv (\neg r) \wedge (\neg s)$
   - Which translates to: "Heather will not go to the concert and Steve will not go to the concert."

Constructing new Logical Equivalence

- The logical equivalence in table that have been established can be used to construct additional logical equivalence.
- A proposition in a compound proposition can be replaced by a compound proposition that is logically equivalent to it without changing the truth value of the original compound proposition
- We could use a truth table to prove the equivalence. However, we need to illustrate the use of logical identities that we already know to establish new logical identities.
- This is practically important for establishing equivalence with large number of variables.

Constructing new Logical Equivalence

Let's solve this to understand better:

- $\neg$ (p $\rightarrow$ q) and p $\wedge \neg$q is logically equivalent

$\neg$ (p $\rightarrow$ q)	$\equiv \neg(\neg p \vee q)$	By conditional/disjunction equivalence
	$\equiv \neg(\neg p \vee \neg q)$	By second De Morgan's Law
	$\equiv p \wedge \neg q$	By double negation law

Constructing new Logical Equivalence

- $\neg (p \vee (\neg p \wedge q)$ and $\neg p \wedge \neg q$ is logically equivalent

p	q	$\neg(p \vee (\neg p \wedge q))$	$\neg p \wedge \neg q$
True	True	False	False
True	False	False	False
False	True	False	False
False	False	True	True

- $(p \wedge q) \rightarrow (p \vee q)$ is a tautology

p	q	$(p \wedge q)$	$(p \vee q)$	$(p \wedge q) \rightarrow (p \vee q)$
True	True	True	True	True
True	False	False	True	True
False	True	False	True	True
False	False	False	False	True

Satisfiability

- A compound proposition is satisfiable if there is an assignment of truth values to its variables that makes it true (Tautology or contingency).
- In other words, a logical formula is satisfiable if there exists some interpretation or assignment of truth values to its variables that makes the entire formula true.
- If no such assignment exists, the formula is unsatisfiable.
- Many problems, in robotics, software testing, AI planning, computer aided design, computer networking, and more, can be designed in terms of propositional satisfiability.
- Some puzzles that can be modeled as satisfiability problems are the n-queen's problem and solving a sudoku.

Predicate and Quantifiers

- Introduce to a powerful type of logic- Predicate logic

Predicates

- A predicate is a sentence depending on variables which becomes a statement upon substituting values in the domain
- Example: x > 5 or Computer x is not functioning properly
- Here variable ‘x’ is the subject of the statement. ‘> 5’ is the predicate, which refers to a property that the statement of the subject can have.
- Denoted as P(x)
- Once a value has been assigned to x, P(x) becomes a proposition and has truth value.

Example: P(x) denoted the statement x > 5. what is the truth value of P(6) and P(2)?

Predicates

- Given the predicate $P(x)$ denotes the statement $x>5$:
  - For $P(6)$: We substitute x with 6. Since $6>5$, the statement $P(6)$ is true.
  - For $P(2)$: We substitute x with 2. Since 2 is not greater than 5, the statement $P(2)$ is false.
- Therefore, the truth values are:
  - $P(6)$ is True.
  - $P(2)$ is False.

Quantifier Expressions

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

- "∀" is the for all or universal quantifier.
  - ∀x P(x) means: For all x in the domain, P(x) is true.

- "∃" 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) is true.

The Universal Quantifier (∀) Example

Let the domain of x be types of cats.
Let P(x) be the predicate "x is a mammal."
Then the universal quantification of P(x), ∀x P(x), is the proposition:

- "All cats are mammal."
- "Every cat is a mammal."

The Universal Quantifier (∀) Example

Let the domain of x be parking spaces at UT
Let P(x) be the predicate "x is full."
Then the universal quantification of P(x), ∀x P(x), is the proposition:

- "All parking spaces at UT are full."
- "Every parking space at UT is full."
- "For each parking space at UT, that space is full."

The Existential Quantifier ∃

Let the domain of x be 'people in the world'.
Let P(x) be the predicate "oldest"
Then the existential quantification of P(x), ∃x P(x), is the proposition:

- "Some people is the oldest in the world."
- "There exist some oldest people in the world."
- "At least one person in the world is the oldest."

The Existential Quantifier ∃

Let the domain of x be parking spaces at UT.
Let P(x) be the predicate "x is full."
Then the existential quantification of P(x), ∃x P(x), is the proposition:

- "Some parking space at UT is full."
- "There is a parking space at UT that is full."
- "At least one parking space at UT is full."

Quantifiers

Type	Statement	When true	When false
Universal	∀	P(x) is true for every x	There is an x for which P(x) is false
Existential	∃	There is an x for which P(x) is true	P(x) is false for every x

The Transition

- P: “Maine Coon is a mammal”
- P(x): “x is a mammal”
- Q: $\forall x$ P(x): “Every cat is a mammal.

Logically equivalent and Negation

- Statement involving predicates and quantifiers are logically equivalent if and only if they have the same truth value no matter which predicate are substituted into these statements and which domain of discourse is used for the variables in these propositional functions.
- Statement 1 is equal to statement 2 $\mathcal{S} \equiv \mathcal{T}$
- Negating Quantifier: Define $\forall x, \exists x$ for a universe with elements {1, 2, 3…..n}

  $\forall x P(x) \equiv P(1) \wedge P(2) \ldots . \wedge P(n)$

  $\exists x P(x) \equiv P(1) \vee P(2) \ldots \vee P(n)$

- All Equivalencies

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

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

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

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

Negate the following that has two quantifiers:

$$\forall x \exists y [P(x, y) \wedge Q(y)]$$

Logic Programming

- Prolog- An important programming language is designed in 1970's to reason using the rules of predicate logic.
- Widely used in AI- IBM Watson's supercomputer
- It is a declarative language- like SQL. Assigns a task, doesn't care about how its done.
- Prolog programs includes a set of declarations consisting of two types of statements, Organized data as 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.
- These facts and rules are kept in one document and in other areas, you can create queries.

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 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,es301)
```

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

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

which means that 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)
yes
```

Returns yes because the fact enrolled(kevin, math273) was provided as input. The query

```
?enrolled(X,math273)
kevin
kiko
```

To produce this response, Prolog determines all possible values of X for which enrolled(X, math273) has been included as Prolog fact. Similarly. to find all the professors who are instructors in classes being taken by Juana, we use the query

```
?teaches(X,juana)
patel
grossman
```