UECM1303 Discrete Mathematics with Applications Topics 1 & 2: Logic & Arguments of Propositions & Quantified Statements (Predicate Calculus, First-Order Logic)

Lecturer: Dr. Liew How Hui
Email: liewhh@utar.edu.my

12+8=20 lecture hours

Discrete mathematics is the study of mathematical structures that are fundamentally **discrete** as opposed to mathematical structures that are continuous (based on real number line) as in Calculus.

The **objects** studied in discrete mathematics are

- First Order Logic (Week 1–6):
  - Logic of Compound Statements (propositions) and Quantified Statements (predicates) → Model Theory
  - Valid and Invalid Arguments → Proof Theory
- Elementary Number Theory and Methods of Proof (Week 6–9)
- Set Relations (Week 10–12)
- Graph Theory (not cover in this course)
- Functional Data Structures, Algorithms and Complexity Analysis (not cover in this course)

Main reference:

1. Epp, S.S., 2020. Discrete Mathematics with Applications. 5th ed. Boston, MA: Brooks/Cole Cengage Learning. (Amazon: In 2019, US$ 232.45)

Additional references:

2. Rosen, K.H., 2019. Discrete Mathematics and its Applications. 8th ed. New York: McGraw-Hill. (Amazon: In 2019, US$ 94.50)
3. Scheinerman, E.R., 2013. Mathematics: A Discrete Introduction. 3rd ed. Boston, Mass.: Brooks/Cole. (Amazon: In 2019, US$ 258.98)

Other references: [Halmos, 1960, Devlin, 1993] formalise "logic" using set theory as the "meta-language". Kac et al. [2008], http://openlogicproject.org/, Huth and Ryan [2004], Mendelson [1997], Rautenberg [2010], Forster [2003], Hinman [2005], etc.

Class arrangement
• Week 1 : Tutorial class starts. We try to complete all classes by Week 13 (we need 46 hours lecture and 12 hours tutorial)
• Week 1 Friday : Awal Muharram public holiday
• Week 11 Friday : Prophet Muhammads's Birthday public holiday
• Week 6/7, 11/12 : Tests.
Coursework Assessment (40%)
• Test 1 (20%, Week 6 or Week 7?, Covering Topics 1 & 2)
• Test 2 (20%, Week 11 or Week 12?, Covering Topics 3 & 4)
Final Exam (60%)
• 4 Questions: Each 15% (1 optional + 3 compulsory)

When completing this subject, one should be able to:

CLO1 Recognise statements and quantified statements. (Topic 1) .....Bloom's Taxonomy Level: C1
CLO2 Determine the validity of an argument. (Topic 2) .....Bloom's Taxonomy Level: C2
CLO3 Demonstrate various proof-techniques. (Topic 3) .....Bloom's Taxonomy Level: C3
CLO4 Express relations correctly with their mathematical properties (Topic 4) .....Bloom's Taxonomy Level: C2

Contents

(Mathematical) logic is an **abstraction** of mathematical and some real-world statements into formalism, i.e. **expressing** statements in terms of **symbols**.

In this topic, we are introducing two domains of mathematical logic:

- semantics : `https://en.wikipedia.org/wiki/Model_theory`; and
- syntactic : `https://en.wikipedia.org/wiki/Proof_theory`

for **propositional logic/calculus** and **predicate logic/calculus**.

A Programming Language from Mathematical Abstraction

To illustrate the relation of discrete mathematics to computer science, we introduce **Standard ML** (ML stands for Meta Language), a kind of **typed functional programming language** which is developed from the abstraction of mathematical functions. We will also be using Standard ML to illustrate how "functions" can be composed to perform computation.

Standard ML is not a popular programming language because it does not have good programming libraries compare to popular languages such as Python, Java, C++, etc. as well as other functional programming languages such as Haskell, Ocaml, Scala, Scheme, Common Lisp, Lean 4, Rocq (Coq) etc.

Despite being not popular, Standard ML is used here because it is a simple programming language which has little changes since 1997. PolyML (https://www.polyml.org/) and MLton (http://mlton.org/) are two freely available Standard ML software.

In a typed functional programming language, the paradigm of programming is

programming = compositions of functions of compatible types

where

> A **function** from a type T_1 to a type T_2 takes **every** element of the type T_1 to an element of the type T_2.

Basic Types and Compound Types

Types are like **sets** but are "**syntactic**" rather than "**semantic**".

Example 1.0.1. 2 is of 'integer' type while 4/2 can be regarded as of 'rational' type or real number type but not integer type despite having the same meaning (value).

The following are the **basic types** defined in Standard ML (beware that tilde is used to denote negative sign):

Type	Object	Examples	Operations
int	integers between Int.minInt and Int.maxInt	2, ~3	~, +, -, *, div, mod, quot, rem, sign, ..., Int.compare (a, b), =, <>, >, >=, <, <=
real	floating-point numbers	2.0, ~3.2	~, +, -, *, /, abs, sign, ..., Real.compare (a, b), >, >=, <, <=, Real.sqrt, Real.sin, Real.cos, Real.atan, Real.exp, Real.ln, Real.tan, Real.asin, Real.acos, Real.log10, ..., Real.isNan, Real.isNornal
string	string of ASCII characters (no UTF-8)	`"Strings"`	-
char	Single character	`#"y"`, `#"\n"`	`ord #"y"`, `chr 98`
bool	Boolean values	`true`, `false`	not, andalso, orelse
unit	Conceptually, the "empty" type	()	print : string → ()

Note: `and` is a keyword used in recursive function definition, so `andalso` is used for Boolean operation.

Some "basic type" conversion and relational functions are:

- convert to bool : `1 = 2`, `1 <> 2`, `1.2 > 1.3`, `1.2<=1.3`, ...
- convert to int : `round 2.5` (or `Real.toInt IEEEReal.TO_NEAREST 2.5`), `ceil 2.1` (or `Real.toInt IEEEReal.TO_POSINF 2.1`), `floor 2.5`, `trunc 2.5`, `valOf (Int.fromString "~3")`, `size "string"`, `ord #"a"`, ...
- convert to real : `Real.fromInt ~3`, `valOf (Real.fromString "-4.9")`, ...
- convert to string (for use with print to output): Int.toString, Real.toString, str, Bool.toString, concat, implode, ...
- convert to characters or list of characters: `chr #"a"`, `explode "abc"`, ...

The following are ways to form **compound types** in Standard ML (Ref: `https://en.wikibooks.org/wiki/Standard_ML_Programming/Types`):

- tuple :

  `(1, ~1.2, (false, "abc"))` : `int * real * (bool * string)` is a 3-tuple with integer type, floating number type and a 2-tuple (pair) of boolean type and string type.

- records (named tuples) :

  `{ a = 5.0, b = "five" }` : `{a:real, b:string}`.

  Note: tuples are just a special case of records, i.e. `(1, ~1.2, (false, "abc"))` is the same as `{1 = 1, 2 = ~1.2, 3 = (false, "abc") }`.

Conditional Expression, Values and Functions

There are two conditional expressions in Standard ML:

- **if** condition **then** expression1 **else** expression2
- **case** value **of** pattern1 ⇒ expression1 | pattern2 ⇒ expression2 | ... **end**

In the programming of functional programming language, there are only two basic concepts

- **values** : the elements of a type, defined using `val`.
- **functions** : the "mapping" from a type to another type, defined using `fun` (must have a name) or `fn` (can be anonymous).

Example 1.0.2. Values can be basic types, compound types, ... or functions

```
# Binding an expression to a name, i.e. val name = expression
val atuple = (1, ~1.2, (false, "abc"));
val f = fn x => 2.0 * x * x - 3.0 * x + 1.0;

fun f (x : real) = 2.0 * x * x - 3.0 * x + 1.0;
# Type is inferred by Standard ML
fun g x = 1 div x;    (* g : int -> int   *)
fun h x = 1.0 / x;    (* h : real -> real *)
fun factorial (n : int) : int =
  if n < 1 then 1 else n * factorial (n-1);
```

Since all values in a type needs to be evaluated in a function. $g(x) = 1$ div x over integer type is not a function "by definition" because $g(x)$ is undefined at $x = 0$.

Checking all values of a type is impossible and so Standard ML handles this by introducing **Exception** ("Division Exception" in this particular situation).

Functions of forms below:

- $f \ : \ T_1 \times T_2 \times \cdots \times T_n \rightarrow T_{n+1}$
- $f \ : \ T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_n \rightarrow T_{n+1}$ which means $f \ : \ T_1 \rightarrow (T_2 \rightarrow \cdots \rightarrow (T_n \rightarrow T_{n+1}))$

are equivalent according to the notion of **currying**.

With the first definition, the function call is $f(t_1, t_2, ..., t_n)$ while with the second definition, the function is $f \ t_1 \ t_2 \cdots \ t_n$ (the use of tuple is not necessary).

Currying Example

```
(* A binary function in tuple form *)
fun h1 (x, y) = 2.0 / (1.0/x + 1.0/y);

(* A binary function in a compact currying form *)
fun h2 x y = 2.0 / (1.0/x + 1.0/y);
(* h2 can also be written in currying form *)
val h3 = fn x => fn y => 2.0 / (1.0/x + 1.0/y);

(* Functions h1, h2, h3 all perform the same computations but
   may have different function calling syntax *)
h1 (1.0, 2.0);
h2  1.0  2.0;
h3  1.0  2.0;
```

There are some functions which are in **infix** form such as addition, subtraction, multiplication, etc. Standard ML allows a function with a pair of values to be used in infix using the **infix function** and for any infix function, it can be expressed as a function using **op** or change back to the normal form using **nonfix**.

Infix Form Example

```
fun xor (p, q) = (p orelse q) andalso not (p andalso q);
xor (true, true);
(* 2 is the level of precedence, larger means higher *)
infix 2 xor;
true xor true;
(* turn of infix form for xor function *)
nonfix xor;
(op +) (3, 4);    (* Change addition to prefix form *)
```

For writing large functions, we may be using case function form and local declarations.

Case function form:

```
fun f pattern1 = expression 1
  | f pattern2 = expression 2
  | ...
  | f patternN = expression N
```

Local declarations in a function:

```
fun f t1 ... tn =
  let
    val v1 = ...
    ...
    val vM = ...
    fun f1 ... = ...
    ...
    fun fN ... = ...
  in
    expression using v1 ... vM f1 ... fN
  end
```

Local Declaration Example

```
fun leapYear (year : int) =
  let
    fun isDivisible (a, b) = (a mod b) = 0
  in
    isDivisible (year, 4) andalso (
      not (isDivisible (year, 100)) orelse (isDivisible (year, 400))
    )
  end
```

List Type

So far, the data and function we define above have fixed size. But in real-world data analysis, we need **data of variable length**. The mathematical concept that captures this is **list**!

In SML, a list can be

- **empty**: `[]` or `nil`
- having an element: `[x1]` is the same as `x1::[]`
- more than en element: `[x1,x2,...]`, is the same as

`x1::x2::  ...  ::[]`

Understanding structure of a list allows us to implement operations associated with a list. For example, the length function can be defined as

Length of a List in Case Function Form

```
fun mylength [] = 0
  | mylength (_::es) = 1 + (mylength es);
```

Note that we do not need to define the basic functions related list since they are available in Standard ML:

- `alist @ blist` : joins two list into one
- null : checks if a list is empty
- length : returns the length of a list
- hd : returns the first element (head) of a list
- tl : returns the list removing the head of a list

- rev : returns a reverse of a list
- map f [x1, x2, ..., xn] : returns [f x1, f x2, ..., f xn]
- List.filter predicate list : only keep those elements from list which the predicate is true
- `foldr (op ^) "End" ["a","b","c"] : ("a" ^ ("b" ^ ("c" ^ "End")))` (folding a binary operation from the right of the list, i.e. the "tail" is put to the right)
- `foldl (op ^) "End" ["a","b","c"] : ("c" ^ ("b" ^ ("a" ^ "End")))` (folding a binary operation from the left of the list, i.e. the "tail" is put to the left)

A typical example of list is the polynomial

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n. \tag{1.1}$$

It can be represented by a list in Standard ML

$$[a0, a1, a2, ..., an]$$

The function to evaluate the polynomial (1.1) at a particular value x can be implemented in Standard ML using the Hörner form of (1.1):

$$a_0 + x(a_1 + x(a_2 + \cdots + x \cdot a_n)) \cdots). \tag{1.2}$$

Evaluate a Polynomial using Horner Form

```
fun evalpoly []      x = 0.0
  | evalpoly (a::p) x = a + x * (evalpoly p x);
```

Suppose we want to calculate $1 + 2x + 3x^2$ at $x = 0.5$, we can run the first command which expands as follows.

```
evalpoly [1.0, 2.0, 3.0] 0.5
= 1.0 + 0.5 * (evalpoly [2.0, 3.0] 0.5)
= 1.0 + 0.5 * (2.0 + 0.5 * (evalpoly [3.0] 0.5))
= 1.0 + 0.5 * (2.0 + 0.5 * (3.0 + 0.5 * (evalpoly [] 0.5)))
= 1.0 + 0.5 * (2.0 + 0.5 * (3.0 + 0.5 * 0.0))
= 1.0 + 0.5 * 2.0 + 0.5^2 * 3.0
= 2.75
```

Adding two polynomials is slightly more complex as follows:

Adding two polynomials

```
fun addpoly poly1   []      = poly1
  | addpoly []      poly2   = poly2
  | addpoly (a::p1) (b::p2) = (a+b) :: (addpoly p1 p2);
```

DataType

Datatypes are used for two basic purposes.

Purpose 1: to define enumerates like C programming language

```
datatype mybool = Mytrue | Myfalse
datatype day = Sun | Mon | Tue | Wed | Thu | Fri | Sat
```

Purpose 2: In contrast to record (or tuple), which is logically like an "**and**" (**product type**). Datatypes is used for defining **sum types** or **or types** — when something needs to be one type or another.

In particular, suppose we want to define a new type "number" that includes both integers and reals. This can be accomplished in SML by using the datatype definition.

```
(* https://www.cs.cornell.edu/courses/cs312/2008sp/recitations/rec02.html *)
datatype num = Int_num of int | Real_num of real

fun num_to_real(n:num):real =
  case n of
     Int_num(i)  => Real.fromInt(i)
   | Real_num(r) => r;
```

Datatype can be used to define binary trees, which is fundamental in computer science.

```
(* https://www.jeremykun.com/2013/04/07/a-sample-of-standard-ml-and-the-treesort-algorith
datatype 'a Tree = empty
                 | leaf of 'a
                 | node of (('a Tree) * 'a * ('a Tree))

val t2 = node(node(leaf(2), 3, leaf(4)), 6, leaf(8))

fun breadth(empty) = 0
  | breadth(leaf(_)) = 1
  | breadth(node(left, _, right)) = breadth(left) + breadth(right)

fun depth(empty) = 0
  | depth(leaf(_)) = 1
  | depth(node(left, _, right)) =
      let
         val (lDepth, rDepth) = (1 + depth(left), 1 + depth(right))
      in
         if lDepth > rDepth then lDepth else rDepth
      end

fun flatten(empty) = []
   | flatten(leaf(x)) = [x]
   | flatten(node(left, x, right)) =
               flatten(left) @ (x :: flatten(right))

fun sort(L) = flatten(foldl(insert)(empty)(L))
```

Datatype with recursion is used the logic section to represent the propositions.

§1.1 Syntax of Compound Statements (Topic 1a)

Informally, **statements** or **propositions**, are **sentences** that are either **true** or **false**, but not both. Normally they will be denoted as p, q, r, etc. or the indexed letters p_1, p_2, etc. These letters are called statement variables, that is, variables that can be replaced by statements.

Example 1.1.1. Determine whether the following sentences are statements or not. If it is a statement, determine its truth value.

(a) The year 1973 was a leap year. ..... _____

(b) 28234423783 is a prime number. ..... _____

(c) The equation $x^2 + 3x + 2 = 0$ has two different roots in $\mathbb{R}$. ..... _____

(d) $x^2 + x + 1 = 0$, x is a real number. ..... _____

(e) She is a computer science major. ..... _____

(f) Maths is fun. ..... _____

(g) Is $2^{10} - 1$ an even integer? ..... _____

(h) Read a maths book. ..... _____

General mathematical statements are combinations of simpler statements formed through some choice of the words *not*, *and*, or, *if ... then ...*, and *if and only if*. These are called **(logical) connectives** (Ref: `https://en.wikipedia.org/wiki/Logical_connective`) and are denoted by the following symbols:

Connectives	Examples	Meaning
$\sim$, $\neg$, !	$\sim P$, $\neg P$, $\overline{P}$, $-P$, $!P$	Not
$\wedge$, $\cdot$, &, &&	$P \wedge Q$, $P \cdot Q$, PQ, $P\&Q$, $P\&\&Q$	And
$\vee$, $+$, |, ||	$P \vee Q$, $P + Q$, $P\|Q$, $P\|\|Q$	Or
$\rightarrow$, $\supset$	$P \rightarrow Q$, $P \supset Q$	If ..., then ...
$\leftrightarrow$	$P \leftrightarrow Q$	If and only if

The following logical connectives are used more often in electronics (e.g. MOS integrated circuit design) rather than mathematics:

Connectives	Examples	Meaning
$\overline{\wedge}$, $\uparrow$, $\bar{\cdot}$	$P\overline{\wedge}Q$, $P \uparrow Q$, $\overline{P \cdot Q}$	NAND
$\overline{\vee}$, $\downarrow$, $\overline{+}$	$P\overline{\vee}Q$, $P \downarrow Q$, $\overline{P + Q}$	NOR
$\underline{\vee}$, $\oplus$	$P\underline{\vee}Q$, $P \oplus Q$	XOR

Formally, the rules to form a proper proposition (or statement) is stated in the following definition.

Definition 1.1.2 (Well-Formed Formula)**.** *Statements* (or *propositions*) are either atomic or compound.

1. Constants T, $\top$, $\bot$, F and single statement variables p, q, r, s, t, p_i, $i = 1, 2, 3, \cdots$ are **atomic** (or *simple* or *primitive*) **statements or formulas**.

2. If ϕ and ψ are statements (abbreviated notations), then the expressions

$$(\sim \phi),\ (\phi \wedge \psi),\ (\phi \vee \psi),\ (\phi \to \psi),\ (\phi \leftrightarrow \psi)$$

   are **(compound) statements** or **(compound) formulas**.

Too many parentheses in a formula (Definition 1.1.2) can be annoying. Parentheses are usually "simplified" in writing based on the *precedence*:

1. Evaluate parentheses first;
2. Then evaluate negations;
3. Then evaluate $\wedge$;
4. Then evaluate $\vee$;
5. Then evaluate $\to$;
6. Then evaluate $\leftrightarrow$.

This means that the "simplified" expression $p \wedge q \vee \sim r$ means $(p \wedge q) \vee (\sim r)$. If you intend to mean $p \wedge (q \vee \sim r)$, you *must* use the parentheses. The expression $p \to q \to r$ means $p \to (q \to r)$ [Huth and Ryan, 2004, Convention 1.3]. When $\to$ and $\leftrightarrow$ appear together, it is advisable to write the parentheses.

Representation of Proposition in Standard ML

```
(* File name: topic1prop.sml *)
datatype Prop = T | F | Atom of string
              | Neg  of Prop
              | Conj of Prop * Prop
              | Disj of Prop * Prop
              | Impl of Prop * Prop
              | Iff  of Prop * Prop
infix  1 Iff
infixr 2 Impl
infix  3 Disj
infix  4 Conj

fun str prop = case prop of
    T          => "T"
  | F          => "F"
  | Atom a     => a
  | Neg  f1    => "~(" ^ (str f1) ^ ")"
  | f1 Conj f2 =>  "(" ^ (str f1) ^ ")/\\(" ^ (str f2) ^ ")"
  | f1 Disj f2 =>  "(" ^ (str f1) ^ ")\\/(" ^ (str f2) ^ ")"
  | f1 Impl f2 =>  "(" ^ (str f1) ^ ")->("  ^ (str f2) ^ ")"
  | f1 Iff  f2 =>  "(" ^ (str f1) ^ ")<->(" ^ (str f2) ^ ")";

val prop1 = Atom "p" Impl Atom "q" Iff (Atom "q" Disj Atom "p" Conj Atom "q") Impl T;
val _ = print (str prop1 ^ "\n");
```

Example 1.1.3. (a) $5 < 3$. ..... atomic statement

(b) A dog is not an animal. ..... compound statement

(c) If the earth is flat, then $3 + 4 = 7$. ..... compound statement

Reading formulas:

(a) $\sim \phi$ or $\neg\phi$ is read as "not ϕ". This statement is called the "negation of ϕ".

(b) $\phi \wedge \psi$ is read as the *conjunction* of ϕ and ψ or just "ϕ and ψ".

(c) $\phi \vee \psi$ is read as the *disjunction* of ϕ and ψ, or just "ϕ or ψ".

(d) $\phi \to \psi$ is read as "ϕ implies ψ" or "if ϕ then ψ". This form of statement is called a *conditional statement* or *implication*. In $\phi \to \psi$, the statement ϕ is called the *hypothesis* and the statement ψ is called the *conclusion* or *consequent*. The statement ϕ is called the *sufficient condition* for ψ and ψ is called the *necessary condition* for ϕ.

(e) $p \leftrightarrow q$ is read as "ϕ if and only if ψ" and is normally written as "p iff q" (popularised by the mathematician Paul Halmos). This form of statement is called the *biconditional of ϕ and ψ* and ϕ is called the *necessary and sufficient condition* for ψ.

Remark 1.1.4.

1. An English sentence "ϕ but ψ" usually means "ϕ and ψ".
2. An English sentence "neither ϕ nor ψ" usually means "$\sim \phi$ and $\sim \psi$.
3. The notation for inequalities involves "and" and "or" statements. Let a, b and c be real numbers. Then
   - $a \leq b$ means "$a < b$" or "$a = b$"
   - $a < b < c$ means "$a < b$" and "$b < c$".

Let us practise how to "read" the statements in English with a few examples.

Example 1.1.5. Given the following atomic statements:

1. p: The integer 10 is even.
2. q: $2 + 3 > 1$
3. r: $3 + 7 = 10$
4. p_1: It is snowing. q_1: I am cold.
5. p_2: $0 < 2$. q_2: $-2 < 0$.

Read the compound statements (a) $\sim p$; (b) $\sim q$; (c) $\sim r$; (d) $p_1 \wedge q_1$; (e) $q \wedge p_2 \vee q_2$.

(a)
(b)
(c)
(d)
(e)

Example 1.1.6 (Tutorial 1, Q1)**.**

Let p, q, r and s denote the following statements.

p: Ali is inside
q: Ali is watching TV
r: Ali is taking his dinner
s: Ali is riding his bicycle

(a) Translate the following into English sentences.

(i) $s \wedge (q \vee \sim r)$
(ii) $p \rightarrow (q \vee r)$
(iii) $(p \vee s) \wedge (p \rightarrow q)$
(iv) $\sim s \rightarrow (p \wedge (q \vee r))$

(b) Translate the following into logical notation.

(i) Ali is neither inside nor is he riding his bicycle.
(ii) Ali is inside, and he is taking his dinner while watching TV.
(iii) Ali is not watching TV only if he is outside.
(iv) Ali is inside and taking his dinner implies that he is not riding his bicycle.
(v) If Ali is not watching TV, then if he is not taking his dinner, he is outside.

Remark 1.1.7. A variety of informal sentences are used to express $p \rightarrow q$:
If p then q
p implies q
q follows from p
p is sufficient for q
q is necessary for p
q when p
q if p
p only if q

Note that to say "p only if q" means that p can take place only if q takes place also. That is, if q does not take place, then p cannot take place (symbolically "$\sim q \rightarrow \sim p$"). Another way to say this is that if p occurs, then q must also occur.

Definition 1.1.8. Let p and q be statement variables:

1. The ***negation*** of $p \rightarrow q$ is $p \wedge \sim q$.
2. The ***contrapositive*** of $p \rightarrow q$ is $\sim q \rightarrow \sim p$.
3. The ***converse*** of $p \rightarrow q$ is $q \rightarrow p$.
4. The ***inverse*** of $p \rightarrow q$ is $\sim p \rightarrow \sim q$.

Example 1.1.9. Write the negation, contrapositive, converse and inverse of the following conditional statements:

1. If 3 is positive then 3 is nonnegative.

Solution:

Negation:	3 is positive and 3 is not nonnegative.
Contrapositive:	If 3 is not nonnegative then 3 is not positive.
Converse:	If 3 is nonnegative then 3 is positive.
Inverse:	If 3 is not positive then 3 is not non-negative.

2. If you study hard, then you will not fail UECM1303.

Example 1.1.10 (Tutorial 1, Q4)**.** Give the negation, converse, inverse and contrapositive of each the following statements.

(a) I will pass the course if I work hard.

(b) If $A = B \cap C$, then $A \subset C$.

(c) If $-2 < 4$ and $3 + 8 = 11$, then $\sin(\pi/2) = 1$.

§1.2 Semantics of Statements and Truth Table

Every language has two aspects: **syntax** and **semantics**. The syntax in Section 1.1 deals with the form or structure of the language of propositional logic. The semantics in this section adds "meaning" to the form. In the mathematical set theory language, suppose all the compound statements form a set Prop, the "semantics" or **(truth) assignment** is a mapping

$$v : \text{Prop} \to \{\text{T}, \text{F}\}.$$

Definition 1.2.1. The *truth value* of a statement/proposition is true (T or 1), if it is a true statement/proposition and false (F or 0), if it is a false statement/proposition.

Definition 1.2.2. [Hinman, 2005]

1. An *atomic truth assignment* is a function v that maps an atomic symbol (Definition 1.1.2) to a value in $\{\text{T}, \text{F}\}$ (or $\{\top, \bot\}$).
2. A ***truth assignment*** (or a ***model*** or a ***valuation***) is a function v such that for any sentences ϕ and ψ,
   (a) $v(\sim \phi) =$ if $v(\phi) = \text{F}$ then T else F
   (b) $v(\phi \vee \psi) =$ if $v(\phi) = \text{T}$ or $v(\psi) = \text{T}$ then T else F
   (c) $v(\phi \wedge \psi) =$ if $v(\phi) = \text{F}$ or $v(\psi) = \text{F}$ then F else T
   (d) $v(\phi \to \psi) =$ if $v(\phi) = \text{F}$ then T else $v(\psi)$
   (e) $v(\phi \leftrightarrow \psi) =$ if $v(\phi) = v(\psi)$ then T else F

Implementation of the Evaluation of Proposition in Standard ML

```
use "topic1prop.sml";

fun teval p tvals = case p of
    T           => true
  | F           => false
  | Atom a      => #2(hd (List.filter (fn (x,y) => x = a) tvals))
  | Neg  f1     => not (teval f1 tvals)
  | f1 Conj f2 => (teval f1 tvals) andalso (teval f2 tvals)
  | f1 Disj f2 => (teval f1 tvals) orelse (teval f2 tvals)
  | f1 Impl f2 => (not (teval f1 tvals)) orelse (teval f2 tvals)
  | f1 Iff  f2 => (teval f1 tvals) = (teval f2 tvals)

(* Example 1.2.3 *)
val prop2 = Atom "q" Impl (Neg (Atom "r") Impl (Atom "r" Impl
              (Atom "p" Disj Atom "s")));
val _ = print (str prop2 ^ "\n");

val tvals = [("p", true), ("q", true), ("r", false), ("s", false), ("t", false)];
val _ = print (Bool.toString (teval prop2 tvals) ^ "\n");
```

Example 1.2.3. Suppose $v(p)$=T, $v(q)$=T, $v(r)$=F and $v(s)$=F, find $v(q \to (\sim r \to (r \to (p \vee s))))$.

Solution: Given $v(p) = v(q) = \text{T}$, $v(r) = v(s) = \text{F}$.

$v\Big(q \to (\sim r \to (r \to (p \vee s)))\Big)$

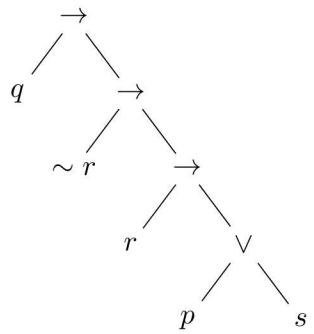
$= \text{T}$ if $v(q) = \text{F}$ else $v\Big(\sim r \to (r \to (p \vee s))\Big) = v\Big(\sim r \to (r \to (p \vee s))\Big)$

$= \text{T}$ if $v(\sim r) = \text{F}$ else $v\Big(r \to (p \vee s)\Big)$

$= \text{T}$ if (T if $v(r) = \text{F}$ else F)=F else $v\Big(r \to (p \vee s)\Big)$

$= \text{T}$ if T=F else $v\Big(r \to (p \vee s))\Big) = v\Big(r \to (p \vee s))\Big) = \text{T}$ if $v(r) = \text{F}$ else $v(p \vee s) = \text{T}$.

It is easier to understand this if we draw the parse tree of the statement $q \to (\sim r \to (r \to (p \vee s)))$.



Example 1.2.4 (Tutorial 1, Q2)**.** Given that p and q are true and r, s and t are false, find the truth value of each statement below.

(a) $(p \vee \sim q) \to (r \wedge s \wedge t)$

(b) $(q \to (r \to s)) \wedge ((p \to s) \to (\sim t))$

Example 1.2.5. Determine the hypothesis and conclusion for each of the following conditional statements. Then determine the truth value.

1. The moon is square only if the sun rises in the East.

   Solution: p: The moon is square, q: the sun rises in the East.
   $v(p) = \text{F}$, $v(q) = \text{T}$.
   According to remark 1.1.7, the informal statement could be written as $p \to q$.

Therefore, $v(p \to q) = \text{T}$.

2. 1 and 3 are prime if 1 multiply 3 is prime.

Solution: p: 1 multiply 3 is prime
q: 1 and 3 are prime
$v(p) = \text{T}$, $v(q) = \text{F}$, so $v(p \to q) = F$.

3. $(\sin \pi)(\cos \pi) = 0$ when $\sin \pi = 0$ or $\cos \pi = 0$.

Solution: p: $\sin \pi = 0$ or $\cos \pi = 0$
q: $(\sin \pi)(\cos \pi) = 0$
$v(p) = \text{T}$, $v(q) = \text{T}$, so $v(p \to q) = \text{T}$.

4. If $1 + 1 = 3$, then cats can fly.

Solution: p: $1 + 1 = 3$
q: cats can fly
$v(p) = F$, $v(q) = F$, so $v(p \to q) = T$.

Example 1.2.6. Determine the truth value of the following statements:

1. $3 < 5$ and $5 + 6 \neq 11$. .....
2. The integer 2 is even but it is a prime number. .....
3. 3 or -5 is negative. .....
4. $\sqrt{2}$ or π is an integer. .....
5. $5 \leq 5$ .....
6. 2 is prime if and only if it is multiple of 2. .....
7. 2 is negative if and only if 4 is negative. .....
8. $0 < 1 \leftrightarrow 2 < 1$ .....

Example 1.2.7 (Tutorial 1, Q3)**.** If statement q is true, determine all truth values assignments for the statements p, r and s for which the truth value of the following statement is true:

$$(q \to [(\sim p \vee r) \wedge \sim s]) \wedge [\sim s \to (\sim r \wedge q)].$$

Some statements in propositional logic is special, i.e. for all possible assignments, it will be true. Going through all possible assignments leads to the notion of truth table.

Definition 1.2.8. The *truth table* for a given statement displays the truth values that correspond to all possible combinations of truth values for its atomic statement variables.

Remark 1.2.9. If a statement s has n **atomic statements**, there will need to be 2^n **rows in the truth table for** s. So truth table is only viable for statements with less than 6 atomic statement variables for manual calculation.

Inefficient Implementation Truth Values Generation in Standard ML

```
(* File Name: topic1ttbl.sml (inspired by the following website)
https://stackoverflow.com/questions/20333184/easier-way-to-generate-a-truth-table
*)

fun tt []         = [[]]
  | tt (x :: xs) =                          (* n atom(s), n >= 1 *)
    let
      val txs = tt xs                       (* xs has (n-1) atom(s) *)
    in
      map (fn l => (x, true ) :: l) txs @  (* Join (x,T) to 2^(n-1) truth values *)
      map (fn l => (x, false) :: l) txs    (* Join (x,F) to 2^(n-1) truth values *)
    end

(* A function to print a truth table generated from the function tt *)
fun ttstr (tab : (string * bool) list list) =
  let
    val sep = " | ";
    fun myconcat (s : string list, sep : string) =
      let
        fun join (s : string list) =
          case s of
            [] => ""
          | x::xs => if xs=[] then x else x ^ sep ^ (join xs)
      in
        join s
      end
    fun b2str(x : (string * bool) list) =
      myconcat ((map (fn (_, e) => (if e then "T" else "F")) x), sep)
    val N  = (length (List.hd tab))*(1+(size sep))-(size sep)
    val h1 = myconcat (map (fn (t, _) => t) (List.hd tab), sep)
    val h2 = concat (List.tabulate (N, fn x=>"-"))
    val st = (foldr (fn (s1, s2) => s1 ^ "\n" ^ s2) "" (map b2str tab))
  in
    h1 ^ "\n" ^ h2 ^ "\n" ^ st
  end

val tab = tt ["p", "q", "r"]
val _ = print (ttstr tab);
```

Output of `topic1ttbl.sml` Program

```
p | q | r
---------
T | T | T
T | T | F
```

```
T | F | T
T | F | F
F | T | T
F | T | F
F | F | T
F | F | F
```

Theorem 1.2.10 (Logical Unary Operation)**.** *Let p be an atomic statement. If $v(p) = \mathrm{T}$, then $v(\sim p) = \mathrm{F}$; if $v(p) = \mathrm{F}$, then $v(\sim p) = \mathrm{T}$. This can be summarised as the truth table below by ignoring the cumbersome evaluation symbol $v(\ )$.*

Truth Table for $\sim p$

p	$\sim p$
T	F
F	T

Theorem 1.2.11 (Logical Binary Operations)**.** *Let p and q be atomic statement variables. The evaluation of the logical binary operations $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$ are given by the following truth table.*

$v(p)$	$v(q)$	$v(p \wedge q)$	$v(p \vee q)$	$v(p \rightarrow q)$	$v(p \leftrightarrow q)$
T	T	T	T	T	T
T	F	F	T	F	F
F	T	F	T	T	F
F	F	F	F	T	T

The truth assignment $v(\cdot)$ will be ignored in the construction of a truth table.

Example 1.2.12. Construct a truth table for the statement $(p \wedge q) \vee \sim r$.

Solution: Steps to construct a truth table for the statement:

1. This statements involves p, q and r ..... 3 statement variables.
2. Set up columns labeled p, q, r, $\sim r$, $(p \wedge q)$ and $(p \wedge q) \vee \sim r$.
3. Fill in the p, q and r columns with all the logically possible combinations of T's and F's. There are altogether $2^3 = 8$ possible combinations of truth values for p, q and r.
4. Use the truth tables for $\wedge$ and $\vee$ to fill in the $\sim r$ and $(p \wedge q)$ columns with the appropriate truth values.
5. Finally, fill in the $(p \wedge q) \vee \sim r$ column by considering truth values for $(p \wedge q)$ and $\sim r$.

The truth table for $(p \wedge q) \vee \sim r$ is shown below:

p	q	r	$\sim r$	$p \wedge q$	$(p \wedge q) \vee \sim r$
T	T	T	F	T	T
T	T	F	T	T	T
T	F	T	F	F	F
T	F	F	T	F	T
F	T	T	F	F	F
F	T	F	T	F	T
F	F	T	F	F	F
F	F	F	T	F	T

Example 1.2.13. Construct a truth table for each statement in (a) $q \wedge \sim (\sim p \rightarrow r)$; (b) $(\sim p \leftrightarrow \sim r) \vee (r \leftrightarrow q)$.

Example 1.2.14. Construct the truth table for ϕ: $(p_1 \wedge p_2) \vee (p_3 \wedge p_4)$.

§1.3 Logical Equivalences and Laws of Logical Equivalences

The truth table is used to define "logical equivalence". First, we introduce the concept of "tautology" (and related concepts).

Definition 1.3.1.

1. A statement ϕ is said to be a **tautology** or a *tautologous statement* if **for all truth assignments** for the atomic statement variables in ϕ, $v(\phi) = \mathrm{T}$, i.e. its truth values in the truth table are **all true**. If ϕ is a tautology, it is denoted as $\models \phi$.
2. A statement ϕ is said to be a **contradiction** or a **contradictory statement** if its truth values in all rows in the truth table are all false.
3. A statement ϕ is said to be a **contingency** if it is neither a tautology nor a contradiction.

Remark 1.3.2. The constant T or $\top$ is a tautology and the constant F or $\bot$ is a contradiction.

Example 1.3.3. Let p, q and r be statement variables. Show that the statement form

1. $\sim p \vee p$ is a tautology.

   Proof: First, we construct the truth table for $\sim p \vee p$:

p	$\sim p$	$\sim p \vee p$
T	F	T
F	T	T

   The truth values of $\sim p \vee p$ are **all true** in the truth table, so it is a tautology by Definition 1.3.1.

2. $\sim p \wedge p$ is a contradiction.

   Proof: We construct the truth table for $\sim p \wedge p$:

p	$\sim p$	$\sim p \wedge p$
T	F	F
F	T	F

   The truth values of $\sim p \wedge p$ are **all false** in the truth table, so it is a contradiction by Definition 1.3.1.

3. $(p \wedge q) \vee \sim r$ is a contingency.

   Proof: From the truth table in Example 1.2.12, some of the truth values of $(p \wedge q) \vee \sim r$ are true and some are false and so it is a contingency.

Example 1.3.4 (Tutorial 1, Q5)**.** Construct truth tables for the following statements:

(a) $(p \leftrightarrow q) \leftrightarrow (\sim p \leftrightarrow \sim q)$
(b) $\sim p \rightarrow (p \vee q)$
(c) $(p \rightarrow q) \leftrightarrow (\sim q \rightarrow \sim p)$

Then determine whether each statement above is a tautology, a contingency or a contradiction.

Example 1.3.5 (Tutorial 1, Q6)**.** Determine whether the following statements are tautologies.

(a) $p \rightarrow [q \rightarrow (p \wedge q)]$
(b) $(p \vee q) \wedge \sim (q \rightarrow q)$
(c) $(p \vee q) \rightarrow [q \rightarrow (p \wedge q)]$

Applying all truth assignments to the proposition from Example 1.3.5(c)

```
use "topic1ttbl.sml";
use "topic1teval.sml";

fun merge (ttbl : (string * bool) list list) (tvals : bool list) (s : string) =
  let
    val l1 = length ttbl and l2 = length tvals
  in
    if l1 <> l2 then
      [[("Different lengths", false)]]
    else
      if l1 = 0 then
        []
      else
        [(hd ttbl) @ [(s, hd tvals)]] @ (merge (tl ttbl) (tl tvals) s)
  end

(* Example 1.3.5(c): merge all possible truth assignments into the proposition *)
val prop3 = (Atom "p") Disj (Atom "q") Impl (Atom "q" Impl Atom "p" Conj Atom "q");
val ttbl  = (tt ["p", "q"]);
val tvals = map (fn tv => teval prop3 tv) ttbl;
val _ = print (ttstr (merge ttbl tvals (str prop3)));
```

The following example illustrates the use of the notation $\models$ for tautology.

Example 1.3.6 (Final Exam May 2013, Q1(a))**.** Use truth table to show that $\models (p \to q) \wedge (r \to s) \to ((p \wedge r) \to (q \wedge s))$. The truth table must contain columns for both $(p \to q) \wedge (r \to s)$ and $(p \wedge r) \to (q \wedge s)$. (10 marks)

Solution: Let s denote $(p \to q) \wedge (r \to s) \to ((p \wedge r) \to (q \wedge s))$. The truth table is ..... [9 marks]

p	q	r	s	$(p \to q) \wedge (r \to s)$	$(p \wedge r) \to (q \wedge s)$	s
T	T	T	T	T	T	T
T	T	T	F	F	F	T
T	T	F	T	T	T	T
T	T	F	F	T	T	T
T	F	T	T	F	F	T
T	F	T	F	F	F	T
T	F	F	T	F	T	T
T	F	F	F	F	T	T
F	T	T	T	T	T	T
F	T	T	F	F	T	T
F	T	F	T	T	T	T
F	T	F	F	T	T	T
F	F	T	T	T	T	T
F	F	T	F	F	T	T
F	F	F	T	T	T	T
F	F	F	F	T	T	T

Since s is always T, it is a tautology. ..... [1 mark]

Two compound statements that have the same truth values in all possible truth assignments are called *logically equivalent*. We can also define this notion as follows.

Definition 1.3.7. [Hinman, 2005, Definition 1.3.2] Two statements ϕ and ψ are called ***logically equivalent*** or ***tautologically equivalent*** if $\phi \leftrightarrow \psi$ is a tautology. We use notations $\phi \equiv \psi$ and $\phi \Leftrightarrow \psi$ to denote that ϕ and ψ are logically equivalent.

In the following examples, we will show that we can either use the truth table of $\phi \leftrightarrow \psi$ or comparing the truth tables of ϕ and ψ to determine if $\phi \equiv \psi$.

Example 1.3.8. Let p and q be two statement variables. Determine whether the following statements are logically equivalent or not.

1. $\sim p \vee \sim q$ and $\sim (p \vee q)$

Solution: By Definition 1.3.7, we can construct a truth table for $(\sim p \vee \sim q) \leftrightarrow (\sim (p \vee q))$ to determine if it is logically equivalent.

p	q	$\sim p$	$\sim q$	$p \vee q$	$\sim (p \vee q)$	$\sim p \vee \sim q$	$(\sim p \vee \sim q) \leftrightarrow (\sim (p \vee q))$
T	T	F	F	T	F	F	T
T	F	F	T	T	F	T	F
F	T	T	F	T	F	T	F
F	F	T	T	F	T	T	T

Since $(\sim p \vee \sim q) \leftrightarrow (\sim (p \vee q))$ is not a tautology, $\sim p \vee \sim q$ and $\sim (p \vee q)$ are not logically equivalent.

2. $p \to q$ and $\sim q \to \sim p$

Example 1.3.9 (Tutorial 1, Q7)**.** Answer true or false. An equivalent way to express

(a) the converse of "p is sufficient for q" is "p is necessary for q". ..... ☐

(b) the inverse of "p is necessary for q" is "$\sim p$ is sufficient for $\sim q$". ..... ☐

(c) the contrapositve of "p is necessary for q" is "$\sim q$ is necessary for $\sim p$". ..... ☐

Example 1.3.10 (Tutorial 1, Q8)**.** Determine whether the 2 statements forms are equivalent.

(a) $p \leftrightarrow q$ and $(p \wedge q) \vee (\sim p \wedge \sim q)$.

(b) $p \to (q \to r)$ and $(p \to q) \to r$.

(c) $(p \vee q) \to r$ and $(p \to r) \wedge (q \to r)$.

Mathematicians have identified the useful laws for simplifying statements based on the concept of logical equivalence. They are summarised in the following theorem.

Theorem 1.3.11 (**Laws of Logical Equivalences**). *Given any atomic statements p, q and r, the following logical equivalences hold [Epp, 2020].*

1. *Double negative law:*
   - $\sim(\sim p) \equiv p$.
2. *Idempotent laws:*
   - $p \wedge p \equiv p$;
   - $p \vee p \equiv p$.
3. *Universal bound laws:*
   - $p \vee \mathrm{T} \equiv \mathrm{T}$;
   - $p \wedge \mathrm{F} \equiv \mathrm{F}$.
4. *Identity laws:*
   - $p \wedge \mathrm{T} \equiv p$;
   - $p \vee \mathrm{F} \equiv p$.
5. *Negation laws:*
   - $p \vee \sim p \equiv \mathrm{T}$;
   - $p \wedge \sim p \equiv \mathrm{F}$.
6. *Commutative laws:*
   - $p \wedge q \equiv q \wedge p$;
   - $p \vee q \equiv q \vee p$.
7. *Absorption laws:*
   - $p \vee (p \wedge q) \equiv p$;
   - $p \wedge (p \vee q) \equiv p$.
8. *Associative laws:*
   - $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$;
   - $(p \vee q) \vee r \equiv p \vee (q \vee r)$.
9. *Distributive laws:*
   - $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$;
   - $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$.
10. *De Morgan's laws:*
    - $\sim(p \wedge q) \equiv \sim p \vee \sim q$;
    - $\sim(p \vee q) \equiv \sim p \wedge \sim q$.
11. *Implication law:*
    - $p \to q \equiv \sim p \vee q$
12. *Biconditional law:*
    - $p \leftrightarrow q \equiv (p \to q) \wedge (q \to p)$.

Proof: Since any statements can only be T or F, truth table can be used to prove 1. to 12. The following are just two demonstrations and the rest are left as exercises.

7. One of the absorption law

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

Since $\models (p \vee (p \wedge q)) \leftrightarrow p$, by definition, $p \vee (p \wedge q) \equiv p$.

11. Implication law

p	q	$\sim p$	$p \to q$	$\sim p \vee q$
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

The last two columns have the same truth values, hence the two statements are logical equivalent.

An important application of the logical equivalence laws is to write sentences in more readable form as demonstrated in the following example.

Example 1.3.12. Write the negation of the given statements:

1. John is smart but lazy.
   In $\sim (p \wedge q)$ form: ..... John is *not* "smart and lazy".
   In equivalent form $\sim p \vee \sim q$: ..... John is not smart or he is not lazy.
2. $2 \leq \sqrt{2}$
   In $\sim (p \vee q)$ form: ..... $2 \not\leq \sqrt{2}$
   In equivalent form $\sim p \wedge \sim q$: .....

Note that the statement $\sim (p \wedge q)$ is called the *negation of conjunction of p and q* while the statement $\sim (p \vee q)$ is called the *negation of disjunction of p and q*.

Example 1.3.13 (Tutorial 1, Q9)**.** Rewrite the following statements in (the logically equivalent) if-then form.

(a) Fix my ceiling or I won't pay my rent.
(b) Study hard or I won't pass Discrete Mathematics.
(c) Catching the 7am bus is a sufficient condition for my being on time for school.
(d) Doing homework regularly is a necessary condition to pass the course.
(e) Ali studies calculus only if he is a math major.
(f) P is a square only if P is a rectangle.
(g) n is divisible by 6 is a sufficient condition for n to be divisible by 2 and n is divisible by 3.

To make the laws of logical equivalences useful, mathematicians have shown that the logical equivalence "$\equiv$" is an equivalence relation (see Set Relations in later topic). We also require logically equivalent statements to be logically equivalent under substitution [Chiswell and Hodges, 2007, Herrmann, 29 Jan 2006].

Definition 1.3.14 (Uniform Substitution)**.** If ϕ, ψ, ψ_i are statements/formulas, and p_i are (atomic) propositional variables, then $\phi[\psi/p_i]$ denotes the result of replacing each occurrence of p_i by an occurrence of ψ in ϕ; similarly, the *simultaneous substitution* S of $p_1, \cdots, p_n$ by formulas $\psi_1, \cdots, \psi_n$ is denoted by $\psi_1/p_1, \cdots, \psi_n/p_n$.

Theorem 1.3.15 (Substitution Theorem)**.** *Let S be a simultaneous substitution (Definition 1.3.14) and $\phi \equiv \psi$. Then $\phi[S] \equiv \psi[S]$. More generally, let ϕ_i and ψ_i be statements such that $\phi_i \equiv \psi_i$, $1 \leq i \leq n$. Then $\phi[\phi_1/p_1, \cdots, \phi_n/p_n] \equiv \psi[\psi_1/p_1, \cdots, \psi_n/p_n]$.*

Proof. See Chiswell and Hodges [2007, Theorem 3.7.6]. □

Example 1.3.16. Show each logical equivalence below using laws of logical equivalences (Theorem 1.3.11).

1. $(p \vee q) \wedge ((p \vee q) \vee r) \wedge \sim (\sim (p \vee q)) \equiv p \vee q$

   Proof: To illustrate the power of the substitution, we note that a substitution of $p \vee q$ for p and r for q in absorption law gives

   $$\underline{(p \vee q) \wedge ((p \vee q) \vee r)} \equiv \underline{p \vee q}.$$

   This leads to the first step in the following logical equivalence, the rest of the steps are of similar reasoning.

   $$\begin{aligned} \underline{(p \vee q) \wedge ((p \vee q) \vee r)} \wedge \sim (\sim (p \vee q)) &\equiv \underline{(p \vee q)} \wedge \sim (\sim (p \vee q)) && \text{(Absorption law)} \\ &\equiv (p \vee q) \wedge (p \vee q) && \text{(Double negative law)} \\ &\equiv p \vee q && \text{(Idempotent law)} \end{aligned}$$

2. $(p \vee q) \wedge \sim (\sim p \wedge q) \equiv p$

3. $\sim [\sim ((p \vee q) \wedge r) \vee \sim q] \equiv q \wedge r$

Example 1.3.17 (Tutorial 1, Q10)**.** Explain why the statement "If today is not cold, then today is cold" is logically equivalent to the statement "Today is cold".

Example 1.3.18 (Tutorial 1, Q11)**.** (a) Show that the following statements are all logically equivalent.

$$p \to (q \vee r), \quad (p \wedge \sim q) \to r \quad \text{and} \quad (p \wedge \sim r) \to q.$$

(b) Use the logical equivalences in (a) to rewrite the sentence "If n is prime, then n is odd or n is 2." in 2 different ways.

The laws of logical equivalence can be used to simplify some long statements to logically equivalent shorter statements as demonstrated below.

Example 1.3.19. Simplify the following statement to a statement with no more than 3 logical connectives by stating the law used in each step of the simplification:

$$(p \vee q) \wedge p \wedge (r \vee q) \wedge (p \vee \sim p \vee r) \wedge (r \wedge \sim q).$$

Solution:

$(p \vee q) \wedge p \wedge (r \vee q) \wedge (p \vee \sim p \vee r) \wedge (r \wedge \sim q)$	
$\equiv (p \vee q) \wedge p \wedge (r \vee q) \wedge (T \vee r) \wedge (r \wedge \sim q)$	(Negation)
$\equiv (p \vee q) \wedge p \wedge (r \vee q) \wedge T \wedge (r \wedge \sim q)$	(Commutative and Universal Bound)
$\equiv p \wedge [(r \vee q) \wedge (r \wedge \sim q)]$	(Absorption, Identity)
$\equiv p \wedge [(r \wedge (r \wedge \sim q)) \vee (q \wedge (r \wedge \sim q))]$	(Distributive)
$\equiv p \wedge [((r \wedge r) \wedge \sim q) \vee (q \wedge (\sim q \wedge r))]$	(Associative, Commutative)
$\equiv p \wedge [(r \wedge \sim q) \vee (F \wedge r)]$	(Idempotent, Associative, Negation)
$\equiv p \wedge [(r \wedge \sim q) \vee F]$	(Universal bound)
$\equiv p \wedge (r \wedge \sim q)$	(Identity)

Example 1.3.20 (Tutorial 1, Q12)**.** Use the laws of logical equivalence to show the following:

(a) $(p \wedge (\sim (\sim p \vee q))) \vee (p \wedge q) \equiv p$.

(b) $\sim (p \vee \sim q) \vee (\sim p \wedge \sim q) \equiv \sim p$.

(c) $\sim ((\sim p \wedge q) \vee (\sim p \wedge \sim q)) \vee (p \wedge q) \equiv p$.

(d) $(\sim p \vee \sim q) \rightarrow (p \wedge q \wedge r) \equiv p \wedge q$.

Example 1.3.21 (Tutorial 1, Q13)**.** Simplify the following compound statement to a statement with no more than 3 logical connectives involving $\sim$, $\vee$ and $\wedge$:

$$[[[(p \wedge q) \wedge r] \vee [(p \wedge q) \wedge \sim r]] \vee \sim q] \rightarrow s.$$

Example 1.3.22 (Tutorial 1, Q14)**.** Verify that

(a) $(p \rightarrow q) \wedge [\sim q \wedge (r \vee \sim q)] \equiv \sim (p \vee q)$

(b) $p \vee q \vee (\sim p \wedge \sim q \wedge r) \equiv p \vee q \vee r$

(c) $(p \leftrightarrow q) \wedge (q \leftrightarrow r) \wedge (r \leftrightarrow p) \equiv (p \rightarrow q) \wedge (q \rightarrow r) \wedge (r \rightarrow p)$

Solution: One can use truth table to prove that they are logically equivalent, this is simpler compare to using laws of logical equivalence as follows (let the right hand side be ψ and the associative law and commutative law are sometimes used without explicitly mentioned).
First, we reduce the right-hand-side to an equivalent form involving only negation, conjunction and disjuction.

$$\begin{aligned}
\psi &\equiv \underline{(\sim p \vee q) \wedge (\sim q \vee r)} \wedge (\sim r \vee p) && \text{[implication law]} \\
&\equiv (\sim p \wedge \sim q \vee \sim p \wedge r \vee \underline{q \wedge \sim q} \vee q \wedge r) \wedge (\sim r \vee p) && \text{[distributive law]} \\
&\equiv (\sim p \wedge \sim q \vee \sim p \wedge r \vee \underline{q \wedge r}) \wedge (\sim r \vee p) && \text{[negation \& identity laws]} \\
&\equiv (\sim p \wedge \sim q \wedge \sim r \vee \sim p \wedge \sim q \wedge p \vee \sim p \wedge r \wedge \sim r \vee \sim p \wedge r \wedge p \\
&\quad \vee q \wedge r \wedge \sim r \vee q \wedge r \wedge p) && \text{[distributive law]} \\
&\equiv (\sim p \wedge \sim q \wedge \sim r) \vee (p \wedge q \wedge r) && \text{[negation \& identity laws]}
\end{aligned}$$

$(p \leftrightarrow q) \wedge (q \leftrightarrow r) \wedge (r \leftrightarrow p)$

$\equiv (p \to q) \wedge (q \to p) \wedge (q \to r) \wedge (r \to q) \wedge (r \to p) \wedge (p \to r)$ [biconditional law]

$\equiv \underline{(p \to q) \wedge (q \to r) \wedge (r \to p)} \wedge (q \to p) \wedge (r \to q) \wedge (p \to r)$ [associative & commutative laws]

$\equiv \psi \wedge \underline{(\sim q \vee p) \wedge (\sim r \vee q)} \wedge (\sim p \vee r)$ [implication law]

$\equiv \psi \wedge (\sim q \wedge \sim r \vee \underline{\sim q \wedge q} \vee p \wedge \sim r \vee p \wedge q) \wedge (\sim p \vee r)$ [distributive law]

$\equiv \psi \wedge (\sim q \wedge \sim r \vee \underline{F} \vee p \wedge \sim r \vee p \wedge q) \wedge (\sim p \vee r)$ [negation law]

$\equiv \psi \wedge (\sim q \wedge \sim r \vee p \wedge \sim r \vee p \wedge q) \wedge (\sim p \vee r)$ [identity law]

$\equiv \psi \wedge (\underline{\sim q \wedge \sim r \wedge \sim p} \vee \underline{\sim q \wedge \sim r \wedge r}$

$\vee \underline{p \wedge \sim r \wedge \sim p} \vee \underline{p \wedge \sim r \wedge r} \vee \underline{p \wedge q \wedge \sim p} \vee \underline{p \wedge q \wedge r})$ [distributive law]

$\equiv \psi \wedge (\underline{\sim q \wedge \sim r \wedge \sim p} \vee F \vee F \vee F \vee F \vee \underline{p \wedge q \wedge r})$ [negation law]

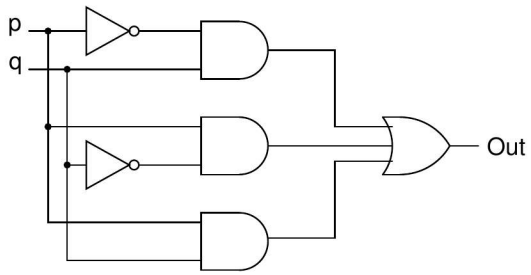
$\equiv \psi \wedge (\underline{\sim p \wedge \sim q \wedge \sim r} \vee \underline{p \wedge q \wedge r}) \equiv \psi \wedge \psi = \psi$ [identity law]

Hence the two statements are logically equivalent.

§1.4 Applications: Logic Circuits, Solving Logic Puzzles, etc.

Logical equivalence is used in logic circuit design [Nisan and Schocken, 2005] and in digital signal processing systems [Woods et al., 2008] to identify logical equivalent circuits which are minimal to save power (larger circuits use more power).

Example 1.4.1. Simplify the logic diagram below.



For the names of the symbols, see Nisan and Schocken [2005, Chapter 1].

Solution: Step 1: Obtain the formula of the circuit: $Out = (\sim p \wedge q) \vee (p \wedge \sim q) \vee (p \wedge q)$
Step 2: Simplify the above formula:

$\text{Out} = (\sim p \wedge q) \vee (p \wedge (\sim q \vee q))$ (Distributive Law)

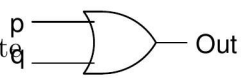
$\equiv (\sim p \wedge q) \vee (p \wedge T)$ (Negation Law)

$\equiv (\sim p \wedge q) \vee p$ (Identity Law)

$\equiv (\sim p \vee p) \wedge (q \vee p)$ (Distributive Law)

$\equiv T \wedge (p \vee q)$ (Negation Law & Commutative Law)

$\equiv p \vee q$ (Identity Law)

Step 3: The above circuit is equivalent to an or-gate 

Example 1.4.2 (Tutorial 1, Q15)**.** In logic circuit design, one of the basic logic gate is the NAND gate. It is logically equivalent to $\sim (p \wedge q)$ and denoted by $(p \uparrow q)$ for any statements p and q.

(a) Represent the logic gates (i) $\sim p$ and (ii) $p \to q$ using the NAND gate.

(b) Are $p \uparrow (q \uparrow r)$ and $(p \uparrow q) \uparrow r$ logically equivalent?

We will introduce generalised concepts related to the "truth assignment" from Definition 1.2.2. Note that we will use the notion of **formula** (which includes propositions and predicates) rather than just proposition.

Definition 1.4.3. [Schöning and Torán, 2013] A formula ϕ is called **satisfiable** if there exists an assignment v such that $v(\phi) = \mathrm{T}$. Otherwise, ϕ is called **unsatisfiable** (or contradiction as defined above).

The set of all satisfiable formulas is denoted by SAT. A satisfying assignment v for a formula ϕ is often also called a **model** for ϕ, in which case we write: $v \models \phi$. (Generation of Definition 1.2.2).

A formula ϕ is called **valid** or **tautology** if, for all assignments v, $v(\phi) = \mathrm{T}$. The set of tautologies is denoted by TAUT. The notation $\models \phi$ expresses that ϕ is a tautology.

A formula ψ **follows** from ϕ (or, ψ is a **consequence** from ϕ, denoted by $\phi \models \psi$, if $\phi \to \psi$ (or $\sim \phi \vee \psi$) is a tautology.

Based on the definition, ϕ is a **tautology** iff $\sim \phi$ is **unsatisfiable**; ϕ is **satisfiable** iff $\sim \phi$ is not a tautology.

Apart from the truth table method, another method to check for satisfiability is the **tableaux method**, which checks a proposition ϕ is satisfiable by expanding ϕ to conjunction and disjunction forms using laws of logical equivalences and identify "open" branches.

Example 1.4.4. Use tableaux method to check the satisfiability of the proposition $\sim ((p \wedge \sim r) \vee (\sim q \wedge r))$.

Solution:

1.	$\sim ((p \wedge \sim r) \vee (\sim q \wedge r))$	Hypothesis
2.	$(\sim (p \wedge \sim r)) \wedge (\sim (\sim q \wedge r))$	De Morgan
3.	$\sim p \vee r$	2 Conjunction & De Morgan
4.	$q \vee \sim r$	2 Conjunction & De Morgan
5.	$\sim p$ $\quad$ r	3 disjunction
6.	q $\quad$ $\sim r$ $\quad$ q $\quad$ $\sim r$	4 disjunction

$\otimes$
$5, 6$

From the right tree, we can see that a valuation $v(q) = \mathrm{T}$ and $v(r) = \mathrm{T}$ to make $v(\sim p \vee r) = \mathrm{T}$ and $v(q \vee \sim r) = \mathrm{T}$. Hence the hypothesis is satisfiable.

From the example above, we can see that **encoding** propositions into propositions involving only negation, conjunction and disjunction is the way to solve **satisfiability problem** (it is important because it is at the crossroads of logic, graph theory, computer science, computer engineering, and operations research). CNF is used in the computerisation of **SAT solver** as well as the generalisation to **SMT (Satisfiability Modulo Theories)** solvers.

Definition 1.4.5. A **literal** is an atom p or a negated atom $\sim p$.

A **clause** is a disjunction of **literals**.

A formula ϕ is in **conjunctive normal form (CNF)** if it is a conjunction of clauses:

$$F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$$

where $C_i = u_{i,1} \vee u_{i,2} \vee \cdots \vee u_{i,k_i}$, $i = 1, ..., m$ are clauses, $u_{i,j}$ are literals. If $|C_i| = k_i \leq k$ for some constant k for all i, ϕ is said to be in k-CNF.

A formula ϕ is in **disjunctive normal form (DNF)** if it is a disjunction of conjunctions of literals.

Example 1.4.6. $(p \vee \sim q) \wedge (\sim p \vee r) \wedge (q \vee r)$ is a CNF.

Example 1.4.7. The proposition $\sim ((p \wedge \sim r) \vee (\sim q \wedge r))$ from Example 1.4.4 has a logically equivalent CNF:

$$(\sim p \vee q) \wedge (q \vee \sim r).$$

Exercise: Feel free to check if the lecturer is correct by using truth table or comparison table.

Theorem 1.4.8. *For every proposition ϕ, there exists a CNF formula which is equivalent to ϕ. However, finding the equivalent CNF usually requires exponential computation time.*

Theorem 1.4.9 (Tseitin)**.** *For every proposition ϕ, there exists a sat-equivalent formula ψ in 3-CNF. The transformation from ϕ to ϕ can be done efficiently in polynomial time.*

When a formula ϕ is transformed into an equivalent conjunctive normal form (CNF), the **DPLL algorithm**, named for its developers Davis, Putnam, Logemann, and Loveland, is an efficient approach to solving **boolean satisfiability (SAT) problems**.

```
1: function DPLL(Φ)
2:     if φ=true then return true
3:     if φ contains a false clause then return false
4:     for all unit clauses ℓ in φ do
5:         φ ← UNIT-PROPAGATE(ℓ, φ)
6:     for all literals ℓ occurring pure in φ do
7:         φ → PURE-LITERAL-ASSIGN(ℓ, φ)
8:     ℓ ← CHOOSE-LITERAL(φ) return DPLL(φ ∧ ℓ) ∨ DPLL(φ∧ ∼ ℓ)
```

There are many SMT solvers. `https://en.wikipedia.org/wiki/Z3_Theorem_Prover`, developed by Microsoft, is one of the most popular and will be used in this course.

Example 1.4.10. Use the Z3 Theorem Prover to solve the SAT problem for the proposition $\sim ((p \wedge \sim r) \vee (\sim \wedge r))$ from Example 1.4.4

Solution: A program using Z3 SMT solver to solve SAT problem from Example 1.4.4 is shown below.

```
;;; Example 1.3.26 and 1.3.30: ~((p & ~r) v (~q & r))
(declare-const p Bool)
(declare-const q Bool)
(declare-const r Bool)
(define-fun proposition () Bool
  (not (or (and p (not r)) (and (not q) r))))
(assert proposition)
(check-sat)
(get-model)
(exit)
```

The truth assignment found by Z3 SMT solver is shown below.

```
sat
```

```
(
  (define-fun p () Bool
    false)
  (define-fun q () Bool
    false)
  (define-fun proposition () Bool
    (not (or (and p (not r)) (and (not q) r))))
  (define-fun r () Bool
    false)
)
```

Although SAT/SMT solvers use CNF in the algorithm, many logical puzzles are expressed in DNF as illustrated in the following example.

Example 1.4.11. Four candidates A, B, C and D are to be selected to participate in a chess competition. However, only two candidates will be chosen in the final list and the following constraints must be obeyed:

- Only one from A and B will participate.
- If C participates, then D will also need to participate.
- At most one of B and D will participate.
- If D is not participating, then A is also not participating.

Solution: Let a, b, c, d denote the statements A, B, C, D participates in the competition respectively. Then the constraints can be written as

$$[(a\wedge\sim b)\vee(\sim a\wedge b)]\wedge[c\to d]\wedge\sim(b\wedge d)\wedge[\sim d\to\sim a]$$

Let us denote it by p and we can either construct a truth table or use the law of logical equivalence to simplify the expressions to involving negations and conjunctions.

$$\begin{aligned}
p &\equiv [(a\wedge\sim b)\vee(\sim a\wedge b)]\wedge\underline{[\sim c\vee d]}\wedge[\sim b\vee\sim d]\underline{\wedge[d\vee\sim a]}\\
&\equiv [(a\wedge\sim b)\vee(\sim a\wedge b)]\wedge[(\sim c\wedge\sim a)\vee d]\wedge[\sim b\vee\sim d]\\
&\equiv [(a\wedge\sim b)\vee(\sim a\wedge b)]\wedge[\underline{(\sim c\wedge\sim a)\wedge[\sim b\vee\sim d]}\vee\underline{d\wedge[\sim b\vee\sim d]}]\\
&\equiv [(a\wedge\sim b)\vee(\sim a\wedge b)]\wedge[\underline{\sim a\wedge\sim c\wedge\sim b}\vee\underline{\sim a\wedge\sim c\wedge\sim d}\vee\underline{\sim b\wedge d}\vee\underline{\sim d\wedge d}]\\
&\equiv [(a\wedge\sim b)\vee(\sim a\wedge b)]\wedge[\underline{\sim a\wedge\sim b\wedge\sim c}\vee\underline{\sim a\wedge\sim c\wedge\sim d}\vee\underline{\sim b\wedge d}]\\
&\equiv \underline{a\wedge\sim b\wedge\sim a\wedge\sim b\wedge\sim c}\vee\underline{a\wedge\sim b\wedge\sim a\wedge\sim c\wedge\sim d}\vee\underline{a\wedge\sim b\wedge\sim b\wedge d}\\
&\quad\vee\underline{\sim a\wedge b\wedge\sim a\wedge\sim b\wedge\sim c}\vee\underline{\sim a\wedge b\wedge\sim a\wedge\sim c\wedge\sim d}\vee\underline{\sim a\wedge b\wedge\sim b\wedge d}\\
&\equiv F\vee F\vee\underline{a\wedge\sim b\wedge d}\vee F\vee\underline{\sim a\wedge b\wedge\sim c\wedge\sim d}\vee F
\end{aligned}$$

Since we need two participants (so $\sim a\wedge b\wedge\sim c\wedge\sim d$ is not useful because only B goes), they are A and D.

```
;;; Example 1.3.32: participate = true; not participate = false
(declare-const A Bool)
(declare-const B Bool)
(declare-const C Bool)
(declare-const D Bool)
(assert (or (and A (not B)) (and (not A) B)))    ; condition 1
(assert (implies C D))                           ; condition 2
(assert (not (and B D)))
(assert (implies (not D) (not A)))
(check-sat)
(get-model)
```

```
(exit)
```

The output matches our mathematical analysis above.

```
sat
(
  (define-fun A () Bool
    false)
  (define-fun D () Bool
    false)
  (define-fun B () Bool
    true)
  (define-fun C () Bool
    false)
)
```

§1.5 Logical Implication and Argument for Statements (Topic 2a)

Logical equivalence (Definition 1.3.7) is defined in terms of the connective of "ordinary" equivalence $\leftrightarrow$. Analogously, the notion of *logical implication* can be defined in terms of "ordinary" implication "$\rightarrow$".

Definition 1.5.1. We say that a statement ϕ ***logically implies*** a statement ψ or ψ is a *tautological consequence* of ϕ [Hinman, 2005, Definition 1.3.2], denoted by $\phi \Rightarrow \psi$, when the statement $\phi \rightarrow \psi$ is a tautology, i.e. $\models \phi \rightarrow \psi$. When this happens, we sometimes say that the statement ϕ is *stronger* then ψ.

Remark 1.5.2. The distinction between implication and logical implication is fundamental. Whereas $p \rightarrow q$ is a compound statement, the statement $p \Rightarrow q$ describes a relationship between two compound statement p and q, it is a ***meta language***!!!

The link between logical equivalence and logical implication is as follows.

Theorem 1.5.3. *Let ϕ and ψ be two statements. Then $\phi \equiv \psi$ iff $\phi \Rightarrow \psi$ and $\psi \Rightarrow \phi$.*

By definition, to show that "$\phi \Rightarrow \psi$", i.e. a statement ϕ logically implies another statement ψ, we can create the truth table for the statement $\phi \rightarrow \psi$ and examine its last column.

Example 1.5.4. Show that $p \wedge q \Rightarrow p \vee q$.

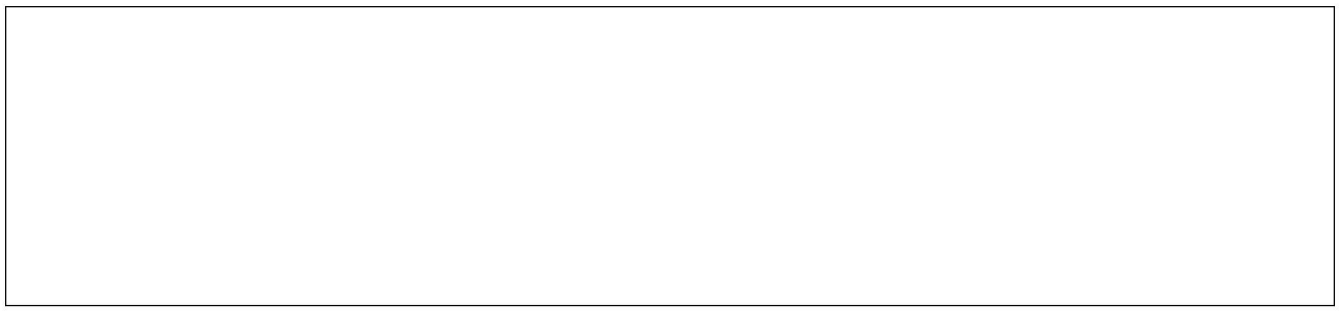
Proof: The corresponding truth table is

p	q	$p \wedge q$	$p \vee q$	$p \wedge q \rightarrow p \vee q$
T	T	T	T	T
T	F	F	T	T
F	T	F	T	T
F	F	F	F	T

Since the last column consists entirely of T's, the implication $p \wedge q \rightarrow p \vee q$ is a *tautology*, i.e. $\models p \wedge q \rightarrow p \vee q$ so that $p \wedge q \Rightarrow p \vee q$.

Remark 1.5.5. For statements related to logical implication, the final step of the truth table is always to evaluate the implication. The *comparison table*, which is a striped version of a truth table, lacks the last column since for implication, can be used. We know that as long as there is no $T \rightarrow F$, we know that the implication will be true.

Example 1.5.6. Show that $\sim p \Rightarrow p \rightarrow q$ using *comparison table*.



An argument is an assertion that the conjunction of several statements implies another statement as defined below.

Definition 1.5.7. For any statements ϕ_1, ϕ_2, $\cdots$, ϕ_n and ψ,

$$\{\phi_1,\ \phi_2,\ \cdots,\ \phi_n\} \ / \therefore \psi \quad \text{or} \quad \begin{array}{l} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_n \\ \hline \therefore\ \psi \end{array}$$

are called ***arguments*** (or *deductive argument*, see `https://en.wikipedia.org/wiki/Deductive_reasoning`). We call $\{\phi_1, \phi_2, \cdots, \phi_n\}$ the set of ***hypotheses***, ***premises*** or ***assumptions*** and ψ the ***conclusion*** of the argument. It is customary to ignore the "bracket" for convenience [Herrmann, 29 Jan 2006].

When there is no hypothesis, an argument is written as $/ \therefore \psi$.

Definition 1.5.8. When the premises ϕ, ϕ_1, ϕ_2, $\cdots$ and ϕ_n are true, ψ is true, then we say the argument is ***valid***. ψ is said to be *deduced* or *inferred* from the premises, and that ψ *follows logically from* or is a *logical consequence* of the premises or that ψ. An argument that is not valid is said to be ***invalid*** or *fallacious*.

Three methods for determining the validity of an argument are

Method 1 : using "truth table" or "comparison table";

Method 2 : using laws of equivalences and implications;

Method 3 : `https://en.wikipedia.org/wiki/Method_of_analytic_tableaux` or "truth tree".

Method 1: In a "truth table", an argument is valid if for all truth assignments, the truth value of

$$\phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n \rightarrow \psi$$

are all true, i.e. the above proposition is a tautology. Otherwise, the argument is invalid. In a "comparison table", an argument is valid when there is no truth assignment such that ψ is false but ϕ_1, ϕ_2, $\cdots$ and ϕ_n are true. If there is **one case** where ϕ_1, ϕ_2, $\cdots$ and ϕ_n are true but ψ is false, the argument is invalid. A mathematical formalism of method 1 is given by the deduction theorem below.

Theorem 1.5.9 (Deduction Theorem)**.** *Let Γ be any finite (possible empty) set of formulas, ϕ, ϕ_1, $\cdots$, ϕ_n and ψ be formulas [Herrmann, 29 Jan 2006, Section 2.8].*

1. $\Gamma,\ \phi \models \psi$ *iff* $\Gamma \models \phi \rightarrow \psi$.
2. $\phi_1,\ \cdots, \phi_n \models \psi$ *iff* $\phi_1 \wedge \cdots \wedge \phi_i,\ \cdots, \phi_n \models \psi$ *where* $1 < i \leq n$.

Based on this theorem, an argument $\phi_1,\ \cdots, \phi_n \ / \therefore \psi$ is valid iff $\models \phi_1 \wedge \cdots \wedge \phi_n \rightarrow \psi$; otherwise, the argument is invalid.

Example 1.5.10. Determine if $p \vee q / \therefore p \wedge q$ is valid.

Solution 1: The truth table for the argument is listed below.

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

According to the last column the statement $(p \vee q) \to (p \wedge q)$ is not a tautology. By definition, the argument is invalid.

Solution 2: We don't need to write down the whole comparison table to confirm that the argument is invalid, we only need to find **one case** where the conclusion $p \wedge q$ is false but the premise $p \vee q$ is true.

Suppose $v(p \wedge q) = \text{F}$, then we can have $v(p) = \text{T}$ and $v(q) = \text{F}$. However, $v(p \vee q) = \text{T}$. We found one case where the premise is true but the conclusion is false. Hence the argument is invalid!

Example 1.5.11. Determine the validity of the argument $p \vee (q \vee r),\ \sim r / \therefore p \vee q$.

Solution: We will use the comparison table to check the validity of the argument.

p	q	r	$p \vee (q \vee r)$	$\sim r$	$p \vee q$
T	T	T	T	F	T
T	T	F	T	T	T
T	F	T	T	F	T
T	F	F	T	T	T
F	T	T	T	F	T
F	T	F	T	T	T
F	F	T	T	F	F
F	F	F	F	T	F

In each situation where the *premises are both true*, the *conclusion is also true*, so the argument is valid.

Example 1.5.12. Determine the validity of the argument $p \to q \vee \sim r,\ q \to p \wedge r\ / \therefore p \to r$ by using the comparison table.

Example 1.5.13 (Tutorial 1, Q16)**.** Use comparison tables to determine whether the arguments are valid.

(a) $p \to (q \vee r),\ \sim q \vee \sim r / \therefore \sim p \vee \sim r$

(b)

$(p \wedge q) \rightarrow \sim r$
$p \vee \sim q$
$\sim q \rightarrow p$
$\therefore \quad \sim r$

Some sequence of English sentences with logical meaning and behaves like statements can be formalised and be analysed as "arguments" as illustrated in the examples below.

Example 1.5.14. Is the following argument valid?

If interest rates are going up, stock market prices will go down.
Interest rates are not going up.
Therefore stock market prices will not go down.

Solution: Let p: "Interest rates are going up"; q: "stock market prices will go down". The argument can be written formally as $p \rightarrow q,\ \sim p\ /\ \therefore \sim q$.

It is not valid since the **third row** of the comparison table indicates that we have a case where the premises are true but the conclusion is false:

p	q	$p \rightarrow q$	$\sim p$	$\sim q$
T	T	T	F	F
T	F	F	F	T
F	T	T	T	F
F	F	T	T	T

Example 1.5.15 (Tutorial 1, Q17(a))**.** Write the symbolic form of each argument below and then determine its validity.

If Tom is not on team A, then Hua is on team B.
If Hua is not on team B, then Tom is on team A.
Therefore Tom is not on Team A or Hua is not on Team B.

Example 1.5.16. Show that the argument $\sim p \to F / \therefore p$ is valid using truth table.

Similar to the laws of logical equivalences (Theorem 1.3.11), the laws of logical implications can be obtained using the truth table method.

Theorem 1.5.17 (Laws of Logical Implications)**.** *Let F be contradiction. Given any atomic statements p, q and r, the following arguments are valid [Epp, 2020].*

1. *Modus Ponens (MP in short):*
   $$p \to q,\ p \models q$$
2. *Modus Tollens (MT in short):*
   $$p \to q,\ \sim q \models \sim p$$
3. *Generalisation:*
   $$p \models p \vee q$$
   $$q \models p \vee q$$
4. *Specialisation:*
   $$p \wedge q \models p$$
   $$p \wedge q \models q$$
5. *Conjunction:*
   $$p,\ q \models p \wedge q$$
6. *Elimination:*
   $$p \vee q,\ \sim q \models p$$
   $$p \vee q,\ \sim p \models q$$
7. *Transitivity:*
   $$p \to q,\ q \to r \models p \to r$$
8. *Contradiction Rule:*
   $$\sim p \to F \models p$$

To apply the laws of logical implications to general propositions, we need the following theorems [Herrmann, 29 Jan 2006, Section 2.8].

Theorem 1.5.18 (Substitution of Equivalence)**.** *If $\xi \equiv \phi_n$ and $\phi_1, \cdots, \phi_{n-1}, \phi_n \models \psi$, then $\phi_1, \cdots, \phi_{n-1}, \xi \models \psi$.*

Theorem 1.5.19 ("Partial Ordering" Theorem)**.** *Let ϕ_i, ψ_j, ξ be formulas.*

1. $\phi_1, \cdots, \phi_n \models \phi_i$ *for each* $i = 1, \cdots, n$.
2. *If* $\phi_1, \cdots, \phi_i \models \psi_j$, *where* $j = 1, \cdots, p$, *and* $\psi_1, \cdots, \psi_p \models \xi$, *then* $\phi_1, \cdots, \phi_n \models \xi$.

Method 2: When an argument is valid, most of the time, it is possible to use on the laws of logical equivalences, laws of logical implication and the various substitution theorems, to show that the conclusion follows from the premises using the laws. However, this method does not directly allow one to check for invalidity. One can use laws of logical equivalences on $\phi_1 \wedge \cdots \wedge \phi_n \rightarrow \psi$ and perform forward checking to find possible assignments to make it false.

Example 1.5.20. Show that the hypotheses

- ϕ_1: "If John takes the computer course, then John stays in the hostel"
- ϕ_2: "John does not stay in the hostel"
- ϕ_3: "If John does not take the computer course, then John takes the language course or stay at home"
- ϕ_4: "If John takes language course then John buys a motorcycle"
- ϕ_5: "If John buys a car, then John does not buy motorcycle"
- ϕ_6: "John buys a car"

lead to the conclusion ξ: "John stays at home".

Solution: Let

r_1: John takes the computer course
r_2: John stays in the hostel
r_3: John takes the language course
r_4: John stays at home
r_5: John buys a motorcycle
r_6: John buys a car

The above argument can be formally written as

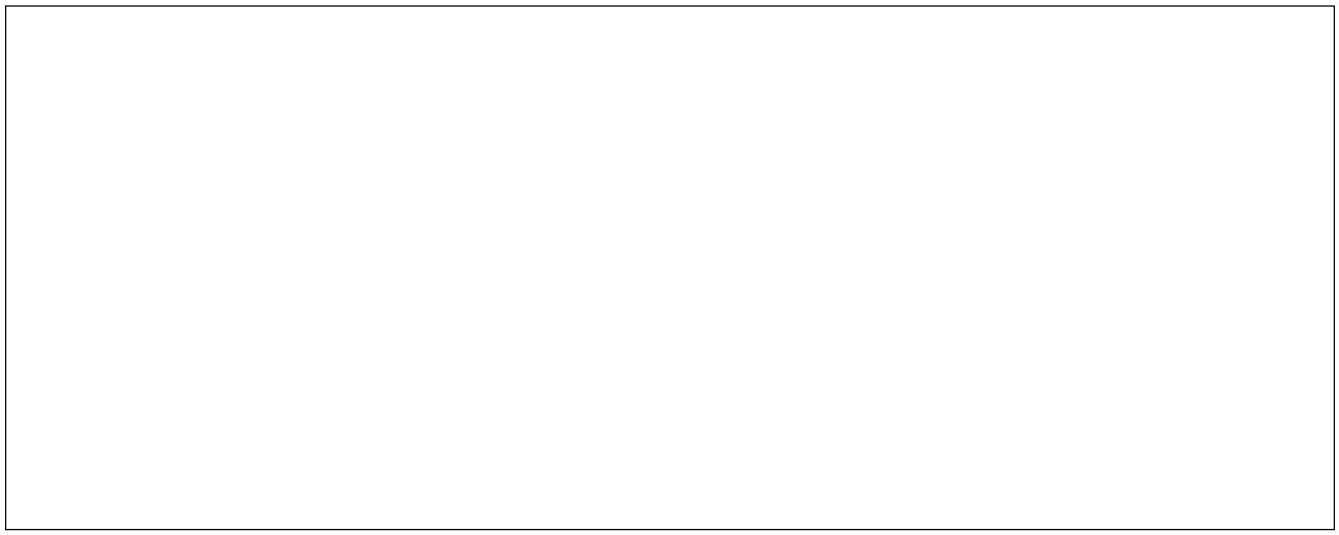
$$r_1 \rightarrow r_2,\ \sim r_2,\ \sim r_1 \rightarrow r_3 \vee r_4,\ r_3 \rightarrow r_5,\ r_6 \rightarrow \sim r_5,\ r_6 \ / \therefore r_4.$$

Instead of using a comparison table, we use the logical equivalence and logical implication rules together with the "Partial Ordering" Theorem to show that the above argument is valid.

ϕ_1 :	$r_1 \rightarrow r_2$	premise
ϕ_2 :	$\sim r_2$	premise
ϕ_3 :	$\sim r_1 \rightarrow r_3 \vee r_4$	premise
ϕ_4 :	$r_3 \rightarrow r_5$	premise
ϕ_5 :	$r_6 \rightarrow \sim r_5$	premise
ϕ_6 :	r_6	premise
ψ_1 :	$\sim r_1$	$\phi_1,\ \phi_2,$ Modus Tollens
ψ_2 :	$r_3 \vee r_4$	$\phi_3,\ \psi_1,$ Modus Ponens
ψ_3 :	$\sim r_5$	$\phi_5,\ \phi_6,$ Modus Ponens
ψ_4 :	$\sim r_3$	$\phi_4,\ \psi_3,$ Modus Tollens
ξ :	r_4	$\psi_2,\ \psi_4,$ Elimination

Example 1.5.21 (Tutorial 1, Q17(b))**.** Write the symbolic form of the argument below and then show that it is valid using laws of logical implications.

If I graduate this semester, then I will have passed Calculus.
If I do not study Calculus for 5 hours a week, then I will not pass Calculus.
If I study Calculus for 5 hours a week, then I cannot play basketball.
Therefore, if I play basketball, I will not graduate this semester.



§1.6 Tableaux: Using Diagram to Check Validity

Method 1 (truth table or comparison table) is not applicable to propositions with too many atomic variables while method 2 (logical laws) cannot be used to show that an argument is invalid. Method 3 (tableaux) can be used to check relatively large arguments.

<u>Method 3</u>: **Tableau** (the plural form is **tableaux**) or **truth tree** can be used to check the validity of an argument by examining **the premises and the negation of the conclusion**. If they are **satisfiable**, the argument is invalid; If they are **unsatisfiable** or a *contradiction*, the argument is valid (recall Definition 1.4.3).

A **tableaux** proof is done by constructing a **tableau** or **truth tree** of which the nodes are labelled by formulas of the logic. In particular, every proof is begun with a tree consisting of a single node and then expanded by applying the procedure (`https://en.wikipedia.org/wiki/Method_of_analytic_tableaux`):

1. Pick an open leaf node. The leaf node in the initial chain is marked open.
2. Pick an applicable node on the branch above the selected node.
3. Apply the applicable node, which corresponds to expanding the tree below the selected leaf node based on expansion rule.

1.	$P \wedge Q$	
2.	P	1.
3.	Q	1.

1.	$P \vee Q$		
2.	P	Q	1.

1.	E	
2.	logical equivalent form	1.
3.	of E from page 23	

where E can be $\sim (P \wedge Q)$ (De Morgan), $\sim (P \vee Q)$ (De Morgan), $\sim (\sim P)$ (double negative), $P \to Q$ (implication), $\sim (P \to Q)$ (implication + De Morgan + double negative), $P \leftrightarrow Q$ (biconditional), $\sim (P \leftrightarrow Q)$ (biconditional + De Morgan).

4. For every newly created node that is both a literal/negated literal, and whose complement appears in a prior node on the same branch, mark the branch as closed. Mark all other newly created nodes as open.

As mentioned in the Section 1.4, tableaux method is tightly link to SAT/SMT solvers. So we will illustrate the use of tableaux method with Z3 SMT solver in the following examples.

Example 1.6.1. Determine the validity of the argument $p \vee (q \vee r), \sim r / \therefore p \vee q$ from Example 1.5.11 using truth tree/tableaux method and a SMT (Satisfiability Modulo Theories) solver.

Solution:

1.	$p \vee (q \vee r)$	Premise
2.	$\sim r$	Premise
3.	$\sim (p \vee q)$	Negated conclusion
4.	$\sim p$	3 De Morgan
5.	$\sim q$	3 De Morgan
6.	p $\quad$ $q \vee r$	1 disjunction
	$\otimes$ 4, 6	
7.	q $\quad$ r	6 disjunction
	$\otimes$ 5, 7 $\quad$ $\otimes$ 2, 7	

```
(set-logic QF_UF)   ; QF_UF supports Bool (propositional variables)
(declare-const p Bool)
```

```
(declare-const q Bool)
(declare-const r Bool)
(define-fun argument () Bool
  (=> (and (or p (or q r)) (not r))
      (or p q)))
(assert (not argument))
(check-sat) ; unsat means that there is no (p q r) making the argument invalid
(exit)
```

Example 1.6.2. Use truth tree/tableaux method and an SMT prover to show that the arguments below from Example 1.5.13 is invalid.

(a) $p \to (q \vee r), \sim q \vee \sim r \ / \ \therefore \sim p \vee \sim r$;

Solution:

1.	$p \to (q \vee r)$ ✓	Premise
2.	$\sim q \vee \sim r$ ✓	Premise
3.	$\sim (\sim p \vee \sim r)$ ✓	Negated conclusion
4.	p	3 De Morgan+Double Negative
5.	r	3 De Morgan+Double Negative
6.	$\sim q$ | $\sim r$ ⊗ 5, 6	2 disjunction
7.	$\sim p$ ⊗ 4, 7 | $q \vee r$	1 disjunction
8.	q ⊗ 6, 8 | r	6 contradiction; 7 disjunction

The leaf r is open, so the argument is invalid.

```
(declare-const p Bool)
(declare-const q Bool)
(declare-const r Bool)
(define-fun argument () Bool
  (=> (and (=> p (or q r)) (or (not q) (not r)))
      (or (not p) (not r))))
(assert (not argument))
(check-sat)
(get-model)   ; p=T, q=F, r=T only
(exit)
```

(b) $(p \wedge q) \to \sim r, p \vee \sim q, \sim q \to p \ / \ \therefore \sim r$.

Example 1.6.3 (Tutorial 1, Q17(c))**.** Write the symbolic form of the argument below and then determine its validity.

If f is integrable, then g or h is differentiable.
If g is not differentiable, then f is not integrable but it is bounded.
If f is bounded, then g or h is differentiable.
Therefore, g is differentiable.

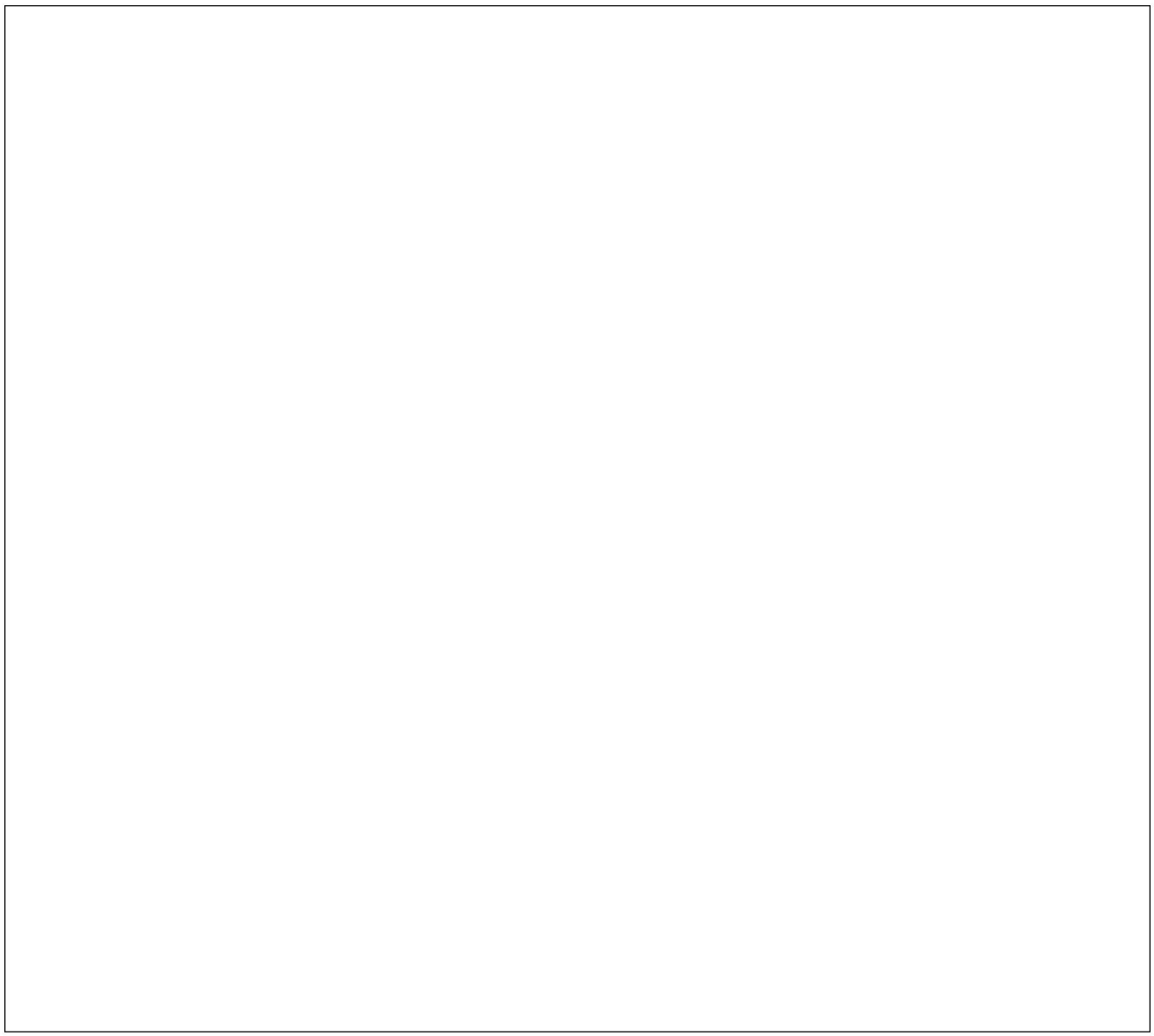
Solution: Let p: "f is integrable", q: "g is differentiable", r: "h is differentiable", s: "f is bounded". Then the argument can be written formally as

$$p \to (q \vee r),\ \sim q \to (\sim p \wedge s),\ s \to (q \vee r) / \therefore q.$$

Using Method 2, we assume conclusion $\boxed{v(q) = \mathrm{F}}$ and try to find if it is possible to make

1. $v(p \to (q \vee r)) = \mathrm{T}$
2. $v(\sim q \to (\sim p \wedge s)) = \mathrm{T}$ requires $v(\sim p \wedge s) = \mathrm{T}$ since $v(\sim q) = \mathrm{T}$.
3. $v(s \to (q \vee r)) = \mathrm{T}$

Item 2. suggests that $\boxed{v(p) = \mathrm{F}}$, $\boxed{v(s) = \mathrm{T}}$ and then Item 3. suggests $\boxed{v(q \vee r) = v(r) = \mathrm{T}}$.



§1.7 Rules of Inference for Statements

Previous laws for "logical inference" are based on **semantics of logic**. In this section, we will venture into ***syntactic implication*** or `https://en.wikipedia.org/wiki/Natural_deduction`) which is going to be fundamental tools for mathematicians and computer scientists to conduct proving in future.

There are many syntactic proving software such as Lean 4 prover, Rocq prover, Isabelle prover, etc. We will be using Lean 4 prover, developed by Microsoft, and is becoming very popular among mathematicians since 2020(?) because it supports Unicode on VS Code editor and mathematicians are trying to integrate AI into it to help mathematicians perform proving.

Most of the syntactic proving software use **typed functional programming (FP)** (e.g. Standard ML, Ocaml) [Harrison, 2009] to implement `https://en.wikipedia.org/wiki/Curry%E2%80%93Howard_correspondence`:

Logic	Proposition	Proof	Implication	Conjunction	Disjunction	T	F
FP	Type	Term	function type	Product Type	Sum Type	identity?	()?

There are many notations for natural deduction/syntactic implication:

- Gentzen (sequent calculus) syntax: Gallier [2015], Boolos et al. [2007, Chapter 14]).
- *box-and-line* syntax or *Jaśkowski nested box* syntax: Huth and Ryan [2004], Bornat [2005] (used in Jape, see `http://japeforall.org.uk`).
- `https://en.wikipedia.org/wiki/Fitch_notation`: Barwise and Etchemendy [1999], `http://forallx.openlogicproject.org/`.

We will be using the Fitch notation which is relatively easy to understand compare to Gentzen syntax used in computer science.

In natural deduction, we have such a collection of **rules of inference**. They allow us to infer a **conclusion** (a **formula**) from a set of **premises** by applying inference rules in succession (called a **proof**) [Huth and Ryan, 2004].

Let ϕ, ψ, ξ be any statements. Mathematicians have identified the following essential rules of inference for natural deduction [Bornat, 2005]:

1. $\wedge$-introduction: $\phi,\ \psi \vdash \phi \wedge \psi$
2. $\wedge$-elimination: $\phi \wedge \psi \vdash \phi$ or $\phi \wedge \psi \vdash \psi$
3. $\to$-introduction or Conditional Proof (CP in short): $\boxed{\phi,\ \cdots,\ \psi} \vdash (\phi \to \psi)$
4. $\to$-elimination: $\phi \to \psi,\ \phi \vdash \psi$
5. $\vee$-introduction: $\phi \vdash \phi \vee \psi$ or $\psi \vdash \phi \vee \psi$
6. $\vee$-elimination: $\phi \vee \psi,\ \boxed{\phi,\ \cdots,\ \xi},\ \boxed{\psi,\ \cdots,\ \xi} \vdash \xi$
7. $\neg$-introduction or $\sim$-introduction: $\boxed{\phi,\ \cdots,\ \bot} \vdash \sim \phi$
8. $\neg$-elimination or $\sim$-elimination: $\phi,\ \sim \phi \vdash \bot$
9. $\bot$-elimination: $\bot \vdash \phi$

Note that 1. and 2. are called the *rules of conjunction*, 3. and 4. are called the *rules of implication*, 5. and 6. are called the *rules of disjunction*, 7. and 8. are called the *rules of negation* [Bornat, 2005].

In the following examples, we will show how to apply the 8 rules to perform inference.

Example 1.7.1. Show that $p \to q \vdash p \to (p \wedge q)$.

Proof:

1	$p \to q$	premise
2	$\quad p$	assumption
3	$\quad q$	1,2 $\to$-elimination
4	$\quad p \wedge q$	2,3 $\wedge$-introduction
5	$p \to p \wedge q$	2,4 $\to$-introduction

Lines 2–4 serve to justify line 5, but they cannot be used in any subsequent line of the proof, they are closed off from the rest of the proof, but you are free to use line 5 as you need it.

Proof in Lean 4:

```
theorem eg1_syn01 (P Q : Prop):
  (P -> Q) -> (P -> (P /\ Q)) :=
    fun h1 : P -> Q =>
    fun h2 : P      =>
    And.intro h2 (h1 h2)
/- Print types -/
#print eg1_syn01
```

Example 1.7.2 (Modus Tollens)**.** Show that $p \to q,\ \sim q \vdash \sim p$.

Proof:

1	$p \to q$	premise
2	$\sim q$	premise
3	$\quad p$	assumption
4	$\quad q$	1, 3 $\to$-introduction
5	$\quad \bot$	2, 4 $\neg$-elimination
6	$\sim p$	$\neg$-introduction

Reference: Huth and Ryan [2004, Section 1.2.2].

Proof in Lean 4:

```
/- Not X = X -> False -/
theorem modus_tollen:
  (P -> Q) -> (Not Q) -> (Not P) :=
    fun h1 : P -> Q     =>
    fun h2 : Q -> False =>
    fun hp : P          =>
    show False from h2 (h1 hp)
```

Example 1.7.3 (Hypothetical Syllogism (Transitivity))**.** Show that $p \to q,\ q \to r \vdash p \to r$.

Hypothetical Syllogism is very easy because $\to$ corresponds to function mapping of types in type functional programming.

Proof:

1	$p \to q$	premise
2	$q \to r$	premise
3	$\quad p$	assumption
4	$\quad q$	1, 3 $\to$-elimination
5	$\quad r$	2, 4 $\to$-elimination
6	$p \to r$	3,5 $\to$-introduction

Proof in Lean 4:

```
-- Hypothetical Syllogism
theorem transitivity (P Q R : Prop):
  (P -> Q) -> (Q -> R) -> (P -> R) :=
    fun h1: P -> Q =>
    fun h2: Q -> R =>
    fun hp: P      =>
    show R from h2 (h1 hp)
```

Example 1.7.4 (Tutorial 1, Q18)**.** Show that $p \to q \vdash p \to (r \to (s \to q))$.

Example 1.7.5. Show that $p\wedge \sim s,\ q \to (r \to s) \vdash (p \to q) \to\sim r$.

Proof: Using Lean 4, we can prove using direct Curry-Howard correspondence on the left or the Lean 4 tactics on the right.

```
-- term mode
theorem syn02 (P Q R S : Prop):
  (P /\ Not S) -> (Q -> (R -> S)) ->
  (P -> Q) -> Not R :=
    fun h1 : (P /\ Not S)     =>
    fun h2 : (Q -> (R -> S)) =>
    fun h3 : (P -> Q)         =>
    fun hr : R                =>
    (And.right h1) ((h2 (h3
      (And.left h1))) hr)
#print syn02
```

```
-- tactic mode (keyword: by)
theorem syn02b (P Q R S : Prop):
  (P /\ Not S) -> (Q -> (R -> S)) ->
  (P -> Q) -> Not R := by
    intro h1
    intro h2
    intro h3
    intro hr
    exact (And.right h1)
      ((h2 (h3 (And.left h1))) hr)
#print syn02b
```

Example 1.7.6 (Disjunctive Syllogism)**.** Show that $p \vee q,\ \sim p \vdash q;\quad p \vee q,\ \sim q \vdash p$.

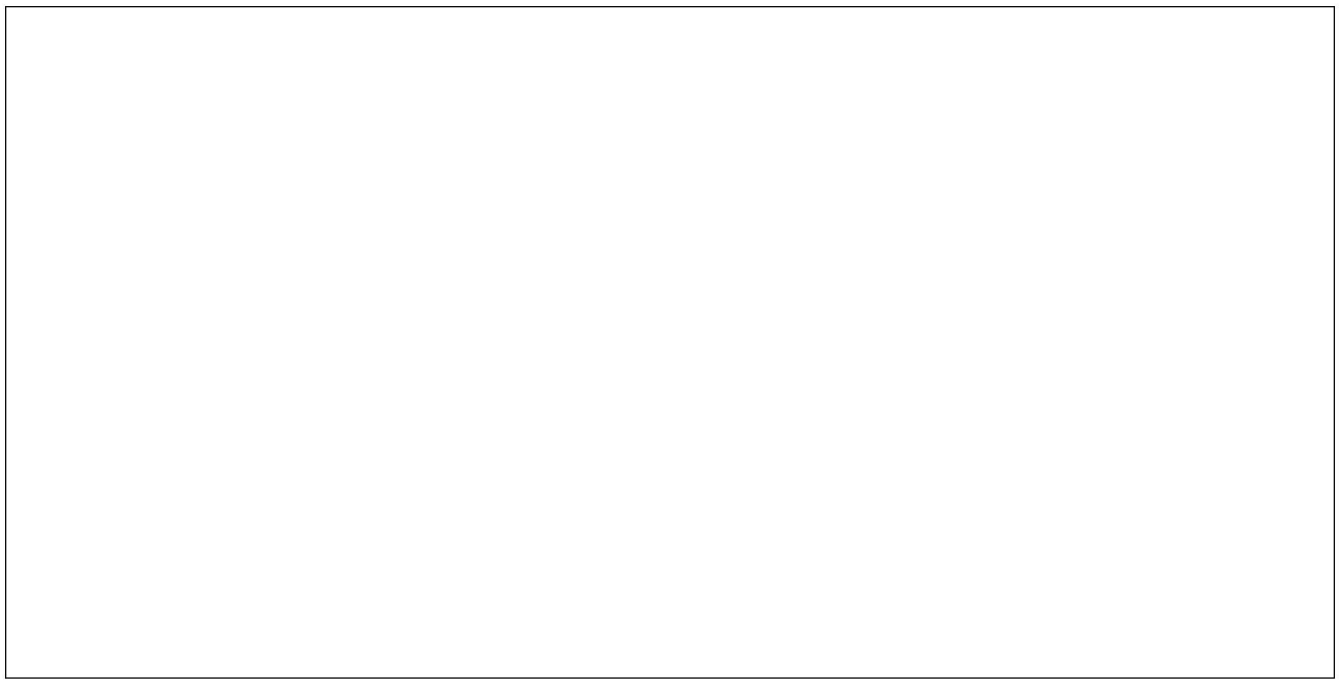
Proof:

1	$p \vee q$	premise
2	$\sim p$	premise
3	$\quad p$	assumption
4	$\quad \bot$	2, 3 $\neg$-elimination
5	$\quad q$	$\neg$-introduction
6	$\quad q$	assumption
7	$\quad q$	
8	q	1,3,5,6,7 $\vee$-elimination

Proof in Lean 4:

```
theorem disjunctive_syllogism:
  (P \/ Q) -> (Not P) -> Q :=
    fun h1: P \/ Q     =>
    fun h2: P -> False =>
    Or.elim h1
      (fun hp: P =>
        (show Q from
          False.elim (h2 hp)))
      (fun hq: Q =>
          (show Q from hq))
#print disjunctive_syllogism
```

Example 1.7.7. By using the rules of inference, show that $\sim p \vee q,\ q \to r,\ q \to s,\ p\ \vdash r \wedge s$.



Mathematically, **semantic argument** (using truth table) and **natural deduction** (using type matching) are different. The former is defined through the concept of **valuation** and the later is defined through a **proof** (solely using rules). They are linked through the following theorems [Johnstone, 1987, Barnes and Mack, 1975].

Definition 1.7.8. A logic system is said to be *sound* if $\phi_1, \cdots, \phi_n \vdash \psi$ implies $\phi_1, \cdots, \phi_n \models \psi$.

Definition 1.7.9. A logic system is said to be *adequate* if $\phi_1, \cdots, \phi_n \models \psi$ implies $\phi_1, \cdots, \phi_n \vdash \psi$.

Theorem 1.7.10 (The Soundness Theorem)**.** *If* $\phi_1, \cdots, \phi_n \vdash \psi$*, then* $\phi_1, \cdots, \phi_n \models \psi$*.*

Theorem 1.7.11 (The Completeness Theorem)**.** *If* $\phi_1, \cdots, \phi_n \models \psi$*, then* $\phi_1, \cdots, \phi_n \vdash \psi$*.*

Proof. Refer to Johnstone [1987] or Forster [2003, p. 94] Theorem 20. □

Theorem 1.7.12 (The Decidability Theorem)**.** *Given propositions/statements* ϕ_1*,* $\cdots$*,* ϕ_n*,* ψ*. There is an* ***algorithm*** *that determines whether or not* $\phi_1, \cdots, \phi_n \vdash \psi$*.*

§1.8 Syntax of Quantified Statements (Topic 1b)

Propositional logic is **too simple when we need to describe infinitely many statements!**

For example, when we want to compare two things, e.g. $x = y$. $2 - 1 = 1$, $1 = 3 - 2$, etc. It is **infeasible to list all of them**.

We need **predicates**, a function that takes in data and return true or false, as well as **quantifiers** to turn predicates into propositions (which summarised "similar" propositions into one or a few propositions).

The simplest logic system to include propositional logic and handle **predicates** and **quantified statements** is the ***first order logic***. Similar to propositional logic, there are two key parts in first order logic, i.e. syntax and semantics. The *syntax* determines which collections of symbols are legal expressions in first-order logic, while the *semantics* determine the meanings behind these expressions.

The ***(well-formed) formula*** or ***quantified statement*** for first order logic are the generalisation of Definition 1.1.2.

Definition 1.8.1. [Manin, 1977, Chapter 1] Terms and well-formed formulae are defined recursively as follows.

- ***terms***: they represent objects, i.e.
  - Variables (normally denoted $x, y, x_1, x_2, \cdots, y_1, \cdots$) and constants (normally denoted as a, b, $a_1, a_2, \cdots$) are ***atomic terms***.
  - If f is a function (which returns a value) of *degree* (or *arity*) r and $t_1, \cdots, t_r$ are terms, then $f(t_1, \cdots, t_r)$ is a ***term***.
- ***formulae***: they are relations or functions, i.e.
  - If P is a *relation* (refer to the Sets and Relations in Topic 4) of *degree (or arity)* r and $t_1, \cdots, t_r$ are terms, then $P(t_1, \cdots, t_r)$ is an ***(atomic) formula***.
  - If ϕ and ψ are formulae (abbreviated notation!), and x is a variable, then the expressions

    $$\sim(\phi),\ (\phi) \wedge (\psi),\ (\phi) \vee (\psi),\ (\phi) \to (\psi),\ (\phi) \leftrightarrow (\psi),\ \forall x(\phi),\ \exists x(\phi)$$

    are ***formulae***.

Remark 1.8.2. A predicate with no arguments is an atomic statement, as in propositional logic (Definition 1.1.2).

Representation of First-Order Logic in Standard ML

```
(* https://www.rbjones.com/rbjpub/logic/log021.htm *)
infix  1 Iff
infixr 2 Impl
infix  3 Disj
infix  4 Conj
datatype term = V of string
and      formula =
           P of string * term list
         | Neg  of formula
         | Conj of formula * formula
         | Disj of formula * formula
         | Impl of formula * formula
         | Iff  of formula * formula
         | FAll of string  * formula;
```

Similar to propositional calculus, conventions have been developed about the precedence of the logical operators, to avoid the need to write parentheses in some cases:

- $\sim$ is evaluated first
- $\wedge$ and $\vee$ are evaluated next
- Quantifiers $\forall$ and $\exists$ are evaluated next
- $\rightarrow$ and $\leftrightarrow$ are evaluated last.

Remark 1.8.3. In English, the "predicate" is the part of the sentence that tells us something about the subject. For example,

$$\underbrace{\text{James}}_{\text{subject}}\ \underbrace{\text{is a student at UTAR}}_{\text{predicate}}.$$

Example 1.8.4.

She lives in the city. ..... neither
live(x, y): x lives in y. ..... predicate (of degree / arity 2)
live(Mary, Austin): "Mary lives in Austin." ..... proposition (or predicate of degree / arity 0)

Example 1.8.5. The expression $\forall x \forall y(P(f(x)) \rightarrow \sim (P(x) \rightarrow Q(f(y), x, z)))$ is a formula where f is a unary function symbol, P a unary predicate symbol, and Q a ternary predicate symbol.

Example 1.8.6. Consider the following logic formula

$$\underbrace{(\forall x.\exists y.\underbrace{P(f(c,y))}_{\phi_2} \wedge \underbrace{Q(g(g(x)),y)}_{\psi_2})}_{\phi_1} \rightarrow \underbrace{(\exists z.\forall w. \sim \underbrace{R(z,w)}_{\phi_3})}_{\psi_1}$$

we can study the formula formally by applying the rules mentioned in Definition 1.8.1. We can also identify

- function symbols: f, c, g.
- predicate symbols: P, Q, R. ■

Example 1.8.7. The expression $\forall x\, x \rightarrow$ is *not* a formula. It is just a string of symbols!

Example 1.8.8 (Tutorial 2, Q1)**.** Let P be a proposition and $Q(x, y)$ be a predicate. Are the following strings well-formed formula?

(a) $\sim$
(b) $(\sim (P)) \wedge (P)$
(c) $\forall x((P \rightarrow Q(x, y)) \vee (P))$
(d) $\sim \forall x Q(x, x^2) \equiv \exists x \sim Q(x, x^2)$
(e) $x^2 + y^2 - 3z^2$
(f) $x^2 + y^2 - 3z^2 = 0$

In a formula, a *variable* may occur *free* or *bound*. Intuitively, a variable is free in a formula if it is not quantified: in $\forall y\, P(x, y)$, variable x is free while y is bound.

Definition 1.8.9. The free and bound variables of a formula are defined inductively as follows.

- Atomic formulas. If ϕ is an atomic formula then x is free in ϕ if and only if x occurs in ϕ. Moreover, there are no bound variables in any atomic formula.

- Negation. x is free in $\sim \phi$ if and only if x is free in ϕ. x is bound in $\sim \phi$ if and only if x is bound in ϕ.
- Binary connectives. x is free in $(\phi \to \psi)$ if and only if x is free in either ϕ or ψ. x is bound in $(\phi \to \psi)$ if and only if x is bound in either ϕ or ψ. The same rule applies to any other binary connective $\wedge$, $\vee$, $\leftrightarrow$ in place of $\to$.
- Quantifiers. x is free in $\forall y\phi$ if and only if x is free in ϕ and x is a different symbol from y. Also, x is bound in $\forall y\phi$ if and only if x is y or x is bound in ϕ. The same rule holds with $\exists$ in place of $\forall$.

Example 1.8.10. Determine if the variables x, y, z, w are free, bound or neither in the formula $\forall x \forall y(P(x) \to Q(x, f(x), z))$.

Solution:

- x and y are bound variables
- z is a free variable
- w is neither because it does not occur in the formula.

Freeness and boundness can be also specialised to specific occurrences of variables in a formula.

Example 1.8.11. For the formula $P(x) \to \forall x\, Q(x)$, the first occurrence of x is free while the second is bound, i.e. the x in $P(x)$ is free while the x in $\forall x\, Q(x)$ is bound.

Example 1.8.12 (Tutorial 2, Q2)**.** Let a be a constant, f_i be functions and P_i be predicates. Determine the bound and free variables of the following formula.

(a) $(\forall x_2 P_1(x_1, x_2)) \to P_1(x_2, a)$.

(b) $P_1(x_3) \to \sim \forall x_1 \forall x_2 P_2(x_1, x_2)$.

(c) $\forall x_2(P_1(f_1(x_1, x_2), x_1) \to \forall x_1 P_2(x_3, f_2(x_1, x_2)))$

§1.9 Formal versus Informal Language

We use informal languages such as English, Chinese, etc. in real life. Informal languages are *difficult for computer to process*. To ask computer to help us solve problems, we need to express human knowledge in a formal language. The results are represented formally by computer and they will normally be converted to informal languages for human comprehension.

First, let us define the classes of formal statements that we constantly encounter in mathematics.

Definition 1.9.1. Let $P(x)$ and $Q(x)$ be any predicates with free variable x.

1. A *universal statement* is a formula of the form
$$\forall x P(x)$$
It corresponds to these English sentences: "For all x $P(x)$", "For every x $P(x)$" and "For any x $P(x)$".

2. An *existential statement* is a formula of the form
$$\exists x Q(x).$$
It corresponds to these English sentences: "There is/exists an x such that $P(x)$", "There is at least one x such that $P(x)$", "Some x satisfies P", "$P(x)$ for some x", etc.

3. A *universal conditional statement* is a formula of the form
$$\forall x(P(x) \to Q(x)).$$
It corresponds to the English sentence "For every x, if $P(x)$ then $Q(x)$".

Formal to Informal Translation

It is generally straightforward to translate from formal language into English, since you can just turn each logical operator directly into an English word or phrase. We will be introducing some notations which are defined in later sections: $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{R}$ which are the set of natural numbers, integers and real numbers, respectively.

Example 1.9.2. Let

$A(x)$ $:= x$ is an animal.
$C(x)$ $:= x$ is a cat.
$S(x)$ $:= x$ is small.
$GP(x)$ $:= x$ is a good pet.

Translate the universal conditional statement

$$\forall x(S(x) \wedge A(x) \to GP(x)) \tag{$*$}$$

into English literally.

Solution: We could write the formal sentence $(*)$ into any of the following English sentences.

1. For every thing, if that thing is small and that thing is an animal, then that thing is a good pet.

This is graceless English, but at least it's comprehensible and correct. The style can be improved:

2. Everything which is small and which is an animal is a good pet.

Even better would be:

3. Small animals make good pets.

Such stylistic improvements in the English are optional. It is important to be sure that the effort to improve the literary style doesn't affect the meaning, but this is a question of proper usage of natural language, not of formal logic.

Example 1.9.3. Rewrite the following formal statements in a variety of equivalent but more informal ways without using the formal symbols $\forall$ and $\exists$.

1. $\forall x(x \in \mathbb{R} \to x^2 \geq 0)$.

   Solution: Any of the following English sentences are acceptable.

   - All real numbers have *nonnegative* squares.
   - Every real number has a nonnegative square.
   - Any real number has a nonnegative square.
   - x has a nonnegative square, for each real number x.
   - The square of any real number is nonnegative.

2. $\forall x(x \in \mathbb{R} \to x^2 \neq -1)$.

Solution:

- All real numbers have square not equal to -1.
- No real numbers have square equal to -1.

3. $\exists m(m \in \mathbb{Z} \wedge m^2 = m)$.

Example 1.9.4. Rewrite the following formal statement in a variety of informal ways. Do not use quantifiers or variables.

$$\forall x((x \in \mathbb{R}) \wedge (x > 2) \to (x^2 > 4)).$$

Informal to Formal Translation

It is sometimes tricky to translate from an informal sentence into a formal sentence, precisely because the informal sentence may not correspond obviously to the logical quantifiers and operators.

Example 1.9.5. Translate the sentence 'Some birds can fly' into logic. Let $B(x)$ mean 'x is a bird' and $F(x)$ mean 'x can fly'.

Solution: 'Some birds can fly' is translated as

$$\exists x(B(x) \wedge F(x)).$$

If we translate this back to 'unpolished' English, it would be 'there is something which is a bird and this thing can fly.'

Remark 1.9.6. Warning! A common pitfall is to translate 'Some birds can fly' as

$$\exists x(B(x) \to F(x)) \quad \textbf{Wrong translation!}$$

To see why this is wrong, let p be a frog that somehow got into the universe. Now $B(p)$ is false, so $B(p) \to F(p)$ is true (remember $F \to F \equiv T$). This is just saying 'If that frog were a bird then it would be able to fly', which is true; it doesn't mean the frog actually is a bird, or that it actually can fly. However, we have now found a value of x — namely the frog p — for which $B(x) \to F(x)$ is true, and that is enough to satisfy $\exists x(B(x) \to F(x))$, which is different from $\exists x(B(x) \wedge F(x))$ which requires us to find a flying bird from birds.

Example 1.9.7. Using the notations from Example 1.9.2, the English sentences in on the left side of the table below can be translated into logic formulae on the right side of the table:

Informal	Formal
Cats are animals.	$\forall x(C(x) \to A(x))$
Cats are small.	$\forall x(C(x) \to S(x))$
Cats are small animals.	$\forall x(C(x) \to S(x) \wedge A(x))$
Small animals are good pets.	$\forall x(S(x) \wedge A(x) \to GP(x))$

Example 1.9.8 (Tutorial 2, Q16)**.** Let $H(x)$ denote the predicate "x is a human" and $T(x, y)$ denote the predicate "x trusts y". Rewrite the following formula into English sentence without quantifiers and variables.

1. $\forall x \exists y(H(x) \to (H(y) \wedge T(x, y)))$
2. $\exists x \forall y(H(x) \wedge (H(y) \to\sim T(x, y)))$.
3. $\forall x \forall y(H(x) \to (H(y) \to\sim T(x, y)))$.

Example 1.9.9 (Tutorial 2, Q15)**.** Let $M(s)$ denote "s is a math major", $C(s)$ denote "s is a computer science student" and $E(s)$ denote "s is an engineering student". Rewrite the following statements by using quantifiers, variables and predicates $M(s)$, $C(s)$ and $E(s)$.

1. There is an engineering student who is a math major.
2. Every computer science student is an engineering student.
3. No computer science students are engineering students.
4. Some computer science students are engineering students and some are not.

Many mathematical quantified statements can be written in one of the form in Definition 1.9.1. However, mathematical writings normally have many implicitly quantified statements, i.e. statements that do not contain the "keywords" *all* or *every* or *any* or *each* or *exist*, which often tell us what class of quantified statements they belong. Hence, we have to look at the sentence and understand the context.

Example 1.9.10. Translate the sentence "Every integer is rational." into a quantified statement.

Example 1.9.11. Consider the statement "There is an integer that is both prime and even." Let $P(n)$ be "n is prime" and $E(n)$ be "n is even." Use the notation $P(n)$ and $E(n)$ to rewrite this statement formally.

Example 1.9.12. In calculus courses, the letter x is mostly always used to indicate a real number, hence the sentence below

"If $x > 2$ then $x^2 > 4$."

is interpreted to mean the same as the statement

"For all real numbers x, if $x > 2$ then $x^2 > 4$."

or formally $\forall x((x \in \mathbb{R} \wedge x > 2) \to (x^2 > 4))$. or
$\forall x(x \in \mathbb{R}) \to (x > 2) \to (x^2 > 4)$.

Example 1.9.13. Translate the following statement formally.

If a number is an integer, then it is a real number.

Solution: The clue that indicates its universal quantification comes from the presence of the indefinite article "a". Hence, this statement is a universal statement: $\forall x(x \in \mathbb{Z} \to x \in \mathbb{R})$.

Example 1.9.14. Let $E(x)$ be the predicate "x is even". Translate the following statement

The number 24 can be written as a sum of two even integers.

into a formal logic formula.

Example 1.9.15 (Tutorial 2, Q14)**.** Consider the predicates

$LT(x, y) : x < y$	$EQ(x, y) : x = y$	$EV(x)$: x is even
$I(x)$: x is an integer	$P(x) : x > 0$	$R(x)$: $x \in \mathbb{R}$.

1. Write the statement using ONLY these predicates and any needed quantifiers.
   (a) Every integer is even.
   (b) Some real numbers are negative integers.
   (c) If $x < y$, then x is not equal to y.
   (d) There is no largest number.
   (e) If $y > x$ and $0 > z$, then $x \cdot z > y \cdot z$.
2. Write the statement $\exists x \sim P(x)$ in English without using variables and symbols.

Example 1.9.16 (Tutorial 2, Q17)**.** Consider the following statement:

$$\exists x(x \in \mathbb{R} \wedge x^2 = 2).$$

Which of the following are equivalent ways of expressing this statement?

1. The square of each real number is 2. .....
2. Some real numbers have square 2. .....
3. If x is a real number, then $x^2 = 2$. .....

Example 1.9.17. Rewrite the following statements formally using quantifiers and variables.

1. Somebody trusts everybody.

2. Given any positive number, there is another positive number that is smaller than the given number.

3. Any even integer equal twice some other integer.

4. For every real number x and every real number y, if x is positive and y is negative, then the product of x and y is negative.

Example 1.9.18. Let S be the set of students in UTAR, M be the set of movies that have ever been released, and let $V(s, m)$ denote "student s has seen movie m". Translate each of the following statements into English.

1. $\forall s \exists m((s \in S) \to ((m \in M) \wedge V(s, m)))$.

2. $\exists s \exists t \exists m((s \in S) \wedge (t \in S) \wedge (m \in M) \wedge (s \neq t) \wedge V(s, m) \wedge V(t, m))$.

3. $\exists s \exists t((s \in S) \wedge (t \in S) \wedge (s \neq t) \wedge \forall m((m \in M) \to (V(s, m) \to V(t, m))))$.

Often the real difficulty in translating English into logic is in figuring out what the English says, or what the speaker meant to say. For example, many people make statements like 'All people are not rich'. What this statement actually says is

$$\forall x \sim R(x),$$

where the universe is the set of people and $R(x)$ means 'x is rich'. What is usually meant, however, by such a statement is

$$\sim \forall x R(x),$$

that is, it is not true that all people are rich (alternatively, not all people are rich). The intended meaning is equivalent to

$$\exists x \sim R(x).$$

Such problems of *ambiguity* or *incorrect grammar* in English cannot be solved mathematically and they will not be considered in this subject. However, they do illustrate one of the benefits of mathematics: simply translating a problem from English into formal logic may expose confusion or misunderstanding.

§1.10 Semantics of Quantified Statements

Semantics of logic is a theory to determine if a formula is true or false as an extension to propositional logic.

In the semantics of propositional logic, we assigned a truth value to each atom.

In the semantics of predicate logic, the truth values of the predicate $P(t_1, \cdots, t_n)$ depends on the terms t_i. We need a **model** to **interpret** the terms.

Definition 1.10.1. A ***model*** (or ***structure***) M consists of a nonempty set D that forms the ***domain of discourse*** (or ***universe of discourse***) and an ***interpretation*** $()^M$, which is a mapping such that

- Each function symbol f of degree n is assigned a function f^M from D^n to D. In particular, each constant symbol is assigned an individual in the domain of discourse.
- Each predicate symbol P of degree n is assigned a relation P^M over D^n, i.e. $P^M \subset D^n$.

The evaluation/interpretation of predicate logic formulas on models is formalised below.

Definition 1.10.2. Let M be a model and t be a term without variables. Then t^M, the **interpretation of t in M**, is given as follows:

- if t is a constant c, then $t^M = c^M$, $c^M \in D$;
- if t is a variable x, then $t^M = \sigma(x)$, where σ is a variable assignment;
- if t is an n-ary function $f(t_1, \cdots, t_n)$, then $t^M = f^M(t_1^M, \cdots, t_n^M)$.

For predicates and logical connectives, the interpretation goes through the process ***T-schema***:

- Atomic formulae.
  1. A formula $P(t_1, \cdots, t_n)$ is associated the value T or F depending on whether $(t_1^M, \cdots, t_n^M) \in P^M \subset D^n$.
  2. A formula $t_1 = t_2$ is assigned T if $t_1^M = t_2^M$, i.e. they evaluate to the same object of the domain of discourse.
- Logical connectives. A formula in the form $\sim \phi$, $\phi \to \psi$, etc. is evaluated according to the truth table for the connective in question, as in propositional logic.
- Existential quantifiers. A formula $\exists x \phi(x)$ is true according to M and σ if there exists an evaluation σ' of the variables that only differs from σ regarding the evaluation of x and such that ϕ is true according to the interpretation M and the variable assignment σ'. This formal definition captures the idea that $\exists x \phi(x)$ is true if and only if there is a way to choose a value for x such that $\phi(x)$ is satisfied.
- Universal quantifiers. A formula $\forall x \phi(x)$ is true according to M and σ if $\phi(x)$ is true for every pair composed by the interpretation M and some variable assignment σ' that differs from σ only on the value of x. This captures the idea that $\forall x \phi(x)$ is true if every possible choice of a value for x causes $\phi(x)$ to be true.

If a formula does not contain free variables, and so is a sentence, then the initial variable assignment does not affect its truth value. In other words, a sentence is true according to M and σ if and only if is true according to M and every other variable assignment σ'.

Example 1.10.3. [Huth and Ryan, 2004, Example 2.19] Consider the sentence "None of Alma's lovers' lovers love her." Let "Alma" can be a constant a, and the concept "x loves y" can be a binary predicate $L(x, y)$.

1. Formalise this sentence.
2. Consider the model M defined by $D = \{p, q, r\}$, $a^M = p$, $L^M = \{(p, p), (q, p), (r, p)\}$. Determine the truth value of the sentence under this model M.

Solution:

1. We can formalise the sentence as

$$\forall x \forall y (L(x, a) \wedge L(y, x) \to\sim L(y, a)).$$

   Intuitively, $L(x, a)$ says that x is Alma's lover, and $L(y, x)$ says that "y loves x", so $L(x, a) \wedge L(y, x)$ says that y is one of Alma's lovers' lovers. The formula $\sim L(y, a)$ says that y does not love Alma, and the quantifiers make sure this is true for any x, y.
2. In order to interpret the formula, we need a variable assignment σ. Consider $\sigma = \{(x, p), (y, q)\}$. Then

$$\begin{aligned} & (\forall x \forall y (L(x, a) \wedge L(y, x) \to\sim L(y, a)))^M(\sigma) \\ \equiv\ & L^M(\sigma(x), a^M) \wedge L^M(\sigma(y), \sigma(x)) \to\sim L^M(\sigma(y), a^M) \\ \equiv\ & L^M(p, p) \wedge L^M(q, p) \to\sim L^M(q, p) \equiv T \wedge T \to\sim T \equiv T \to F \equiv F. \end{aligned}$$

   The formula is false under this model.

To show that a universal formula is true, we need to make sure that it is true for all possible variable assignments.

Example 1.10.4 (Tutorial 2, Q10)**.** Suppose the model M of the predicate $P(x, y)$ has a universe of discourse $D = \{1, 2, 3\}$ and corresponding Boolean function (an equivalent way to describe a relation) P^M. Write the propositions $\exists y \sim P(1, y)$ and $\forall x P(x, 2)$ using disjunctions and conjunctions.

Remark 1.10.5. The inspirations of the above examples are

- universal quantifier is like the generalisation of conjunction: $\forall x \phi(x) \equiv \phi(x_0) \wedge \phi(x_1) \wedge \cdots$;
- existential quantifier is like the generalisation of disjunction: $\exists x \phi(x) \equiv \phi(x_0) \vee \phi(x_1) \vee \cdots$.

Example 1.10.6 (Tutorial 2, Q13)**.** Determine whether each of the following statements is true or false over the model of integer sets.

1. $\forall x \exists y \exists z (x = 7y + 5z)$
2. $\forall x \exists y \exists z (x = 31y + 41z)$
3. $\forall x \exists y \exists z (x = 4y + 6z)$

Example 1.10.7 (Tutorial 2, Q6)**.** Determine the truth value of the following statements assuming we are interpreting these formulae over the real number domain.

1. $\forall x (x > \frac{1}{x})$.
2. $\exists x (x \in \mathbb{Z} \wedge \frac{x-3}{x} \notin \mathbb{Z})$.
3. $\exists m \exists n (m \in \mathbb{Z} \wedge n \in \mathbb{Z} \wedge m > 0 \wedge n > 0 \wedge mn \geq m + n)$.
4. $\forall x \forall y (\sqrt{x + y} = \sqrt{x} + \sqrt{y})$.

Logical Equivalence

The logical equivalence in propositional logic is simple, we only need to verify it to be true for all assignments (using truth table or comparison table).

The logical equivalence in predicate logic needs to be verified to be true for all models!

The laws of logical equivalences (Theorem 1.3.11) and "substitution principle" (Theorem 1.3.15) are also valid for predicate calculus [Aho and Ullman, 1992]. However, we need to replace the term "atomic statement" with "formula" and introduce the extended notion of logical equivalence.

Definition 1.10.8. [Chiswell and Hodges, 2007] Let ϕ and ψ be two formulae with free variables $x_1, \cdots, x_n$, they are *logically equivalent*, i.e. $\phi \equiv \psi$ if $\models \forall x_1 \cdots \forall x_n(\phi \leftrightarrow \psi)$.

In addition to laws of logical equivalences from propositional logic, the following laws for quantified statements.

Theorem 1.10.9. [Chiswell and Hodges, 2007, Chapter 7] *Let ϕ and ψ be any formulas **with** free variable x (and y). Let ξ be any formula **without** free variable x. Then*

(a) $\sim \forall x\phi \equiv \exists x \sim \phi$; ..... *(Generalised de Morgan law)*

(b) $\sim \exists x\phi \equiv \forall x \sim \phi$; ..... *(Generalised de Morgan law)*

(c) $\forall x(\phi \wedge \psi) \equiv (\forall x\ \phi) \wedge (\forall x\ \psi)$; ..... *(Quantified conjunctive law)*

(d) $\exists x(\phi \vee \psi) \equiv (\exists x\ \phi) \vee (\exists x\ \psi)$; ..... *(Quantified disjunctive law)*

(e) $\forall x \forall y\phi \equiv \forall y \forall x\phi$; ..... *(Universal quantifiers swapping law)*

(f) $\exists x \exists y\phi \equiv \exists y \exists x\phi$; ..... *(Existential quantifiers swapping law)*

(g) $\xi \equiv \forall x\xi \equiv \exists x\xi$; ..... *(Independent quantifier law)*

(h) Suppose the variable x has no free occurrences in ξ and is substitutable for x in ξ. Then
$\forall x\xi \equiv \forall y\xi[y/x]$; $\exists x\xi \equiv \exists y\xi[y/x]$; ..... *(Variable renaming laws)*

(i) $\forall x(\xi \wedge \psi) \equiv \xi \wedge (\forall x\psi)$; $\exists x(\xi \wedge \psi) \equiv \xi \wedge (\exists x\psi)$; $\forall x(\xi \vee \psi) \equiv \xi \vee (\forall x\psi)$; $\exists x(\xi \vee \psi) \equiv \xi \vee (\exists x\psi)$.
..... *(Free variable laws)*

Proof. Refer to Chiswell and Hodges [2007, Chapter 7] and Gallier [2015, Chapter 5]. □

Example 1.10.10 (Tutorial 2, Q5)**.** Determine whether $\exists x(P(x) \vee Q(x))$ and $\exists xP(x) \vee \exists xQ(x)$ have the same truth value. Explain.

Corollary 1.10.11. *Let ϕ and ψ be any predicates with free variable x. Let ξ be any formula **without** free variable x. Then*

1. $\sim(\forall x(\phi \to \psi)) \equiv \exists x(\phi \wedge \sim \psi)$;
2. $\exists x(\phi \to \psi) \equiv (\forall x\ \phi) \to (\exists x \psi)$.
3. $\forall x(\xi \to \psi) \equiv \xi \to (\forall x\ \psi)$

Proof.

Example 1.10.12 (Tutorial 2, Q3)**.** Show that the statements $\exists x P(x) \wedge \exists x Q(x)$ and $\exists x(P(x) \wedge Q(x))$ are not logically equivalent in general.

Example 1.10.13 (Tutorial 2, Q4)**.** Determine whether the statements $\forall x P(x) \wedge \exists x Q(x)$ and $\forall x \exists y(P(x) \wedge Q(y))$ are logically equivalent.

A variation of Definition 1.1.8 is stated below.

Definition 1.10.14. Let ϕ and ψ be formulae with free variable x. Consider a statement of the form

$$\forall x\ (\phi \to \psi). \tag{1.3}$$

1. Its ***negation*** is $\exists x(\phi \wedge \sim \psi)$.
2. Its ***contrapositive*** form is $\forall x(\sim \psi \to \sim \phi)$.
3. Its ***converse*** form is $\forall x(\psi \to \phi)$.
4. Its ***inverse*** form is $\forall x(\sim \phi \to \sim \psi)$.

Remark 1.10.15. A universal conditional statement (1.3) is logically equivalent to its contrapositive form but not logically equivalent to its converse form and inverse form.

Example 1.10.16. Write formal negations for the statement "Every prime number is odd." using the predicate prime(n) for "n is prime" and odd(n) for "n is odd".

Solution: $\sim \forall n(\text{prime}(n) \to \text{odd}(n)) \equiv \exists n(\text{prime}(n) \wedge \sim \text{odd}(n))$

We need to exercise special care to avoid mistakes when writing negations of statements that are given informally. One way to avoid error is to rewrite the statement formally and take the negation using the formal rule as demonstrated in the example below.

Example 1.10.17. Write an informal negation for the statement

If a computer program has more than 100,000 lines, then it contains a bug.

Example 1.10.18. Write the contrapositive, converse and inverse for the following statement:

If a real number is greater than 2, then its square is greater than 4.

Remark 1.10.19. If we want to specify the domain D of x, then (1.3) will be of the form $\forall x(x \in D \to (\phi \to \psi))$ and the contrapositive, converse form and inverse form will be of the forms $\forall x(x \in D \to (\sim \psi \to \sim \phi))$, $\forall x(x \in D \to (\psi \to \phi))$ and $\forall x(x \in D \to (\sim \phi \to \sim \psi))$ respectively.

Example 1.10.20 (Tutorial 2, Q30)**.** Write a negation for the following statement:

> For all real numbers $y > 0$, there exists a real number $z > 0$ such that if $a - z < x < a + z$ then $L - y < f(x) < L + y$.

The CNF (Definition 1.4.5) plays an important role in SAT solver for propositional logic.

The predicate logic CNF for SMT solver requires one to transform a formula to prenex normal form (PNF) and then to CNF.

Definition 1.10.21. A formula of the predicate calculus is in ***prenex***[1] ***normal form*** if it is written as a string of quantifiers (referred to as the *prefix*) followed by a quantifier-free part (referred to as the *matrix*).

Theorem 1.10.22. *[Rautenberg, 2010, Theorem 4.2] Every formula in predicate logic is logically equivalent to a formula in prenex normal form.*

Example 1.10.23. Given quantifier-free formulae $\phi(y)$, $\psi(z)$, and $\rho(x)$ then

$$\forall x \exists y \forall z(\phi(y) \vee (\psi(z) \to \rho(x)))$$

is in prenex normal form with matrix $\phi(y) \vee (\psi(z) \to \rho(x))$, while

$$\forall x((\exists y \phi(y)) \vee ((\exists z \psi(z)) \to \rho(x)))$$

is logically equivalent but not in prenex normal form.

Example 1.10.24. Write logical equivalent form of $\sim \forall x \exists y \exists z P(x, y, z)$ in prenex normal form.

Solution:

$$\begin{aligned} \sim \forall x \exists y \exists z P(x, y, z) &\equiv \exists x \sim \exists y \exists z P(x, y, z) && \text{[Generalised de Morgan law (GDM)]} \\ &\equiv \exists x \forall y \sim \exists z P(x, y, z) && \text{[GDM]} \\ &\equiv \exists x \forall y \forall z \sim P(x, y, z) && \text{[GDM]} \end{aligned}$$

Example 1.10.25 (Tutorial 2, Q28)**.** Derive the following rule using laws of equivalence:

$$\sim (\forall x(x \in D \to (\forall y(y \in E \to P(x, y))))) \equiv \exists x \exists y(x \in D \wedge (y \in E \wedge \sim P(x, y))$$

Example 1.10.26 (Tutorial 2, Q29)**.** Show that $\forall x[(C(x) \wedge \exists y(T(y) \wedge L(x, y))) \to \exists y(D(y) \wedge B(x, y))] \equiv \forall x \forall y \exists z[(C(x) \wedge T(y) \wedge L(x, y)) \to (D(z) \wedge B(x, z))]$. [Barwise and Etchemendy, 1999, p324]

[1] The term 'prenex' comes from the Latin praenexus "tied or bound up in front", past participle of praenectere

The following are some semantic examples of two-quantifier statements.

Example 1.10.27. Let $P(x, y)$ denote "$x+y = 0$". What are the truth values of the quantified statements $\exists y \forall x P(x, y)$ and $\forall x \exists y P(x, y)$?

- $\exists y \forall x P(x, y)$ .....

- $\forall x \exists y P(x, y)$ .....

Example 1.10.27 demonstrates that $\exists y \forall x P(x, y) \not\equiv \forall x \exists y P(x, y)$ and this shows that the order of quantifiers is IMPORTANT!

Example 1.10.28. Let $Q(x, y)$ be the statement "$x + y = x - y$". If model M of this formula consist the universe of discourse $D = \mathbb{Z}$ and the operations are the normal operations associated with $\mathbb{Z}$. Determine the truth values of the following formula.

1. $\forall y Q(1, y)$
2. $\exists x \exists y Q(x, y)$
3. $\forall x \exists y Q(x, y)$
4. $\exists y \forall x Q(x, y)$
5. $\forall y \exists x Q(x, y)$

Example 1.10.29 (Tutorial 2, Q7)**.** Determine the truth value of each of these statements if they are modelled over the set of integers.

1. $\forall n \exists m(n^2 < m)$
2. $\exists n \forall m(nm = m)$
3. $\exists n \exists m(n^2 + m^2 = 6)$

Example 1.10.30 (Tutorial 2, Q8)**.** Let $P(x, y)$ be a predicate and the domain of discourse be a nonempty set. Given that $\forall x \exists y P(x, y)$ is true, which of the following are not necessarily true?

1. $\exists x \exists y P(x, y)$
2. $\forall x \forall y P(x, y)$
3. $\exists x \forall y P(x, y)$

Example 1.10.31 (Tutorial 2, Q9)**.** What are the truth values of $\exists y \forall x (y \geq x)$ and $\forall x \exists y (y \geq x)$ if they are interpreted over the model with the domain of nonnegative integers?

Example 1.10.32 (Tutorial 2, Q11)**.** Let odd(x) be the predicate "x is an odd positive integer." Determine the truth value of each of the following statements for the model M with domain of positive integers. If the statement is false, provide an explanation or suggest a counterexample.

1. $\forall x \forall y(\text{odd}(x+y))$.
2. $\exists x \forall y(\text{odd}(x+y))$.
3. $\forall x \exists y(\text{odd}(x+y))$.
4. $\exists x \exists y(xy+1=0)$.

§1.11 Logical Implication and Arguments for Quantified Statements (Topic 2b)

To extend the notion of argument from propositional logic to predicate logic, we need very complicated mathematics as Open Logic Project [2019] illustrates. In addition to the laws of logical equivalences (Theorem 1.3.11), "substitution principle" (Theorem 1.3.15), we have the following theorems which imply the laws of logical implications (Theorem 1.5.17).

Theorem 1.11.1. [Open Logic Project, 2019, Chapter 12] *Let* $\Gamma = \{\phi_1, \cdots, \phi_n\}$. *If* $\Gamma \subset \Gamma'$ *and* $\Gamma \models \psi$, *then* $\Gamma' \models \psi$.

It basically says that if ϕ is true for "preconditions" Γ, it is also true in a larger system Γ' containing Γ.

Theorem 1.11.2 (Semantic Deduction Theorem)**.** *Let* $\Gamma = \{\phi_1, \cdots, \phi_n\}$. $\Gamma \cup \{\phi\} \models \psi$ *iff* $\Gamma \models \phi \to \psi$.

Theorem 1.11.3. [Open Logic Project, 2019, Chapter 12] *If* $P(x)$ *is a formula with one free variable* x *and* s *and* a *are closed terms. Then*

$$P(s) \models \exists x P(x); \qquad \forall x P(x) \models P(a).$$

For statements related to quantifiers, we have the following laws.

Proposition 1.11.4. *Let* ϕ *be a formula with free variable* x *and* a *and* s *are terms free with respect to* x *in* ϕ. *Then we have the following laws.*

1. *Universal instantiation (or specialisation):* $\forall x\phi \Rightarrow \phi[a/x]$, *here* $[a/x]$ *means "a replaces x". In particular, when* ϕ *is* $P(x)$, *we have* $\forall x P(x) \Rightarrow P(a)$.
2. *Universal generalisation: If the term* a *in* $\phi[a/x]$ *is arbitrary, then* $\phi[a/x] \Rightarrow \forall x\phi$.
3. *Existential instantiation (or specialisation):* $\exists x\phi \Rightarrow \phi[s/x]$, *here* $[s/x]$ *means "s replaces x". In particular, when* ϕ *is* $P(x)$, *we have* $\exists x P(x) \Rightarrow P(s)$.
4. *Existential generalisation: For a particular term* s, $\phi[s/x] \Rightarrow \exists x\phi$.

The following theorem is a few logical implication involving quantified statements with single variable which can be shown using the laws mentioned above.

Theorem 1.11.5. *Let* p *and* q *be two formula without free variable* x, $P(x)$ *and* $Q(x)$ *be formula with free variable* x. *Then*

1. $\exists x(P(x) \wedge Q(x)) \Rightarrow (\exists x\ P(x)) \wedge (\exists x\ Q(x))$

2. $(\forall x P(x)) \vee (\forall x\ Q(x)) \Rightarrow \forall x\ (P(x) \vee Q(x))$

3. $(\exists x P(x)) \rightarrow (\forall x\ Q(x)) \Rightarrow \forall x (P(x) \rightarrow Q(x))$

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

Proof:

	$\phi_1:$	$\exists x(P(x) \wedge Q(x))$	premise
	$\psi_1:$	$P(s) \wedge Q(s)$	ϕ_1, existential specialisation
	$\psi_2:$	$P(s)$	ψ_1, specialisation
1.	$\psi_3:$	$Q(s)$	ψ_1, specialisation
	$\psi_4:$	$\exists x P(x)$	ψ_2, existential generalisation
	$\psi_5:$	$\exists x Q(x)$	ψ_3, existential generalisation
	$\therefore$	$\exists x P(x) \wedge \exists x Q(x)$	ψ_4, ψ_5, conjunction

Remark: $\Leftarrow$ is not true because we have $(\exists x(x < 0)) \wedge (\exists x(x > 0)) \equiv T \wedge T \equiv T$ but $\exists x((x < 0) \wedge (x > 0)) \equiv F$.

	$\phi_1:$	$(\forall x P(x)) \vee (\forall x\ Q(x))$	premise
	$\psi_1:$	$(\forall x P(x)) \vee (\forall y\ Q(y))$	ϕ_1, variable renaming law
	$\psi_2:$	$\forall y(\forall x P(x) \vee Q(y))$	ψ_1, free variable equivalence
2.	$\psi_3:$	$\forall x P(x) \vee Q(t)$	ψ_2, universal instantiation
	$\psi_4:$	$\forall x(P(x) \vee Q(t))$	ψ_3, free variable law
	$\psi_5:$	$P(t) \vee Q(t)$	ψ_4, universal instantiation
	$\psi_6:$	$\forall x(P(x) \vee Q(x))$	ψ_5, universal generalisation

	$\phi_1:$	$(\exists x P(x)) \rightarrow (\forall x\ Q(x))$	premise
	$\psi_1:$	$\sim \exists x P(x) \vee (\forall x\ Q(x))$	ϕ_1, implication law
	$\psi_2:$	$(\forall x \sim P(x)) \vee (\forall x\ Q(x))$	ψ_1, generalised de Morgan
	$\psi_3:$	$(\forall x \sim P(x)) \vee (\forall y\ Q(y))$	ψ_2, change of variable
3.	$\psi_4:$	$\forall y((\forall x \sim P(x)) \vee Q(y))$	ψ_3, free variable equivalence
	$\psi_5:$	$(\forall x \sim P(x)) \vee Q(t)$	ψ_4, universal instantiation
	$\psi_6:$	$\sim P(t) \vee Q(t)$	ψ_5, universal instantiation
	$\psi_7:$	$P(t) \rightarrow Q(t)$	ψ_6, implication law
	$\therefore$	$\forall x(P(x) \rightarrow Q(x))$	ψ_7, universal generalisation

4. Inspire by the syntactic proof in `https://en.wikipedia.org/wiki/Universal_generalization`, the deduction should be something like:

$\phi_1:$	$\forall x(P(x) \rightarrow Q(x))$	premise
$\phi_2:$	$\forall x P(x)$	premise
$\psi_1:$	$P(t) \rightarrow Q(t)$	ϕ_1 UI
$\psi_2:$	$P(t)$	ϕ_2 UI
$\psi_3:$	$Q(t)$	ψ_1, ψ_2 MP
$\psi_4:$	$\forall x Q(x)$	ψ_3 UG

$\downarrow$

$\phi_1:$	$\forall x(P(x) \rightarrow Q(x))$	premise
$\psi_5:$	$\forall x P(x) \rightarrow \forall x Q(x)$	ϕ_2, ψ_4 Semantic Deduction Theorem

Example 1.11.6 (Tutorial 2, Q22)**.** Show that $\sim(\forall x(P(x) \to Q(x))) \Rightarrow \exists x(P(x) \wedge \sim Q(x))$.

Example 1.11.7 (Tutorial 2, Q26)**.** Show that the following argument is valid.

$$\begin{array}{ll} & \exists x(F(x) \wedge S(x)) \to (\forall y(M(y) \to W(y))) \\ & \exists y(M(y) \wedge \sim W(y)) \\ \hline \therefore & \forall x(F(x) \to \sim S(x)) \end{array}$$

Example 1.11.8 (Tutorial 2, Q20)**.** For the following arguments, state which are valid and which are invalid. Justify your answers.

1. All healthy people eat an apple a day. John is not a healthy person. Therefore John does not eat an apple a day.

2. Every student who studies discrete mathematics is good at logic. John studies discrete mathematics. Therefore John is good at logic.

3. No heavy object is cheap. XYZ is not a heavy object. Therefore XYZ is cheap.

Example 1.11.9. Rewrite the following argument using quantifiers, variables, and predicate symbols:

If a number is even, then its square is even.
k is a particular number that is even.
$\therefore$ k^2 is even.

Is the argument valid? Why?

§1.12 Satisfiability Modulo Theories (SMT)

Satisfiability (Definition 1.4.3), a problem of determining whether a formula expressing a constraint has a solution, is a fundamental problem in theoretical computer science [Bjørner and de Moura, 2009].

Many-sorted first-order logic is a commonly used formalism and framework for formulating SMT problems. A ***(many-sorted) signature*** is composed of a set of *sorts* (data types), a set of *function symbols*, and a set of *predicate symbols*. Each function symbol f has associated with it an arity of the form $\sigma_1 \times \cdots \times \sigma_n \to \sigma$, where $\sigma_1, \cdots, \sigma_n, \sigma$ are sorts. If $n = 0$, we say f is a *constant symbol*. Similarly, each predicate symbol p has associated with it an arity of the form $\sigma_1 \times \cdots \times \sigma_n$. If $n = 0$, we say p is a propositional symbol. [**?**]

We assume a set of *variables* X, where each variable is associated with a sort. A *term* t with sort σ has the form x or $f(t_1, ..., t_n)$, where x is a variable with sort σ, and f is a function symbol with arity $\sigma_1 \times \cdots \times \sigma_n \to \sigma$, where for each $i \in \{1, ..., n\}$, t_i has sort σ_i. An *atom* is of the form $p(t_1, ..., t_n)$ where p is a predicate symbol with arity $\sigma_1, \cdots, \sigma_n$, and for each $i \in \{1, ..., n\}$, t_i is a term with sort σ_i. A *formula*

$$\phi = \text{atom} | \sim \phi_0 | \phi_0 \wedge \phi_1 | \phi_0 \vee \phi_1 | \phi_0 \to \phi_1 | \phi_0 \leftrightarrow \phi_1 | (\forall x : \sigma . \phi_0) | (\exists x : \sigma . \phi_0)$$

where ϕ_0, ϕ_1 are smaller formulas. Note that it is similar to Definition 1.8.1 where the variable x has no type while here the variable x has a type σ. A Σ-*formula* ϕ is a formula where each symbol in ϕ is in Σ. We say a variable x is *free* in formula ϕ if it is not bound by any quantifier $\exists$, $\forall$. A ***sentence*** is a formula **without free variables**. We use $vars(\phi)$ to denote the set of free variables in ϕ. A *quantifier-free formula* is a formula not containing $\exists$ or $\forall$. [**?**]

A ***structure*** M for a signature Σ and variables X consists of non-empty domains $|M|_\sigma$ for each sort in Σ, for each $x \in X$ with sort $\sigma, M(x) \in |M|_\sigma$, for each function symbol f with arity $\sigma_1 \times \cdots \times \sigma_n \to \sigma$, $M(f)$ is a total map from $|M|_{\sigma_1} \times \cdots \times |M|_{\sigma_n} \to |M|_\sigma$, and for each predicate symbol p with arity $\sigma_1 \times \cdots \times \sigma_n$, $M(p)$ is a subset of $|M|_{\sigma_1} \times \cdots \times |M|_{\sigma_n}$. The ***interpretation*** of a term t is given by $M[[x]] = M(x)$ and $M[[f(t1, ..., tn)]] = M(f)(M[[t1]], ..., M[[tn]])$. We assume that, for each sort σ, the equality $=_\sigma$ is a builtin predicate symbol with arity $\sigma \times \sigma$ that does not occur in any signature and for every structure M, $M(=_\sigma)$ is the identity relation over $|M|_\sigma \times |M|_\sigma$. As a notational convention, we will always omit the subscript. We use $M\{x \mapsto \nu\}$ to denote a structure where the variable symbol x with sort σ is interpreted as $\nu \in |M|_\sigma$, and all other variables, function and predicate symbols have the same interpretation as in M. Given a formula ϕ and a structure M, satisfaction $M \vdash \phi$ is defined as:

- $M \models p(t_1, ..., t_n)$ iff $\langle M[[t_1]], ..., M[[t_n]] \in M(p)$
- $M \models \sim \phi$ iff $M \not\models \phi$
- $M \models \phi_0 \vee \phi_1$ iff $M \models \phi_0$ or $M \models \phi_1$
- $M \models \phi_0 \wedge \phi_1$ iff $M \models \phi_0$ and $M \models \phi_1$
- $M \models (\exists x : \sigma . \phi)$ iff $M\{x \mapsto \nu\} \models \phi$ for some $\nu \in |M|_\sigma$
- $M \models (\forall x : \sigma . \phi)$ iff $M\{x \mapsto \nu\} \models \phi$ for all $\nu \in |M|_sigma$

A formula ϕ is **satisfiable** if there is a structure M s.t. $M \models \phi$, and is **valid** if for all structures M, $M \models \phi$. A structure M satisfies a *set of formulas* S $(M| = S)$ if $M \models \phi$ for every $\phi \in S$. [**?**]

Related: `https://ocw.mit.edu/courses/6-001-structure-and-interpretation-of-computer-programs-spring-2005/pages/readings/`

A theory is a set of sentences. More formally, a ***Σ-theory*** is a collection of sentences over a signature Σ. Given a theory T, we say ϕ is satisfiable modulo T if $T \cup \{\phi\}$ is satisfiable. We use $M \models_T \phi$ to denote $M \models \{\phi\} \cup T$.

Example 1.12.1. Let Σ be the signature containing the symbols 0, 1, +, − and <, and $\mathbb{Z}$ be the structure that interprets these symbols in the usual way over the integers, then the theory of linear arithmetic is the set of first-order sentences that are true in $\mathbb{Z}$.

Let Ω be a class of structures over a signature Σ, then we use $Th(\Omega)$ to denote the set of all sentences ϕ over Σ such that $M \models \phi$ for every M in Ω. We say the satisfiability problem for theory T is ***decidable*** if there is a procedure $\mathfrak{S}$ that checks whether any quantifier-free formula is satisfiable or not. In this case, we say $\mathfrak{S}$ is a decision procedure for T. [**?**]

Theorem 1.12.2. *The satisfiability problem for predicate logic is undecidable.*

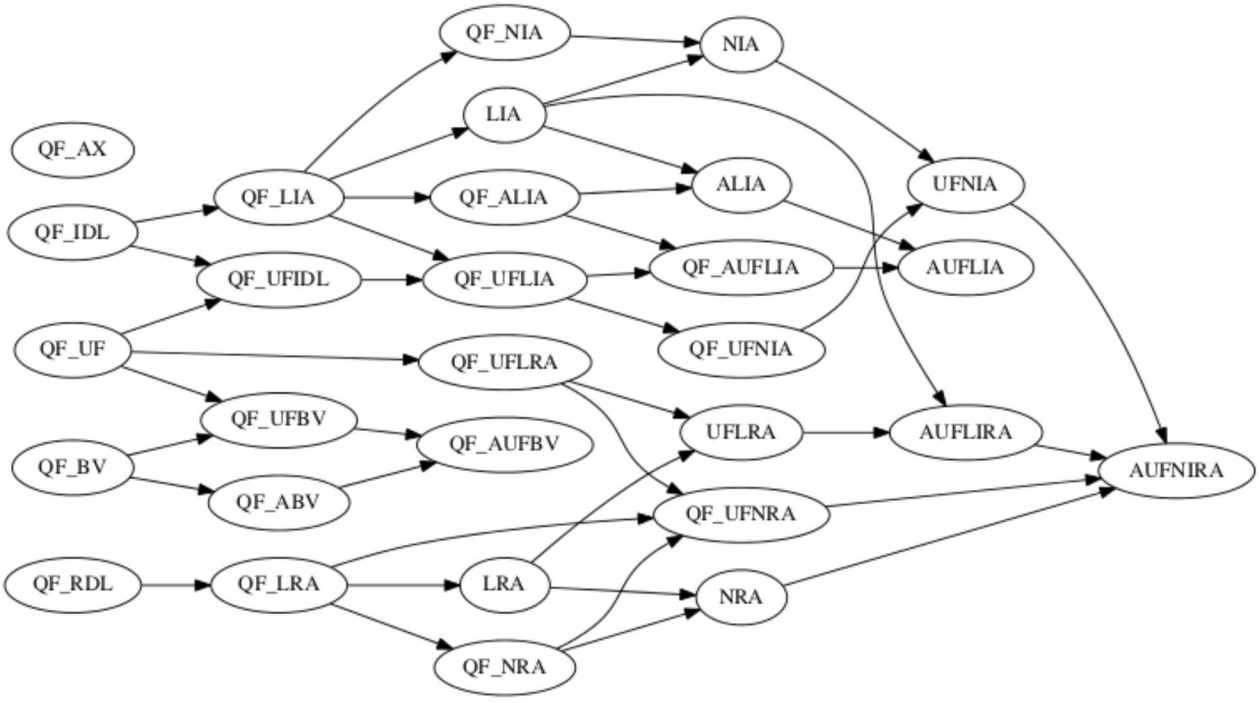
Theorem 1.12.3. *For any predicate logic formula ϕ, $\vdash \phi$ iff $\models \phi$.*

Satisfiability Modulo Theories (SMT) is an area of automated deduction that studies methods for checking the satisfiability of first-order formulas with respect to some logical theory T of interest.

SMT-LIB (`https://smt-lib.org/index.shtml`) is an international initiative aimed at facilitating research and development in SMT by promoting common specification for SMT solvers such as Z3 (Section 1.4) as well as benchmarks (`https://zenodo.org/communities/smt-lib`). It specifies for the following logical theories:

- Core: Core theory, defining the basic Boolean operators (which we used in Section 1.4)
- Ints: Integer numbers (to use in next topic?)
- Reals: Real numbers
- FloatingPoint: Floating point numbers
- FixedSizeBitVectors: Bit vectors with arbitrary size
- ArraysEx: Functional arrays with extensionality
- Strings: Unicode character strings and regular expressions

SMT-LIB supports the following sub-logics based on the stated theories.



Overview of the sub-logics, ordered by inclusion ($L_1 \to L_2$ means every formula of L_1 is also a formula of L_2
Source: `https://smt-lib.org/logics.shtml`

The conventions of the abbreviations are shown below:

- QF for the restriction to quantifier free formulas
- A or AX for the theory ArraysEx
- BV for the theory FixedSizeBitVectors
- FP (forthcoming) for the theory FloatingPoint

- IA for the theory Ints (Integer Arithmetic)
- RA for the theory Reals (Real Arithmetic)
- IRA for the theory Reals_Ints (mixed Integer Real Arithmetic)
- IDL for Integer Difference Logic
- RDL for Rational Difference Logic
- L before IA, RA, or IRA for the linear fragment of those arithmetics
- N before IA, RA, or IRA for the non-linear fragment of those arithmetics
- UF for the extension allowing free sort and function symbols

SMT solvers are used in many formal method based applications such as the mCRL2 is a formal specification language (`https://mcrl2.org/web/index.html`).

Example 1.12.4. The SMT-LIB script to check the formulas in Example 1.10.28 in Ints theory is shown below.

```
(set-logic LIA)
;;; Q(x,y) : x + y = x - y
; 1. forall y Q(1, y)
(push)
(assert (forall ((y Int)) (= (+ 1 y) (- 1 y))))
(check-sat)
(pop)
; 2. E x E y Q(x, y)
(push)
(assert (exists ((x Int) (y Int)) (= (+ x y) (- x y))))
(check-sat)    ; Not returning any model
(pop)
(push)
(declare-fun x () Int)
(declare-fun y () Int)
(assert (= (+ x y) (- x y)))
(check-sat)
(get-model)
(pop)
; 3. A x E y Q(x, y)
(push)
(assert (forall ((x Int)) (exists ((y Int)) (= (+ x y) (- x y)))))
(check-sat)
(pop)
; 4. E y A x Q(x, y)
(push)
(assert (exists ((y Int)) (forall ((x Int)) (= (+ x y) (- x y)))))
(check-sat)
(pop)
; 5. A y E x Q(x, y)
(push)
(assert (forall ((y Int)) (exists ((x Int)) (= (+ x y) (- x y)))))
(check-sat)
(pop)
```

§1.13 Rules of Inference for Quantified Statements

The "syntactic implication" or "natural deduction" of quantified statements and the Curry-Howard correspondence is the foundation to modern formal proving software such as Lean 4 prover, Rocq prover, etc.

The Curry-Howard correspondence for quantifiers is

Logic	**Proposition**	**Conjunction**	**Disjunction**	**Universal Quantification**	**Existential Quantification**
FP	**Type**	**Product Type**	**Sum Type**	**Dependent Product Type**	**Dependent Sum Type**
Logic	**Hilbert-style deduction system**	**natural deduction**	**hypotheses**	**implication elimination (modus ponens)**	**implication introduction**
FP	**type system for combinatory logic**	**type system for lambda calculus**	**free variables**	**application**	**abstraction**

In predicate logic, we have the following rules of inference related to quantifiers and equality in addition to the rules of inference from Section 1.7:

10. $\forall$-introduction: $^t\phi(t) \vdash \forall x \phi(x)$
11. $\forall$-elimination: $\forall x \phi(x) \vdash \phi(t)$
12. $\exists$-introduction: $\phi(s) \vdash \exists x \phi(x)$
13. $\exists$-elimination: $\exists x \phi(x), \boxed{\phi(s) \cdots \xi} \vdash \xi$
14. $=$-introduction: $\vdash t = t$
15. $=$-elimination: $t_1 = t_2,\ \phi(t_1) \vdash \phi(t_2)$

Here,

- s is a ***fresh variable*** which does not occur in the premise $\exists x \phi(x)$, conclusion ξ or any assumptions in the syntactic derivation;
- t, t_1, t_2 are arbitrary terms.

Note: $\vdash$ indicates logical entailment based on the rules of inference instead of the laws of logical equivalences and implications based on the semantic theory.

Example 1.13.1. Consider the formula " $\forall n(\text{prime}(n) \wedge (n > 2) \to \text{odd}(n))$. " By applying universal instantiation with n "instantiate" to the term "$m + 4$" we obtain

$$\text{prime}(m + 4) \wedge (m + 4 > 2) \to \text{odd}(m + 4).$$

Here $m + 4$ is a **term** and m is an **arbitrary constant**.

Example 1.13.2 (`https://github.com/OpenLogicProject/fitch`)**.** Show that $\forall x \sim P(x) \vdash\sim \exists x P(x)$.

Proof:

	1	$\forall x \sim P(x)$	premise
	2	$\quad \exists x P(x)$	assumption
	3	$\quad \overset{s}{\vert} \; P(s)$	fresh variable
	4	$\quad \vert \quad \sim P(s)$	1, $\forall$I
	5	$\quad \vert \quad \bot$	3,4, $\sim$E
	6	$\quad \bot$	2,3–5, $\exists$E
	7	$\sim \exists x P(x)$	2–6, $\sim$I

Example 1.13.3. Show $\exists x(P(x) \wedge Q(x)) \vdash (\exists x\ P(x)) \wedge (\exists x\ Q(x))$ (Theorem 1.11.5(a)).

Proof:

	1	$\exists x(P(x) \wedge Q(x))$	premise
	2	$\overset{s}{\vert} \; P(s) \wedge Q(s)$	1, $\exists$E
	3	$\quad \vert \quad P(s)$	2, $\wedge E_1$
	4	$\quad \vert \quad Q(s)$	2, $\wedge E_2$
	5	$\exists x P(x)$	1,2–3, $\exists$I
	6	$\exists x Q(x)$	1,2–4, $\exists$I
	7	$\exists x P(x) \wedge \exists x Q(x)$	5,6, $\wedge$I

Formal proving using Lean 4 Prover

```
theorem eg_1_13_3 (a : Type) (p q : a -> Prop) :
  (exists x : a, p x /\ q x) -> (exists x : a, p x) /\ (exists x : a, q x) :=
    fun premise : (exists x : a, p x /\ q x) =>
      Exists.elim premise
       (fun s =>
        fun hs : p s /\ q s =>
        And.intro ⟨s, And.left hs⟩ ⟨s, And.right hs⟩)
```

Note: the ⟨ and ⟩ are in Unicode and needs to be typeset using Unicode editor. E.g. in Emacs, `C-x 8 RET 27e8 C-x 8 RET 27e9`.

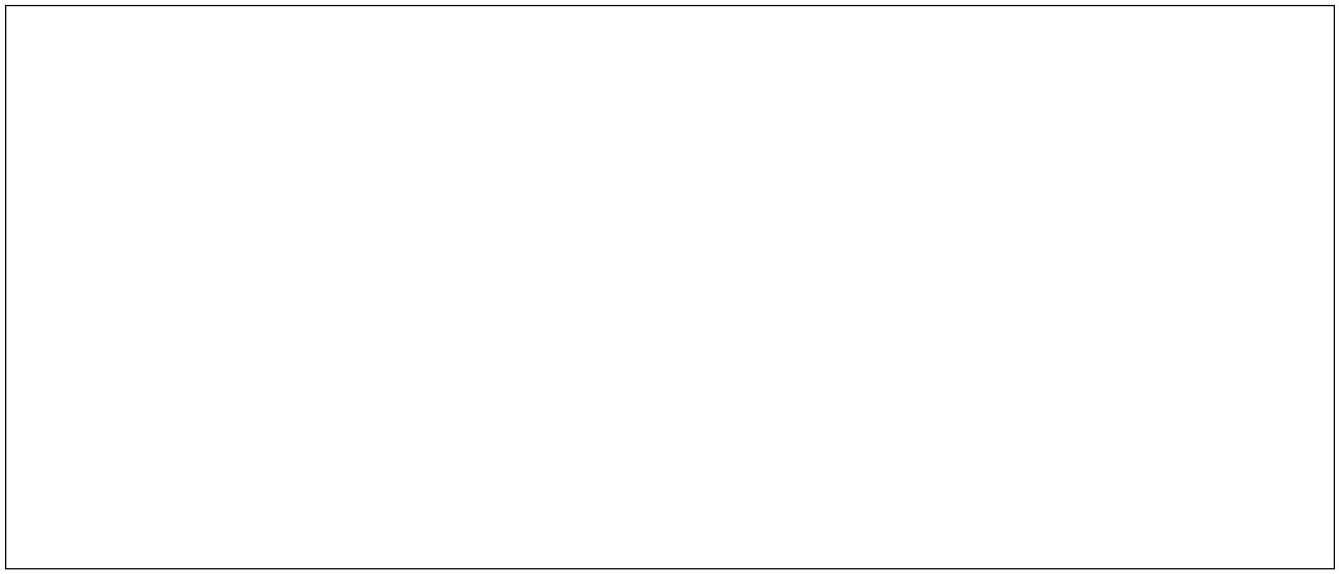
Example 1.13.4. Show that $\sim R(c),\ \forall t(P(t) \to Q(t)),\ \forall t(Q(t) \to R(t)) \vdash \sim P(c)$.

Proof:

	1	$\sim R(c)$	premise
	2	$\forall t(P(t) \to Q(t))$	premise
	3	$\forall t(Q(t) \to R(t))$	premise
	4	$P(c) \to Q(c)$	2, $\forall$E
	5	$Q(c) \to R(c)$	3, $\forall$E
	6	$\quad \vert \quad P(c)$	assumption
	7	$\quad \vert \quad Q(c)$	6,4 $\to$E
	8	$\quad \vert \quad R(c)$	7,5 $\to$E
	9	$\quad \vert \quad \bot$	1,8 $\sim$E
	10	$\sim P(c)$	6,9 $\sim$I

Example 1.13.5 (Tutorial 2, Q21)**.** Use ONLY the rules of inference to show that

$$\forall x(P(x) \to (Q(x) \wedge R(x))),\ \forall x(P(x) \wedge S(x)) \ \vdash\ \exists x(R(x) \wedge S(x))$$



Example 1.13.6 (Tutorial 2, Q23)**.** Use rules of inference to show that

$$\exists x P(x) \to \forall x(P(x) \vee Q(x) \to R(x)),\ \exists x(P(x) \wedge Q(x)) \vdash \exists y R(y)$$

Example 1.13.7 (Tutorial 2, Q27)**.** What is wrong with the following proof?

1	$\forall x \exists y(x > y)$	premise
2	$\exists y(c > y)$	$\forall$-elimination, c arbitrary
3	$(c > s)$	$\exists$-elimination, s specific
4	$\forall x(x > s)$	$\forall$-introduction
5	$\exists y \forall x(x > y)$	$\exists$-introduction

Example 1.13.8 (Tutorial 2, Q24)**.** Show that $\forall x(Q(x) \to R(x)) \wedge (\exists x(Q(x) \wedge I(x)) \vdash \exists x(R(x) \wedge I(x))$.

Now we look at examples of arguments written in English sentences and validate the argument using the laws and rules mentioned above.

Example 1.13.9. Show that the following arguments is valid with a formal proof using the laws of inference.

No junior or senior is enrolled in calculus class.

Ali is enrolled in calculus class. Therefore Ali is not a senior.

Let $J(x)$: "x is a junior"; $S(x)$: "x is a senior"; and $C(x)$: "x is enrolled in calculus class."

Proof:

1	$\forall x((J(x) \vee S(x)) \to\sim C(x))$	premise
2	$C(\text{Ali})$	premise
3	$(J(\text{Ali}) \vee S(\text{Ali})) \to\sim C(\text{Ali})$	1, $\forall$-eliminination
4	$S(\text{Ali})$	assumption
5	$J(\text{Ali}) \vee S(\text{Ali})$	$\vee$-intro2
6	$\sim C(\text{Ali})$	3,4 $\to$-elim
7	$\perp$	2,5 $\neg$-elim
8	$\sim S(\text{Ali})$	6, $\neg$-intro

Example 1.13.10. Show that the premises "A student in this class has not read the book," and "Everyone in this class passed the first exam" imply the conclusion "someone who passed the first exam has not read the book."

Example 1.13.11 (Tutorial 2, Q25)**.** Prove that the following argument is valid: "Every undergraduate is either an arts student or a science student. Some undergraduates are top students. James is not a science student, but he is a top student. Therefore if James is an undergraduate, he is an arts student."

Applications of Predicate Logic

The formal logic system can be applied in

- logic programming
- *hardware and software specification*: Bjørner [2006], Lamport [2003]
- *hardware verification*: Kropf [1999]
- compiler verification: Appel et al. [2014]

In this section, we will only explore logic programming, the relatively less complex subject.

According to `https://en.wikipedia.org/wiki/Logic_programming`, ***logic programming*** is a type of programming paradigm which is largely based on predicate logic. Any program written in a logic programming language is a set of sentences, expressing ***facts*** and ***rules*** about some problem domain.

Major logic programming language families include *Prolog* (which stands for Programming in Logic), ECLiPSe, *Datalog* (a subset of Prolog for database and serves as an alternative query system to SQL, refer to Abiteboul et al. [1995]) and *answer set programming* (ASP).

Prolog [Sterling and Shapiro, 1999], which dates back to 1972, was once believed to be the artificial intelligence (AI) language. However, the limitations of the first order logic and the difficulty to get experts to encode domain knowledge have prevented Prolog from getting popular. Despite these, Prolog is still useful in knowledge-based (or ontology) systems [Merritt, 2000].

Definition 1.13.12. [van Le, 1993] In Prolog, a ***term*** is either a ***constant***, a ***variable*** (starts with a capital letter or underscore) or a ***compound term*** of the form $f(t_1, \cdots, t_n)$ where f is a function symbol of arity n and t_i ($i = 1, \cdots, n$ and $n > 0$) are terms.

An ***atomic formula*** or ***actomic term*** is an expression of the form $p(t_1, \cdots, t_n)$ where p is a predicate symbol and t_i are terms. If $n = 0$, there is no argument and the parentheses are omitted.

A ***constant*** can either be a number or an ***atomic term*** (start from small letter, a string in single quotes or symbols such as $+$, $=/=$, etc.).

The rules in Prolog are written in the form of ***Horn clauses***, which can be regarded as "arguments" in Definition 1.5.7.

$$\underbrace{\underbrace{H}_{\text{head}} :- \underbrace{B_1, \cdots, B_n}_{\text{body}}.}_{\text{rule}} \tag{1.4}$$

where is the same as $\wedge$, $:-$ is the same as $\rightarrow$, H and B_i are usually atomic formulas. So we read (1.4) as a logical implication "H if B_1 and $\cdots$ and B_n." ***Facts*** are rules that have **no body**, and are written in the simplified form (which is comparable to a "tautology"):

$$H. \tag{1.5}$$

Remarks on Prolog syntax:

- A string starts and ends with double quote `"`
- Don't use single quotes, they are for '**atom**' (or **symbol**)
- Anything that starts with **small letter** is used as "predicate" or atom.
- Anything that starts with **capital letter** is used as "variables".

Running commands sequentially in Prolog:

```
% swipl -q hellolines.pl
:- format("This is first line\n").
:- format("This is second line: ~d ~d ~d\n", [1, 2, 3]).
:- halt.
```

Note that rules without head are called **directives** and are immediately executed by Prolog.

Prolog goes through all predicates and we sometimes need to tell Prolog to stop trying out all possibilities by **cutting** using the "exclaimation mark":

```
thegrade(Marks, Grade) :- Marks < 0,   Grade="????", !.
thegrade(Marks, Grade) :- Marks > 100, Grade="????", !.
thegrade(Marks, Grade) :- Marks >= 90, Grade="A+", !.
thegrade(Marks, Grade) :- Marks >= 80, Grade="A", !.
thegrade(Marks, Grade) :- Marks >= 75, Grade="A-", !.
thegrade(Marks, Grade) :- Marks >= 70, Grade="B+", !.
thegrade(Marks, Grade) :- Marks >= 65, Grade="B", !.
thegrade(Marks, Grade) :- Marks >= 60, Grade="B-", !.
thegrade(Marks, Grade) :- Marks >= 55, Grade="C+", !.
thegrade(Marks, Grade) :- Marks >= 50, Grade="C", !.
thegrade(Marks, Grade) :- Grade="F".
```

Prolog uses recursion on predicates to perform looping. Suppose we want to find the sum of the squares of a list numbers of any size:

$$x_1^2 + x_2^2 + ... + x_n^2$$

this is how it is done in Standard ML:

```
fun sum_sq [] = 0
  | sum_sq (x::xs) = x*x + sum_sq xs;

print (Int.toString (sum_sq [1,3,5,7]) ^ "\n")
```

The implementation in Prolog is similar:

```
% gprolog --consult-file sum_sq.pl
sum_sq([], 0).
% Be careful with X and Xs must be capitalised.
% Otherwise, they are treated as atoms.
sum_sq([X|Xs], Sum) :- sum_sq(Xs, Sum_rest), Sum is X*X + Sum_rest.

%sum_sq([1,3,5,7],X).
```

Expression (1.4) is not sufficient for efficient computing. Prolog allows the body of a clause to be negations of atomic formulas to allow non-monotonic logic. In Datalog and ASP, logic programs have only a declarative reading, and their execution is performed by means of a proof procedure or model generator whose behaviour is not meant to be under the control of the programmer.

Prolog could be used to solve some of the logic problems discussed in propositional logic which is illustrated below.

Example 1.13.13. Use Prolog to determine all truth values assignments in Example 1.2.7.

Solution:

```
% Tutorial 1, Q3 (Example 1.2.7)
proposition(P,Q,R,S,Val) :- not(P,NotP), or(NotP,R,Expr1),
       not(S,NotS), and(Expr1,NotS,Expr2), imply(Q,Expr2,Expr3),
       not(R,NotR), and(NotR,Q,Expr4), imply(NotS,Expr4,Expr5),
       and(Expr3,Expr5,Val).
```

```
%write_list([]).
%write_list([H|T]) :- write(H), write(', '), write_list(T).

solv:- format("Determine all truth values assignment [P,R,S]:~n",[]),
       findall([P,R,S], proposition(P,true,R,S,true), Results),
       write(Results), nl.  % Raw output: write_canonical
```

Remark: More examples are given in Clocksin [1997, Chapter 7].

To illustrate the use of Prolog in real-world problem, I will use illustrate it with the following "to be developed education query system":

utar_dmas.pl

```
course('UECM1024', 'Calculus I', core, 4, []).
course('UECM1034', 'Calculus II', core, 4, ['UECM1024']).
course('UECM1204', 'Probability and Statistics I',core,4,[]).
course('UECM1224', 'Probability and Statistics II',core,4,['UECM1204']).
course('UECM1314', 'Fundamentals of Linear Algebra',core,4,[]).
course('UECM1304', 'Discrete Mathematics with Applications',core,4,[]).
course('UECM1703', 'Introduction to Scientific Computing',core,3,[]).
course('UECM2353', 'Linear Algebra',core,3,['UECM1314']).
course('UECS1004', 'Programming and Problem Solving',core,4,[]).
course('UECS1044', 'Object-Oriented Application Development',core,4,['UECS1004']).
course('UECS2083', 'Problem Solving with Data Structures and Algorithms',core,3,['UECS104
course('UECS2094', 'Web Application Development',core,4,['UECS1044']).
course('UECM2023', 'Ordinary Differential Equations',core,3,['UECM1034']).
course('UECM3034', 'Numerical Methods',core,4,[]).

courses_without_prerequisites(X,Y) :- course(X,Y,_,_,[]).
courses_with_prerequisites(X,Y) :- course(X,Y,_,_,Z), length(Z, N), N =\= 0.
```

1. To **load** the Prolog database, one can use either the commands below:

   ```
   ?- [utar_dmas].
   ?- consult('utar_dmas.pl').
   ```

2. To **query** the courses with only the code without prerequisite, we can type the following query. Note that underscore is used to indicate a value we are not interested in.

   ```
   ?- courses_without_prerequisites(X,_).
   ```

3. To **query** the courses (showing both code and name) with prerequisite, we can type the following query.

   ```
   ?- courses_with_prerequisites(X,Y).
   ```

4. To **exit** Prolog, type `halt.`

One interesting use of Prolog is probably in puzzle solving.

Real-world business software consist a lot of SQL statements. A dictionary of SQL relational algebra and Datalog is given below [Ullman and Widom, 2008, Chapter 5].

SQL Concept	Relational Algebra	Datalog
Intersection	$R(x,y) \cap T(x,y)$	$I(x,y)$:-$R(x,y), T(x,y)$.
Union	$R(x,y) \cup T(x,y)$	$U(x,y)$:-$R(x,y)$.
		$U(x,y)$:-$T(x,y)$.
Difference	$R(x,y) \setminus T(x,y)$	$D(x,y)$:-$R(x,y)$, not $T(x,y)$.
Projection	$\pi_x(R)$	$P(x)$:-$R(x,y)$
Selection	$\sigma_{x>10}(R)$	$S(x,y)$:-$R(x,y), x > 10$
Product	$R \times T$	$P(x,y,z,w)$:-$R(x,y), T(z,w)$
Natural Join	$R \bowtie T$	$J(x,y,z)$:-$R(x,y), T(y,z)$
Theta Join	$R \bowtie_{R.x>T.z} T$	$\theta(x,y,z,w)$:-$R(x,y), T(z,w), x > z$

The website `http://perugini.cps.udayton.edu/teaching/courses/Spring2017/cps499/Languages/notes/PROLOG.html` did not provide any solution for this, but did provide an example to detect loops in the path as follows (using "not" operator \+ as breaking point to infinite loop?):

```
cycle(Start, Visited) :- cycle(Start, Start, [Start], Visited).

cycle(Orig, Start, Path, Visited) :-
   edge(Start,Orig),
   reverse([Orig|Path], Visited).

cycle(Orig, Start, Path, Visited) :-
   edge(Start, Next),
   \+ member(Next, Path),
   cycle(Orig, Next, [Next|Path], Visited).
```

Exercise with Past Year Questions

Only 2019 Q1-Q3 and 2021 questions are set by me. The rest are by other lecturers.

Example 1.13.14 (Final May 2019 Sem, Q1)**.** (a) Let p, q, r be atomic statements. **State** the truth table for the following compound statement

$$\sim (p \to ((p \vee q) \wedge r)).$$

Use the truth table to **recognise** whether the compound statement is a tautology, contingency or contradiction. (10 marks)

Solution: The truth table is stated below. ..... [8 marks]

p	q	r	$(p \vee q) \wedge r$	$p \to ((p \vee q) \wedge r)$	$\sim (p \to ((p \vee q) \wedge r))$
T	T	T	T	T	F
T	T	F	F	F	T
T	F	T	T	T	F
T	F	F	F	F	T
F	T	T	T	T	F
F	T	F	F	T	F
F	F	T	F	T	F
F	F	F	F	T	F

It is sometimes true, sometimes false, depending on the truth assignment, by definition, the compound statement is a *contingency*. ..... [2 marks]

(b) Show that the statement $(p \to q \vee r)$ and the statement $(p \wedge q \to r)$ are not logically equivalent. (4 marks)

Solution: One can either construct a truth table or just give a counterexample below to show that they are not equivalent:

p	q	r	$p \to q \vee r$	$p \wedge q \to r$
T	T	T	T	T
T	T	F	T	F

..... [2 marks]

When $v(p)$ =T, $v(q)$ =T and $v(r)$ =F, the two statements has different truth values and they are not logically equivalent. [2 marks]

(c) Simplify the following statement

$$((p \vee q) \to (p \wedge q)) \vee (\sim p \wedge q).$$

to a logically equivalent statement with no more than TWO(2) logical connectives from the set $\{\sim, \wedge, \vee\}$ by stating the law used in each step of the simplification. (5 marks)

Solution: The steps are shown below:

$((p \vee q) \to (p \wedge q)) \vee (\sim p \wedge q)$
$\equiv (\sim (p \vee q) \vee (p \wedge q)) \vee (\sim p \wedge q)$ [Implication law, 1 mark]
$\equiv (\sim p \wedge \sim q) \vee (p \wedge q) \vee (\sim p \wedge q)$ [de Morgan law, 1 mark]

$\equiv (\sim p \wedge \sim q) \vee (p \wedge q) \vee (\sim p \wedge q) \vee (\sim p \wedge q)$ [Idempotent law, 1 mark]
$\equiv \sim p \wedge (q \vee \sim q) \vee ((p \vee \sim p) \wedge q)$ [Distributive law, 1 mark]
$\equiv \sim p \vee q$ [Negation and identity, 1 mark]

(d) Let $F(u, x, y)$, $G(y, v)$ and $H(x)$ be predicates. **List** down the steps and the logical equivalent rules to transform the following quantified statement

$$\sim \big[\forall x \exists y F(u, x, y) \to \exists x\big(\sim \forall y G(y, v) \to H(x)\big)\big]$$

to prenex normal form. (6 marks)

Solution: The steps and rules are listed below:

$\sim \big[\forall x \exists y F(u, x, y) \to \exists x\big(\sim \forall y G(y, v) \to H(x)\big)\big]$
$\equiv \sim \big[\sim \forall x \exists y F(u, x, y) \vee \exists x\big(\sim \forall y G(y, v) \to H(x)\big)\big]$ [Implication law, 0.5 mark]
$\equiv \forall x \exists y F(u, x, y) \bigwedge \sim \exists x\big(\sim \forall y G(y, v) \to H(x)\big)$ [de Morgan law, double negative, 1 mark]
$\equiv \forall x \exists y F(u, x, y) \bigwedge \Big[\forall x \sim \big(\sim \forall y G(y, v) \to H(x)\big)\Big]$ [Generalised de Morgan law, 0.5 mark]
$\equiv \forall x \exists y F(u, x, y) \bigwedge \Big[\forall x \sim \big(\forall y G(y, v) \vee H(x)\big)\Big]$ [Implication law, double negative, 1 mark]
$\equiv \forall x \exists y F(u, x, y) \bigwedge \Big[\forall x \big(\forall y \sim G(y, v) \wedge \sim H(x)\big)\Big]$ [Generalised de Morgan law, 0.5 mark]
$\equiv \forall x \exists y F(u, x, y) \bigwedge \Big[\forall x \forall y\big(\sim G(y, v) \wedge \sim H(x)\big)\Big]$ [Free variable law, 0.5 mark]
$\equiv \forall x \Big[\exists y F(u, x, y) \bigwedge \forall y\big(\sim G(y, v) \wedge \sim H(x)\big)\Big]$ [Quantified conjunctive law, 0.5 mark]
$\equiv \forall x \Big[\exists y F(u, x, y) \bigwedge \forall z\big(\sim G(z, v) \wedge \sim H(x)\big)\Big]$ [Quantifier renaming law, 0.5 mark]
$\equiv \forall x \exists y \forall z \Big[F(u, x, y) \wedge \sim G(z, v) \wedge \sim H(x)\Big]$ [Free variable law, 1 mark]

Example 1.13.15 (Final May 2019 Sem, Q2)**.** (a) Let p, q, r be atomic statements. Use a truth table or a comparison table to show that

$$(p \to r) \wedge (q \to r) \equiv (p \vee q) \to r.$$ (9 marks)

Solution: The comparison table is given below.

p	q	r	$(p \to r) \wedge (q \to r)$	$(p \vee q) \to r$
T	T	T	T	T
T	T	F	F	F
T	F	T	T	T
T	F	F	F	F
F	T	T	T	T
F	T	F	F	F
F	F	T	T	T
F	F	F	T	T

..... [8 marks]

Since the last two columns are the same for all different assignments, therefore, the two statements $(p \to r) \wedge (q \to r)$ and $(p \vee q) \to r$ are logically equivalent. ..... [1 mark]

(b) Simplify the following statement to a logically equivalent statement with no more than TWO(2) logical connectives from the set $\{\sim, \wedge, \vee\}$ by stating the law used in each step of the simplification:

$$(\sim p \wedge q) \vee (\sim p \wedge r) \vee (p \wedge \sim q \wedge r) \vee (q \wedge r). \qquad \text{(7 marks)}$$

Solution: The simplification is shown below:

$$\begin{aligned}
&(\sim p \wedge q) \vee (\sim p \wedge r) \vee (p \wedge \sim q \wedge r) \vee (q \wedge r). \\
&\equiv \Big(\sim p \wedge (q \vee r) \Big) \vee (p \wedge \sim q \wedge r) \vee (q \wedge r) && \text{[Distributive law, 1 mark]} \\
&\equiv (q \wedge r) \vee \Big((q \vee r) \wedge \sim p \Big) \vee (p \wedge \sim q \wedge r) \\
& && \text{[Associative and commutative laws, 1 mark]} \\
&\equiv (q \wedge r) \vee (p \wedge \sim q \wedge r) && \text{[Absorption law, 1 mark]} \\
&\equiv \Big(q \vee (p \wedge \sim q) \Big) \wedge r && \text{[Distributive law, 1 mark]} \\
&\equiv \Big((q \vee p) \wedge (q \vee \sim q)) \Big) \wedge r && \text{[Distributive law, 1 mark]} \\
&\equiv \Big((p \vee q) \vee T \Big) \wedge r && \text{[Negation law, 1 mark]} \\
&\equiv \Big(p \vee q \Big) \wedge r && \text{[Identity law, 1 mark]}
\end{aligned}$$

(c) Given the following quantified statement:

$$\forall x \forall y \big[((x > 0) \wedge (y > 0)) \to \big(\sqrt{x + y} = \sqrt{x} + \sqrt{y} \big) \big]. \qquad (*)$$

(a) Translate the quantified statement into an informal English sentence. (2 marks)

Solution: The square root of the sum of two numbers is equal to the sum of the square roots of the two numbers

(b) Determine whether the quantified statement is true or false in the domain of real numbers. You need to defend your answer. (2 marks)

Solution: The quantified statement is *false*. ..... [1 mark]
To defend, we write a counterexample: Let $x = y = 1$, $\sqrt{x + y} = \sqrt{2} \neq \sqrt{1} + \sqrt{1} = 2$. [1 mark]

(c) Write down the negation of the quantified statement $(*)$ in prenex normal form. (5 marks)

Solution: By applying the generalised de Morgan law, the negation of $(*)$ is logically equivalent to

$$\exists x \exists y \sim \big[((x > 0) \wedge (y > 0)) \to \big(\sqrt{x + y} = \sqrt{x} + \sqrt{y} \big) \big].$$

In prenex normal form, it can be written as

$$\exists x \exists y \big[(x > 0) \wedge (y > 0) \wedge \big(\sqrt{x + y} \neq \sqrt{x} + \sqrt{y} \big) \big]. \qquad \text{[5 marks]}$$

Example 1.13.16 (Final May 2019 Sem, Q3)**.** 1. Use **truth table** to explain whether the following argument is valid or invalid:

$$\begin{array}{rr} & (p \vee q) \to (p \wedge q) \\ & \sim (p \vee q) \\ \hline \therefore & \sim (p \wedge q) \end{array} \qquad \text{(9 marks)}$$

Solution: The truth table is

p	q	$(p \vee q) \rightarrow (p \wedge q)$	$\sim (p \vee q)$	$\sim (p \wedge q)$
T	T	T	F	F
T	F	F	F	T
F	T	F	F	T
F	F	T	T	T

..... [$4 \times 2 = 8$ marks]

We observe that when the premises are true (row 4), the conclusion is true, therefore, the argument is **valid**. ..... [1 mark]

2. Infer the argument

$$p \vee q,\ p \rightarrow r,\ \sim s \rightarrow \sim q \vdash r \vee s$$

syntatically by stating the **rules of inference** in each step. (6 marks)

Solution:

1	$p \vee q$	premise
2	$p \rightarrow r$	premise
3	$\sim s \rightarrow \sim q$	premise
4	p	assumption
5	r	2,4 $\rightarrow$E
6	$r \vee s$	5 $\vee$I$_1$
7	q	assumption
8	$\sim s$	assumption
9	$\sim q$	3,8 $\rightarrow$E
10	$\perp$	7,9 $\neg$E
11	s	8–10 $\neg$I
12	$r \vee s$	11 $\vee$I$_2$
13	$r \vee s$	1,4–6,6–12 $\vee$E

The p-assumption ..... [2 marks]

The q-assumption ..... [3 marks]

Line 12 ..... [1 mark]

3. Show that the following argument

$$\begin{array}{rl} & \forall x(F(x) \rightarrow \sim G(x)) \\ & \exists x(H(x) \wedge G(x)) \\ \hline \therefore & \exists x(H(x) \wedge \sim F(x)) \end{array}$$

is valid using the rules of logical equivalence and implication. (5 marks)

Solution: The semantic deduction is shown below

ϕ_1	$\forall x(F(x) \to\sim G(x))$	premise
ϕ_2	$\exists x(H(x) \wedge G(x))$	premise
ψ_1	$H(s) \wedge G(s)$	ϕ_2, existential instantiation ..... [1 mark]
ψ_2	$F(s) \to\sim G(s)$	ϕ_1, universal instantiation
ψ_3	$G(s)$	ψ_1, specialisation .....[1 mark]
ψ_4	$\sim F(s)$	ψ_2, ψ_3, MT .....[1 mark]
ψ_5	$H(s)$	ψ_1, specialisation .....[1 mark]
ψ_6	$H(s)\wedge \sim F(s)$	ψ_3, ψ_4 conjunction
$\therefore$	$\exists x(H(x)\wedge \sim F(x)$	ψ_6, existential generalisation ..... [1 mark]

4. Let $R(x, y)$ be a predicate with two variables. Infer the argument involving quantified statements

$$\forall x\forall y(R(x, y) \to\sim R(y, x)) \vdash \forall x(\sim R(x, x))$$

syntatically by stating the **rules of inference** in each step. (5 marks)

Solution: Let t be an arbitrary term independent of variables x and y.

1	$\forall x\forall y(R(x, y) \to\sim R(y, x))$	premise
2	$\forall y(R(t, y) \to\sim R(y, t))$	1 $\forall$-elimination .....[1 mark]
3	$R(t, t) \to\sim R(t, t)$	2 $\forall$-elimination
4	$R(t, t)$	assumption .....[1 mark]
5	$\sim R(t, t)$	3,4 $\to$I .....[1 mark]
6	$\perp$	4,5 $\neg$E
7	$\sim R(t, t)$	4–6 $\neg$I ..... [1 mark]
8	$\forall x(\sim R(x, x))$	7 $\forall$-introduction .....[1 mark]

Example 1.13.17 (Final May 2021 Sem, Q1 (during MCO)). (a) Let p, q, r, s be atomic statements. **State** the truth table for the following compound statement

$$(\sim (p \leftrightarrow q)) \wedge (r \to s) \wedge (s \to (\sim (\sim p \to q)))$$

and determine if the compound statement is tautology, contradiction or contingency. (4 marks)

Solution: Let P_1 be the compound statement. The truth table for the compound statement is stated below .....[3.5 marks]

p	q	r	s	$\sim (p \leftrightarrow q)$	$r \rightarrow s$	$s \rightarrow (\sim (\sim p \rightarrow q))$	P_1
T	T	T	T	F	T	F	F
T	T	T	F	F	F	T	F
T	T	F	T	F	T	F	F
T	T	F	F	F	T	T	F
T	F	T	T	T	T	F	F
T	F	T	F	T	F	T	F
T	F	F	T	T	T	F	F
T	F	F	F	T	T	T	T
F	T	T	T	T	T	F	F
F	T	T	F	T	F	T	F
F	T	F	T	T	T	F	F
F	T	F	F	T	T	T	T
F	F	T	T	F	T	T	F
F	F	T	F	F	F	T	F
F	F	F	T	F	T	T	F
F	F	F	F	F	T	T	F

Since the compound statement is sometimes true and sometimes false depending on the truth assignment, the compound statement is a contingency. ..... [0.5 mark]

(b) Let p, q, r and s be atomic statements. Simplify the following statement

$$(\sim p \wedge q \wedge (\sim s \vee (\sim r \wedge s))) \vee (q \wedge (p \vee (\sim p \wedge r \wedge s)))$$

to a logically equivalent statement with NO logical connectives by stating the law used in each step of the simplification. Write the important laws used clearly. (4 marks)

Solution:

$$\begin{aligned}
& (\sim p \wedge q \wedge (\sim s \vee \sim r \wedge s)) \vee (q \wedge (p \vee \sim p \wedge r \wedge s)) \\
\equiv\ & q \wedge ((\sim p \wedge (\sim s \vee \sim r \wedge s)) \vee (p \vee (\sim p \wedge r \wedge s)) && \text{[Distributive law, 0.5 mark]} \\
\equiv\ & q \wedge ((\sim p \wedge (\sim s \wedge \sim r \vee \sim s \wedge r \vee \sim r \wedge s)) \vee (p \vee (\sim p \wedge r \wedge s)) \\
& && \text{[Identity, negation, distributive, 0.5 mark]} \\
\equiv\ & q \wedge ((\sim p \wedge (\sim s \wedge (\sim r \vee r) \vee \sim r \wedge (\sim s \vee s))) \vee (p \vee (\sim p \wedge r \wedge s)) \\
& && \text{[Idempotent and distributive, 0.5 mark]} \\
\equiv\ & q \wedge ((\sim p \wedge (\sim s \vee \sim r)) \vee (p \vee (\sim p \wedge r \wedge s)) \\
& && \text{[Negation and identity, 0.5 mark]} \\
\equiv\ & q \wedge ((\sim p \wedge (\sim s \vee \sim r)) \vee ((p \vee \sim p) \wedge (p \vee r \wedge s))) && \text{[Distributive, 0.5 mark]} \\
\equiv\ & q \wedge ((\sim p \wedge (\sim s \vee \sim r)) \vee (T \wedge (p \vee r \wedge s))) && \text{[Negation, 0.5 mark]} \\
\equiv\ & q \wedge (\sim (p \vee (r \wedge s)) \vee (p \vee (r \wedge s))) && \text{[Identity, De Morgan} \times 2\text{, 0.5 mark]} \\
\equiv\ & q && \text{[Negation, identity, 0.5 mark]}
\end{aligned}$$

(c) Predicate logic can be used to perform computations using the notion of recursion. Suppose that the predicate that returns the result of the following summation

$$S = 1 + \frac{1}{2} + \cdots + \frac{1}{n} = \sum_{i=1}^{n} \frac{1}{i}$$

is `mysum(n, S)`. Write down the recursive expression for the predicate `mysum( n, S)` in an appropriate formal predicate logic formulation (alternatively, using Prolog formulation). (2 marks)

Solution: The predicates to express `mysum(n, S)` in Prolog formulation are stated below [2 marks]

```
mysum(1, S) :- S is 1.
mysum(N, S) :- N > 1, N1 is N-1, mysum(N1,S1), S is (1/N)+S1.
```

[Total: 10 marks]

Example 1.13.18 (Final May 2021 Sem, Q2 (during MCO)). (a) Use **comparison table** or **truth table** to show that the following argument is **invalid**:

$$\begin{array}{r} p \to q \vee r \\ (q \wedge r) \leftrightarrow s \\ \sim (\sim q \wedge s) \\ \hline \therefore \quad p \vee (q \to r) \end{array}$$

(4 marks)

Solution: A comparison table is listed below.

p	q	r	s	$p \to q \vee r$	$(q \wedge r) \leftrightarrow s$	$\sim (\sim q \wedge s)$	$p \vee (q \to r)$
T	T	T	T	T	T	T	T
T	T	T	F	T	F	T	T
T	T	F	T	T	F	T	T
T	T	F	F	T	T	T	T
T	F	T	T	T	F	F	T
T	F	T	F	T	T	T	T
T	F	F	T	F	F	F	T
T	F	F	F	F	T	T	T
F	T	T	T	T	T	T	T
F	T	T	F	T	F	T	T
F	T	F	T	T	F	T	F
F	T	F	F	T	T	T	F
F	F	T	T	T	F	F	T
F	F	T	F	T	T	T	T
F	F	F	T	T	F	F	T
F	F	F	F	T	T	T	T

..... [3.5 marks]

Since the row 12 has a false conclusion with all true premises, the argument is invalid. [0.5 mark]

(b) Let p, q, r, s, t, u, v be atomic propositions. Show that the argument below

$$(p \wedge \sim q) \to r, \quad (s \vee u) \to p, \quad q \to t, \quad s \to \sim t, \quad s \wedge v \ / \ \therefore r$$

is valid using the **laws of logical implication** and **laws of logical equivalence** only. (3 marks)

Solution:

ϕ_1 :	$(p \wedge \sim q) \to r$	premise
ϕ_2 :	$(s \vee u) \to p$	premise
ϕ_3 :	$q \to t$	premise
ϕ_4 :	$s \to \sim t$	premise
ϕ_5 :	$s \wedge v$	premise
ψ_1 :	s	ϕ_5, Specialisation
ψ_2 :	$\sim t$	ψ_1, ϕ_4, Modus Ponens
ψ_3 :	$\sim q$	ψ_2, ϕ_1, Modus Tollens
ψ_4 :	$s \vee u$	ψ_1, Generalisation
ψ_5 :	p	ψ_4, ϕ_2, Modus Ponens
ψ_6 :	$p \wedge \sim q$	ψ_3, ψ_5, Conjunction
ξ :	r	ψ_6, ϕ_1, Modus Ponens

.....[3 marks]

(c) Use the **rules of inference ONLY** to show that

$$((A \to B) \to (C \to D)) \vdash C \to (B \to D).$$

Write the steps using Fitch style proof. In each step, state the rules of inference.

(3 marks)

Solution: The **syntatic** inference is shown below. .....[$2 \times 0.5 + 5 \times 0.4 = 3$ marks]

1	$(A \to B) \to (C \to D)$	premise.....[0.5 mark]
2	$\quad C$	assumption.....[0.5 mark]
3	$\quad\quad B$	assumption
4	$\quad\quad\quad A$	assumption
5	$\quad\quad B$	3
6	$\quad\quad A \to B$	4–5 $\to$I.....[0.4 mark]
7	$\quad\quad C \to D$	6,1 $\to$E.....[0.4 mark]
8	$\quad\quad D$	2,7 $\to$E.....[0.4 mark]
9	$\quad B \to D$	3,8 $\to$I.....[0.4 mark]
10	$C \to (B \to D)$	2,9 $\to$I.....[0.4 mark]

[Total: 10 marks]

Example 1.13.19 (Final May 2024 Sem, Q1)**.** (a) Write formally the converse, inverse, negation and contrapositive of the following compound statement.

"If 6 is even or 11 is odd, then $11 \equiv 6(\mod 5)$."

Hence, determine their truth values. Give reason(s) to support your answer.

Recall Definition 1.1.8: For $p \to q$, converse is $q \to p$, inverse is $\sim p \to \sim q$, negation is $p \wedge \sim q$ and contrapositive is $\sim q \to \sim p$.

Solution: Let p denote "6 is even" ($v(p) = \mathrm{T}$) q denote "11 is odd" ($v(q) = \mathrm{T}$) and r denote $11 \equiv 6 \pmod 5$ (since $11 = 5 \times 1 + 6$, $v(r) = \mathrm{T}$).

The compound statement can be expressed formally as

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

and

$$v((p \vee q) \to r) = (\mathrm{T} \vee \mathrm{T}) \to \mathrm{T} = \mathrm{T}.$$

The converse is

$$r \to (p \vee q) \qquad \text{If } 11 \equiv 6 \pmod 5, \text{ then 6 is even or 11 is odd.}$$

and

$$v(r \to (p \vee q)) = \mathrm{T} \to (\mathrm{T} \vee \mathrm{T}) = \mathrm{T}.$$

The inverse is

$$\sim (p \vee q) \to \sim r \qquad \text{If 6 is not even and 11 is not odd, then } 11 \not\equiv 6 \pmod 5.$$

and

$$v(\sim (p \vee q) \to \sim r) = \sim (\mathrm{T} \vee \mathrm{T}) \to \sim \mathrm{T} = \mathrm{F} \to \mathrm{F} = \mathrm{T}.$$

The negative is

$$(p \vee q) \wedge \sim r \qquad \text{6 is even or 11 is odd, but } 11 \not\equiv 6 \pmod 5.$$

and

$$v((p \vee q) \wedge \sim r) = (\mathrm{T} \vee \mathrm{T}) \wedge \sim \mathrm{T} = \mathrm{T} \wedge \mathrm{F} = \mathrm{F}.$$

The contrapositive is

$$\sim r \to \sim (p \vee q) \qquad \text{If } 11 \not\equiv 6 \pmod 5, \text{ then 6 is not even and 11 is not odd.}$$

and

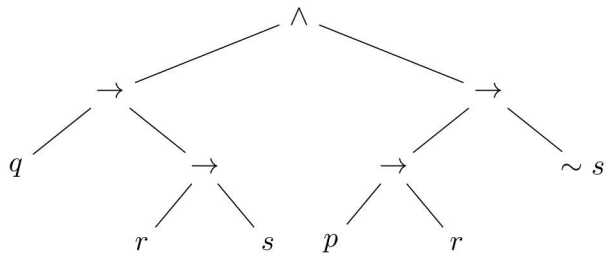
$$v(\sim r \to \sim (p \vee q)) = \sim \mathrm{T} \to \sim (\mathrm{T} \vee \mathrm{T}) = \mathrm{F} \to \mathrm{F} = \mathrm{T}.$$

(b) Suppose $p \to q$ is false and $r \wedge s$ is true. Find the truth value of

$$(q \to (r \to s)) \wedge ((p \to r) \to (\sim s)). \qquad \text{(3 marks)}$$

Given $v(p \to q) = \mathrm{F}$ and $v(r \wedge s) = \mathrm{T}$. This means

$$v(p) = \mathrm{T}, \quad v(q) = \mathrm{F}, \quad v(r) = \mathrm{T}, \quad v(s) = \mathrm{T}.$$



Since $v(q) = \mathrm{F}$, left subtree is T; Since $v(p \to r) = \mathrm{T}$ and $v(\sim s) = \mathrm{F}$, right subtree is F and

$$v((q \to (r \to s)) \wedge ((p \to r) \to (\sim s))) = \mathrm{F}.$$

(c) Determine whether the compound statement

$$\phi = (p \to q) \wedge [(\sim p \vee \sim q) \to (p \wedge q \wedge r)]$$

is a tautology or contradiction by using a truth table and also law of logic. Show all steps clearly and provide explanations for each step. (12 marks)

Solution: The lecturer probably missed out the word "contingency" from the question.

p	q	r	$p \to q$	$(\sim p \vee \sim q)$	$p \wedge q \wedge r$	ϕ
T	T	T	T	F	T	T
T	T	F	T	F	F	T
T	F	T	F	T	F	F
T	F	F	F	T	F	F
F	T	T	T	T	F	F
F	T	F	T	T	F	F
F	F	T	T	T	F	F
F	F	F	T	T	F	F

From the truth table, ϕ is a **contingency**.

Using laws of logical equivalences, we can simplify the contingency

$\phi \equiv (p \to q) \wedge [\sim (p \wedge q) \to (p \wedge q \wedge r)]$ [De Morgan]
$\equiv (\sim p \vee q) \wedge [(p \wedge q) \vee ((p \wedge q) \wedge r)]$ [Implication & Double Negation]
$\equiv (\sim p \vee q) \wedge [p \wedge q]$ [Absorption]
$\equiv ((\sim p \wedge p) \vee (p \wedge q)) \wedge q$ [Associative & Distributive]
$\equiv (p \wedge q) \wedge q$ [Negative & Identity]
$\equiv p \wedge q$ [Associative & Idempotent]

Example 1.13.20 (Final May 2024 Sem, Q2)**.** (a) State and prove the Modus Tollens. (7 marks)

Solution: MT: $p \to q,\ \sim q \models \sim p$

Semantic proof: Use truth table.

Syntactic proof: Refer to Example 1.7.2.

(b) Determine the validity of the following argument:

(isPrime(2) $\rightarrow$ isEven(2)) $\wedge$ (isComposite(6) $\rightarrow$ canFly(dog)
(canFly(dog) $\vee$ isEven(2)) $\rightarrow \sim$ in(Ipoh, Malaysia)
in(Ipoh, Malaysia)

$\therefore$ isComposite(2) or isPrime(6)

Semantic Proof: Proving the argument using natural deduction would be too tedious, we will prove using laws of equivalences and laws of implication.

ϕ_1 :	(isPrime(2) $\rightarrow$ isEven(2)) $\wedge$ (isComposite(6) $\rightarrow$ canFly(dog))	premise
ϕ_2 :	(canFly(dog) $\vee$ isEven(2)) $\rightarrow\sim$ in(Ipoh, Malaysia)	premise
ϕ_3 :	in(Ipoh, Malaysia)	premise
ψ_1 :	$\sim$ (canFly(dog) $\vee$ isEven(2))	Modus Tollens on ϕ_2, ϕ_3
ψ_2 :	$\sim$ (canFly(dog)$\wedge \sim$ isEven(2)	De Morgans on ψ_1
ψ_3 :	$\sim$ isEven(2)	Specialisation on ψ_2
ψ_4 :	isPrime(2) $\rightarrow$ isEven(2)	Specialisation on ϕ_1
ψ_5 :	$\sim$ isPrime(2)	Modus Tollens on ψ_3, ψ_4
$\therefore$	$\sim$ isPrime(2) $\vee$ isPrime(6)	Generalisation on ψ_5

Note that the lecture implicitly assume that $\sim$isPrime(x) is equivalent to isComposite(x) for x$\geq$ 2.

(c) Give a negation for the following statement:

"All even integers can be divided by 2".

Determine the truth value for the following statement

$$\phi := \text{“}\exists x \in \mathbb{Z}^+, \exists y \in \mathbb{Z}^+, x + y \text{ is even.”}$$

and justify your answer. (6 marks)

Solution: Formally,

$$\forall x(even(x) \rightarrow 2|x)$$

The negation is

$$\sim \forall x(even(x) \rightarrow 2|x) \equiv \exists x \sim (even(x) \rightarrow 2|x) \equiv \exists x(even(x) \wedge \sim (2|x))$$

which informally reads (compare to Example 1.9.5)

Some even integers cannot be divided by 2.

ϕ is false because the negative of ϕ below is true:

$$\forall x \exists y(x + y \text{is not even}).$$

We only need to take $y = x + 1$ and $x + y = 2x + 1$ is not even.

References

S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Database*. Addison-Wesley, 1995. `http://webdam.inria.fr/Alice/`.

A. V. Aho and J. D. Ullman. *Foundations of Computer Science*. Computer Science Press, Inc., New York, NY, USA, 1992. ISBN 0-7167-8233-2.

A. W. Appel, R. Dockins, A. Hobor, L. Beringer, J. Dodds, G. Stewart, S. Blazy, and X. Leroy. *Program Logics For Certified Compilers*. Cambridge University Press, 2014.

D. W. Barnes and J. M. Mack. *An Algebraic Introduction to Mathematical Logic*, volume 22 of *Graduate Texts in Mathematics*. Springer-Verlag New York Inc., 1975.

J. Barwise and J. Etchemendy. *Language, Proof and Logic*. CSLI Publications, 1999. (a introduction to logic).

D. Bjørner. *Software Engineering 1: Abstraction and Modelling*. Springer-Verlag Berlin Heidelberg, 2006.

N. Bjørner and L. de Moura. z3[10]: Applications, enablers, challenges and directions. In *Sixth International Workshop on Constraints in Formal Verification Grenoble, France*, 2009.

G. S. Boolos, J. P. Burgess, and R. C. Jeffrey. *Computability and Logic*. Cambridge University Press, 5th edition, 2007.

R. Bornat. *Proof and Disproof in Formal Logic: An introduction for programmers*. Oxford University Press, 2005.

I. Chiswell and W. Hodges. *Mathematical Logic*. Oxford University Press, 2007. (a really nice and rigorous introduction to logic).

W. F. Clocksin. *Clause and Effect: Prolog Programming for the Working Programmer*. Springer-Verlag Berlin Heidelberg, 1997.

K. Devlin. *The Joy of Sets: Fundamentals of Comtemporary Set Theory*. Springer-Verlag New York, Inc., 2nd edition, 1993.

S. S. Epp. *Discrete Mathematics with Applications*. Cengage Learning Inc., 5th edition, 2020.

T. Forster. *Logic, Induction and Sets*. London Mathematical Society Student Texts. Cambridge University Press, 2003. doi: 10.1017/CBO9780511810282. URL `https://www.dpmms.cam.ac.uk/~tf/baby.html`.

J. H. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Dover Publications, 2015.

P. R. Halmos. *Naive Set Theory*. Van Nostrand Reinhold, 1960.

J. Harrison. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, 2009.

R. A. Herrmann. Logic for everyone. arXiv:0601709, 29 Jan 2006.

P. G. Hinman. *Fundamentals of Mathematical Logic*. A. K. Peters, Ltd., 2005. (abstract mathematics).

M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2nd edition, 2004.

P. T. Johnstone. *Notes on Logic and Set Theory*. Cambridge University Press, 1987.

M. Kac, G.-C. Rota, and J. T. Schwartz. *Discrete Thoughts: Essays on Mathematics, Science, and Philosophy*. Birkhäuser Boston, 2nd edition, 2008.

T. Kropf. *Introduction to Formal Hardware Verification*. Springer-Verlag Berlin Heidelberg, 1999.

L. Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers.* Addison-Wesley, 2003.

Y. I. Manin. *A Course in Mathematical Logic*, volume 53 of *Graduate Texts in Mathematics.* Springer-Verlag New York Inc., 1977.

E. Mendelson. *Introduction to Mathematical Logic.* Chapman & Hall, 4th edition, 1997.

D. Merritt. *Building Expert Systems in Prolog.* Amzi! inc., 2000.

N. Nisan and S. Schocken. *The Elements of Computing Systems: Building a Modern Computer from First Principles.* MIT Press, 2005.

Open Logic Project. The open logic text. http://openlogicproject.org/, 2019.

W. Rautenberg. *A Concise Introduction to Mathematical Logic.* Universitext. Springer Science+Business Media, LLC, 3rd edition, 2010. (abstract mathematics).

U. Schöning and J. Torán. *The Satisfiability Problem: Algorithms and Analysis.* Lehmanns Media: Berlin, 2013.

L. Sterling and E. Shapiro. *The Art of Prolog: Advanced Programming Techniques.* The MIT Press, 2nd edition, 1999.

J. D. Ullman and J. Widom. *A First Course in Database Systems.* Pearson Education, Inc., 3th edition, 2008.

T. van Le. *Techniques of PROLOG PROGRAMMING: With Implementation of Logical Negation and Quantified Goals.* John Wiley & Sons, Inc., 1993.

R. Woods, J. McAllister, G. Lightbody, and Y. Yi. *FPGA-based Implementation of Signal Processing Systems.* John Wiley & Sons, Ltd, 2008.