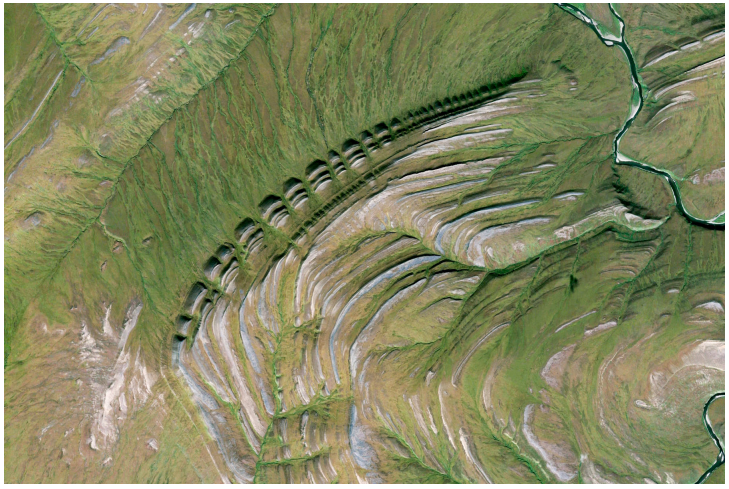
Chapter 1

LOGIC THEORY

With Application in Computing



MATHEMATICAL MODELING- AN OVERVIEW

CSE = *Computer Science and Engineering* = COMPUTING (by ACM) [1]

- The text provides the **mathematical foundations for many CS areas**, e.g.,
  algorithm analysis & design, artificial intelligence (AI),
  data analytics [interacting with AI and Statistics, Traffic Engineering],
  process analytics [interacting with Smart Manufacturing], etc.
  Applications of Logic, Discrete Modeling & Optimization in CSE will be highlighted.
- The next part of this text introduces CSE students to automata and formal language, then the basic concepts of logic (e.g., theories, models, logical consequence, and proof) together with application in theoretically software verification.
- In the third part, students will be learned mathematical modeling through integer optimization, particularly integer linear programming (ILP).
- Finally, to see connection with newly developed areas/ fields we learn stochastic programming, dynamical systems, and Petri net (extra reading).

Key Reference texts of the course

1. REF 1. Lecture notes and text of the instructor (like this document)
2. REF 2. Michael Huth & Mark D. Ryan. **Logic in Computer Science** (2nd Ed.), Cambridge, 2004. (Ch. 1, 2)
   REF 2A. Michael R.A. Huth and Mark D. Ryan. Logic in Computer Science: *Solutions to designated exercises* (2nd Ed.), Cambridge University Press, 2004. (Chapters 1, 2)
3. Matousek et al. Understanding and using linear programming, Springer, 2007.
4. Peter Linz. *An Introduction to Formal Languages and Automata* 2001. (**Chap. 1-6**)

[1] ACM: Association for Computing Machinery

5. John Hopcroft et. al. *Introduction to Automata Theory, Languages & Computation* (3rd Ed.)
6. W.M.P. Aalst, van der, *PROCESS MINING*, 2nd edition, 2016, Springer, Chapter 5
7. Cleiton dos Santos Garcia et. al. *Process Mining Techniques and Applications- A systematic Mapping Study*, in Expert Systems With Applications, **Vol 133** (2019) 260-295, Elsevier,

1.1 PART I: Propositional Logic

OVERVIEW: Logic, and Discrete Optimization

Mathematical Logic nowadays finds applications in many areas of computing.

- **The laws of logic** are employed in the design of the digital circuitry in a computer.
- Logical expressions occur as conditions in the control structures in *algorithms* and computer programs, and in the commands used for **querying databases**.
- **Expert systems** employing knowledge-based software use rules of logical inference to draw conclusions from known facts.

LOGIC THEORY- **The Blueprint**

PART I. Propositional Logic (review), Section 1.1

PART II. Advanced Propositional Logic, Section 1.2

PART III. Predicate Logic, in Section 1.3

PART IV (1.4)- ADVANCED LOGIC THEORY with APPLICATION includes

1. Advanced Predicate Logic
2. Logic for Program Verification
3. Hoare Triples

We remind Propositional Logic:

- Propositions– Propositions versus predicate
- Connectives and truth tables - Compound propositions
- Tautology– Contradiction – Logical equivalence
- Decomposing complicated proposition
- Semantics of propositional logic
- Natural deduction and Provable equivalence- Rule of Inference

We begin by looking at examples involving everyday English sentences. This is followed by an introduction to the more formal mathematical approach used in **propositional** and **predicate** logic.

Propositions- Propositional logic

The fundamental objects we worked in Discrete Structure course are **set, graph**, and in **arithmetic** are *numbers*. Similarly, the fundamental objects in **logic** are *propositions*.

Definition 1.1 ♣

A proposition is a statement that is either *true* or *false*. Whichever of these (true or false) is the case is called the **truth value** of the proposition.

Here are some examples of English sentences that are propositions:

- 'Canberra is the capital of Australia.'
- 'There are 8 days in a week.'
- 'Isaac Newton was born in 1642.'

- '5 is greater than 7.'
- 'Every even number greater than 2 is expressed as the sum of two prime numbers.'

Propositions– truth values

The first and third of these propositions are true, and the second and fourth are false. Is the last truly proposition? It is **not known** at present whether the **fifth proposition** is true or false.

The following sentences are not propositions:

'*Where are you going*?' OR '*Come here*!'

How about these?

'*Anne is tall*.'

'*Ice cream is delicious*.'

'$x > 5$.'

The last of the three sentences given above is an example of a **predicate**.

Propositions versus predicate

Definition 1.2

A predicate is a statement containing one or more variables; it cannot be assigned a truth value until the values of the variables are specified. We will investigate *predicate logic* later.

Statements containing variables commonly occur in algorithms and computer programs.

For example, an algorithm might contain the statement '$x > 5$'

treated as the condition in a *control structure* such as an **If-then**.

In this case, however, the truth value of the statement is determined when the line is executed while the program is being run with a particular set of inputs,

so statements of this type can be treated as propositions.

1.1.1 Connectives

Logic is not concerned with determining the truth values of propositions from specific application information.

(E.g., the truth value of '*Canberra is the capital of Australia*'

is a question of **geography**, not **logic**.)

The next example is different, however:

'***If*** *Brian and Angela are not both happy*,

then *either Brian is not happy or Angela is not happy*.'

The sentence about Brian and Angela is an example of a *compound proposition*. It is built up from the *atomic propositions*

- '*Brian is happy*' and
- '*Angela is happy*'

using the words **and**, **or**, **not** (also written $\neg$) and **if-then**.

Connectives and truth tables

These words are known as *connectives*. As we will see, the role of connectives in logic is analogous to the role played by operations such as $+$ and $\times$ in algebra.

Atomic propositions.

The study of the structure of *compound propositions* is made easier by the use of symbols for atomic propositions and connectives.

■ We will use lower-case letters such as p, q and r to denote *atomic propositions*.

There are five connectives that we will use in our work; they are listed in table below, together with their symbols.

Connective	*Symbol*
and	$\wedge$
or	$\vee$
not	$\neg$
if-then	$\rightarrow$
if-and-only-if	$\leftrightarrow$

Table 1.1: *Connectives and associated symbols*

1.1.2 Compound propositions

With the exception of **not**, the symbols for these connectives are written between the two operands (the propositions they connect).

For example, if

p denotes the proposition '*Today is Monday*', and

q denotes the proposition '*It is raining*',

then we can write

- the symbol $\overline{p}$, $\neg p$ or **not** p before the proposition to which it applies; thus,

  $\neg p$ means '*Today is **not** Monday*'.
- the symbol $p \wedge q$ to denote the proposition

  '*Today is Monday and it is raining*'.

Compound propositions– $p \wedge q$ has the true value given in table below

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

Table 1.2: *Connective **and***

Can we form compound propositions $p \vee q$, and $p \rightarrow q$ similarly?

Writing compound propositions in *symbolic form*

We now have the notation we need in order to be able to write compound propositions in *symbolic form*. Below example shows how this is done.

♦ **EXAMPLE 1.1.** *Express the proposition below in symbolic form:*

*'**Either** my program runs and it contains no bugs, **or** my program contains bugs'*

Truth table for an expression

♦ **EXAMPLE 1.2.** *Construct the truth table for the expression* $(p \wedge \neg q) \vee q$.

- Notice that each column is obtained using the truth table for the principal connective in the expression at the top of the column.
- If an expression contains three variables (p, q and r, say), then the table will have eight lines instead of four (there are $2^3 = 8$ different ways of allocating truth values to three expressions), but the method is the same.

1.1.3 Tautology

♦ **EXAMPLE 1.3.** *Look back the proposition we introduced at the beginning of this section*

***If** Brian and Angela are not both happy,*

***then** either Brian is not happy or Angela is not happy*

a/ Write this compound proposition in *symbolic form* and

b/ provide its truth table.

Definition 1.3

(A) An expression that is always true, regardless of the truth values of the variables it contains, is called a ***tautology****.*

(B) An expression that is always false, regardless of the truth values of the variables it contains, is called a ***contradiction****.*

Mathematically , contradictions are expressions of the form $\phi \wedge \neg\phi$ *or* $\neg\phi \wedge \phi$

where ϕ *is any proposition (simple or compound).*

(C) ***Logical equivalence****: Two expressions (composed of the same variables) are* ***logically equivalent*** *if they have the same truth values for every combination of the truth values of the variables. Formally,* $A(p, q, r, ...)$ *is logically equivalent to* $B(p, q, r, ...)$*, write*

$$A(p, q, r, ...) \Leftrightarrow B(p, q, r, ...)$$

if we have $A(p, q, ...) = B(p, q, ...)$ *for every choice of boolean values of* $p, q, ...$

♣

Definition 1.4 (Converse and Contrapositive)

♣

- Expressions of the form $p \to q$ are called *implications*.
- The *converse* of $p \to q$ is $q \to p$
- The *contrapositive* of $p \to q$ is $\neg q \to \neg p$

♦ **EXAMPLE 1.4.** *Can you write English sentences for the converse and the contrapositive of the statement?* **If 250 is divisible by 4 then 250 is an even number**.

Logical equivalence and connective if-and-only-if

There is a subtle but important distinction between

- the connective **if-and-only-if** or $\leftrightarrow$
- and the concept of **logical equivalence** $\Leftrightarrow$.

When we write $A \leftrightarrow B$, we are writing a *single* logical expression.

$$(A \rightarrow B) \wedge (B \rightarrow A).$$

Hence the value is either $(A \leftrightarrow B) \equiv TRUE$ or $(A \leftrightarrow B) \equiv FALSE$.

Logical equivalence, $A \Longleftrightarrow B$, read [expressions A and B are logically equivalent]

on the other hand, is a relationship between A and B: they are both TRUE or FALSE.

Connection of the two concepts:

A and B are logically equivalent if and only if the expression $A \leftrightarrow B$ is a tautology.

1.1.4 Decomposing proposition

We began this section with an example of a **complicated proposition** that we showed to be logically equivalent to a simpler one. Occasions often arise in practice where it is desirable to replace a logical expression with a **simpler expression** that is logically equivalent to it.

For example, we have seen how logical expressions representing propositions can occur in *algorithms* and *computer programs*. By writing these expressions as simply as possible, we can make a program **more efficient** and reduce the chance of error.

How do we do? Use Laws of Logic (Logical equivalence)

In order to be able to simplify logical expressions effectively, we need to establish a list of pairs of expressions that are *logically equivalent*.

We will use the symbol $\equiv$ or $\Leftrightarrow$ placed between two expressions to indicate that they are equivalent. A statement of the form

$$P \equiv Q$$

or

$$P \Leftrightarrow Q$$

where P and Q are logical expressions is called a **law of logic**.

A list of the most important laws of logic is given as follows.

Most important laws

De Morgan	$\neg(\neg p \wedge \neg q)$	$\equiv$	$p \vee q$
Contrapositive	$p \to q$	$\equiv$	$\neg q \to \neg p$
Equivalence law	$p \leftrightarrow q$	$\equiv$	$(p \to q) \wedge (q \to p)$
Implication law	$p \to q$	$\equiv$	$\neg p \vee q$
Double negation law	$\neg(\neg p)$	$\equiv$	p
Idempotent laws	$p \wedge p$	$\equiv$	p
	$p \vee p$	$\equiv$	p
Commutative laws			?
Associative laws			?
Distributive laws			?
Inverse laws			?
Absorption laws			?

$p \vee q$	$\equiv$	$q \vee p$	Commutative laws
$p \wedge q$	$\equiv$	$q \wedge p$	
$(p \vee q) \vee r$	$\equiv$	$p \vee (q \vee r)$	Associative laws
$(p \wedge q) \wedge r$	$\equiv$	$p \wedge (q \wedge r)$	
$p \vee (q \wedge r)$	$\equiv$	$(p \vee q) \wedge (p \vee r)$	Distributive laws
$p \wedge (q \vee r)$	$\equiv$	$(p \wedge q) \vee (p \wedge r)$	
$\neg(p \wedge q)$	$\equiv$	$\neg p \vee \neg q$	De Morgan's law
$\neg(p \vee q)$	$\equiv$	$\neg p \wedge \neg q$	
$p \vee (p \wedge q)$	$\equiv$	p	Absorption laws
$p \wedge (p \vee q)$	$\equiv$	p	

Using important laws to simplify logical expressions

QUIZ

1. **Converse and contrapositive**. (5 min) Prove that

   $p \to q$ is not logically equivalent to its converse, but that it is logically equivalent to its contrapositive. That is

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

2. Use the laws of logic to simplify the expression:

   $$p \vee \neg(\neg p \to q)$$

3. An algorithm contains the following line:

$$\textbf{If} \quad \textbf{not} \quad (x > 5 \quad \textbf{and} \quad x \leq 10) \quad \textbf{then}...$$

How could this be written more simply?

1.1.5 Semantics of propositional logic

REMINDER: There are two truth values, T and F, denoted for *truth* and *falsity*.

The meaning of logical connectives: The propositional connectives are:

- negation ($\neg$),
- conjunction $\wedge$
- disjunction $\vee$
- implication $\rightarrow$ or $\Rightarrow$
- biimplication $\Leftrightarrow$

and be read as

- “not”,
- “and”,
- “or”,
- “if-then”,
- “if and only if” respectively.

The meaning of logical connectives

- $\neg r$: the negation of r
- $p \wedge q$: the conjunction of p and q
- $p \vee q$: ?
- $p \rightarrow q$: the implication of q by p
- $a \Rightarrow b$: the logical implication of b by a
- $p \leftrightarrow q$: equivalence

$p \leftrightarrow q$ means:

- "p is `necessary and sufficient` for q".
- "if p then q, and `conversely`".
- "p (if-and-only-if) `iff` q".

The connectives $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$ are designated as *binary*,

while $\neg$ is designated as *unary*.

Practice- Translating Natural Sentences

Consider a sentence as follows.

`I will buy a new phone only if I have enough money to buy iPhone 4` **or** `my phone is not working.`

If define

- p: I will buy a new phone
- q: I have enough money to buy iPhone 4
- r: My phone is working

then its symbolic form is: $p \rightarrow (q \vee \neg r)$

But if consider the sentence:

"`I will buy a new phone if I have enough money to buy iPhone 4` **or** `my phone is not working`".

What answer (symbolic form) do you get?

Summary of PART I

Let p, q be arbitrary statements. We distinguish few cases:

- The relationship between $\rightarrow$ and $\Rightarrow$

  $p \rightarrow q$: the implication of q by p, or p implies q

  $p \Rightarrow q$: logical implication, p logically implies q.

  We understand that: if $p \rightarrow q$ is a tautology, then we say

  p logically implies q, and write $p \Rightarrow q$.

- The relationship between $\leftrightarrow$ and $\Leftrightarrow$

  $p \leftrightarrow q$: bi-implication of p and q, p if and only if q

  $p \Leftrightarrow q$: logical equivalence, p is logically equivalent to q

  We understand that:

  if $p \leftrightarrow q$ is a tautology, then we say

  p is logically equivalent to q, and write $p \Leftrightarrow q$.

  How did it go?

- Logical equivalence $\Leftrightarrow$ means bi-implication $\leftrightarrow$ is tautology.
- Five rules for infering

$$
\begin{array}{ll}
 & p \\
 & p \to q \\
\hline
\therefore & q \\
\hline
 & p \to q \\
 & q \to r \\
\hline
\therefore & p \to r
\end{array}
$$

...

PRACTICE 1.1. *(30 min).*

1. Determine whether the following argument is valid:

   '*The file is either a binary file or a text file. If it is a binary file then my program won't accept it. My program will accept the file. Therefore the file is a text file.*'

2. Translating Natural Sentences.

   He will not run the red light if he sees the police unless he is too risky.

3. System specifications

   - "When a user clicked on *Help* button, a pop-up will be shown up"

4. **Logic puzzles**

   - There are two kinds of inhabitants on an island, `knights, who always tell the truth`, and their opposites, `knaves, who always lie`.

     You encounter two people A and B.

What are A and B if A says "B **is a knight**" and B says "**The two of us are opposite types**"?

Hints for PRACTICE

An argument of this type consists of some *premises* (in this example, the first three sentences), which together are supposed to imply the *conclusion* (the last sentence). The argument takes the form of the logical expression:

$$(P_1 \wedge P_2 \wedge P_3) \rightarrow Q$$

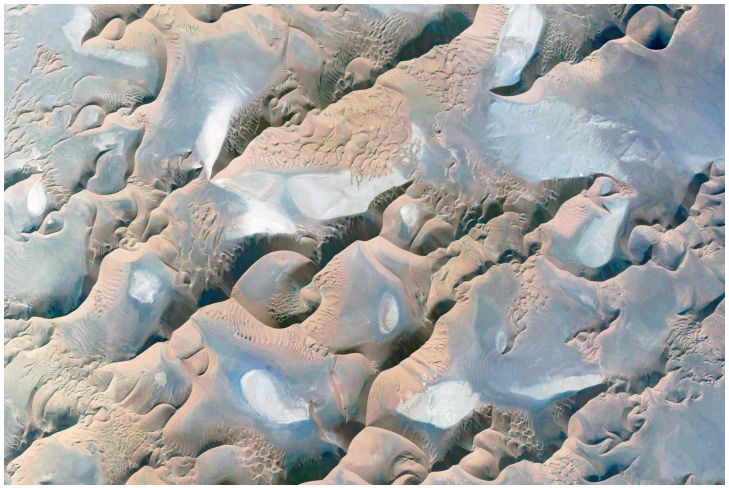
where P_1, P_2 and P_3 are the premises, and Q is the conclusion.

(There is no ambiguity in writing $P_1 \wedge P_2 \wedge P_3$ without brackets, because the connective $\wedge$ obeys the associative law.)

If the argument is valid, the expression should be a tautology.

You have to define a few propositional atoms to form P_1, P_2 and P_3.

PART II. Advanced Propositional Logic



Keys:

IIA. REMINDER on Logic Laws

IIB. Propositional Logic as a Formal Language

IIC. Soundness and Completeness of Propositional Logic

IID. Proof theory of Propositional Logic

1.2 PART II: Advanced Propositional Logic

1.2.1 REMINDER on Rules of Inference (Logic Laws)

Consider an implication of the form $\phi_1, \phi_2, \ldots, \phi_n \to \psi$

Valid arguments. This implication is understood as $\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n \to \psi$.

When all the premises $\phi_1, \phi_2, \ldots, \phi_n$ have true value T, and find that under these circumtances ψ also has the value T, then the implication

$$\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n \to \psi$$

is tautology, and we have a **valid argument**. Valid (correct) arguments

1. enable us to consider only the cases wherein all the premises $\phi_1, \phi_2, \ldots, \phi_n$ are true;
2. used in the development of step-by-step validation of how the conclusion ψ

   logically follows from the premises $\phi_1, \phi_2, \ldots, \phi_n$ in the implication of the form

$$\phi_1, \phi_2, \ldots, \phi_n \to \psi$$

A few well known valid arguments are summarized next.

I: Rule of Modus Ponens $p \wedge (p \to q) \to q$

Modus Ponens is also called the Rule of Detachment.

Quiz : Can you check $p \wedge (p \to q) \to q$ is a valid argument? In the tabular form:

p

$p \to q$

— ———

$\therefore \quad q$

where the $\therefore$ stands for '*therefore*', indicating that q is the conclusion of two premises p and $p \to q$ above the line.

♦ **EXAMPLE 1.5.** *Consider the following statements.*

1/ Barack Obama is a human being.

2/ If x is a human being then x will die.

3/ $\therefore$ Barack Obama will die.

II: Rule of Syllogism

$(p \to q) \wedge (q \to r) \to (p \to r)$ where p, q and r are any statement. In the tabular form, it is written

$p \to q$

$q \to r$

— ———

$\therefore \quad p \to r$

III: Rule of Modus Tollens, a method of denying.

In the tabular form:

$p \to q$

$\neg q$

— ———

$\therefore \quad \neg p$

Quiz: Use Modus Tollens to prove the follwing scheme

$p \to r$

$r \to s$

$t \vee \neg s$

$\neg t \vee u$ for any propositions p, r, s, t, u

$\neg u$

$\therefore \quad \neg p$

IV: Rule of Disjunctive Syllogism says $[(p \vee q) \wedge \neg p] \to q$.

This is derived from Modus Ponens and the logical equivalence $p \to q \Leftrightarrow \neg p \vee q$ or

$$p \vee q \Leftrightarrow \neg p \to q$$

In the tabular form:

$p \vee q$

$\neg p$

$\therefore \quad q$

This rule of inference arises when there are exactly two possibilities to consider, and we are able to elliminate one of them as being true, i.e. it is false. Then the other possibility must be true.

V: Rule of Contradiction

Denote F_0 to be a contradiction. You can prove that

$$(\neg p \to F_0) \to p$$

is a tautology! This called the Rule of Contradiction, written as:

$$\begin{array}{ll} & \neg p \to F_0 \\ \hline \therefore & p \end{array}$$

The Rule of Contradiction is a key method of establishing the validity of an argument, the method **Proof by Contradiction**. Moreover we can employ this to the case when we wish to prove

$$(p_1 \wedge p_2 \wedge \ldots \wedge p_n) \to q$$

then we can form the validity of the argument

$$(p_1 \wedge p_2 \wedge \ldots \wedge p_n) \wedge \neg q \to F_0 \, WHY?$$

1.2.2 Propositional Logic as a Formal Language

In natural deduction, we have a collection of laws of logic or *proof rules*. They allow us to infer *formulas* (i.e. logical expressions) from other simple or given formulas, called **premises**;

then eventually we may infer a **conclusion** from a set of premises.

Definition 1.5 (Natural deduction in propositional logic) ♣

This intention is symbolically denoted as

$$\phi_1, \phi_2, \ldots, \phi_n \text{ (premises)} \quad \vdash \quad \psi \text{ (conclusion)}$$

This expression is called a natural deduction or a **sequent**.

It is *valid* if a proof for it can be found.

Definition 1.6 (Well-formed formulas) ♣

The well-formed formulas of propositional logic are those which we obtain by using the construction rules below, and only those finitely many times:

- **Atom**: every propositional atom $p, q, r, \ldots$; and $p_1, p_2, \ldots$ is a well-formed formula.
- $\neg$: if ϕ is a well-formed formula then so is $(\neg\phi)$
- $\wedge$: if ϕ and ψ are well-formed formulas, then so is $(\phi \wedge \psi)$

- $\vee$: if ϕ and ψ are well-formed formulas, then so is $(\phi \vee \psi)$
- $\rightarrow$: if ϕ and ψ are well-formed formulas, then so is $(\phi \rightarrow \psi)$

We define a *propositional language* **L** is a set of propositional atoms $p, q, r, ...$

Let **L** be a propositional language. An **L**-assignment is a mapping

$$M : \{p | p \text{ is an atomic } \mathbf{L} - formula\} \rightarrow \{T, F\}.$$

Note that if **L** has exactly n atoms then there are exactly 2^n different **L**- assignments.

Lemma 1.1 ♡

Given an **L**-assignment M , there is a unique **L**-valuation

$$v_M : \{A | A \text{ is an } \mathbf{L} - formula\} \rightarrow \{T, F\}$$

Definition 1.7 ♣

A set of formulas S is said to be *satisfiable* if there exists an assignment M, and a corresponding **L**-valuation v_M which satisfy S, i.e., $v_M(S) = T.$

Definition 1.8 (Provable or Logical implication) ♣

Let ϕ and ψ be formulas of propositional logic. Build the implication $S = \phi \longrightarrow \psi$.

We say S is *provable implication*, denoted as $\phi \Rightarrow \psi$, **iff** $\phi \longrightarrow \psi$ is valid,

that means we can show a rigorous series of arguments (reasonings)

such that the implication is true, i.e.

$$\phi \to \psi \equiv \texttt{True}$$

Question: Given a provable implication,

$$S = \phi \to \psi,$$

is it satisfiable, that is there exists **L**-valuation v_M which satisfies S, i.e.,

$$v_M(S(p, q, ...)) = T \quad \text{for all } p, q, \in S.$$

ANSWER: not sure. To be answered in Section 1.4.4.

Definition 1.9 (Provable equivalence)

Let ϕ and ψ be formulas of propositional logic.

*We say ϕ and ψ are provable equivalent **iff** the sequents*

$$\phi \to \psi \text{ and } \psi \to \phi \text{ are valid .}$$

*In other words, [ϕ and ψ are provably equivalent **iff** $\phi \Leftrightarrow \psi$ (also write $\phi \equiv \psi$)].* ♣

1.2.3 Soundness and Completeness of Propositional Logic

Consider a provable implication

$$\phi_1, \phi_2, \ldots, \phi_n \text{ (premises)} \quad \Rightarrow \quad \psi \text{ (conclusion)}$$

Soundness of Propositional Logic means:

Every provable implication is satisfiable for all valuations!

We need to show: for all valuations of $p, q, \ldots$ in which

all propositions $\phi_1, \phi_2, \ldots, \phi_n$ evaluate to `True`, then ψ evaluates to `True`.

Definition 1.10 (Semantic entailment relation)

♣

If for all valuations in which all $\phi_1, \phi_2, \ldots, \phi_n$ evaluate to `True`,

formula ψ evaluates to `True` as well, we say that

$\phi_1, \phi_2, \ldots, \phi_n$ (premises) $\models$ ψ holds, and

call $\models$ the semantic entailment relation.

Theorem 1.1 (Soundness)

Let $\phi_1, \phi_2, \ldots, \phi_n$ and ψ be propositional logic formulas. If

$$\phi_1, \phi_2, \ldots, \phi_n \quad \Rightarrow \quad \psi$$

is valid, then the semantic entailment relation

$$\phi_1, \phi_2, \ldots, \phi_n \text{ (premises)} \quad \models \quad \psi$$

holds. ♡

How about the other way? That is, if we have

$$\phi_1, \phi_2, \ldots, \phi_n \text{ (premises)} \quad \models \quad \psi,$$

will we have the provable implication

$$\phi_1, \phi_2, \ldots, \phi_n \quad \Rightarrow \quad \psi?$$

Completeness of Propositional Logic

We show that the provable implications of Propositional Logic is complete, i.e: whenever

$$\phi_1, \phi_2, \ldots, \phi_n \text{ (premises)} \quad \models \quad \psi$$

holds, then there exists a provable implication

$$\phi_1, \phi_2, \ldots, \phi_n \quad \Rightarrow \quad \psi$$

being valid (i.e. you have to find a proof for the implication

$$\phi_1, \phi_2, \ldots, \phi_n \quad \rightarrow \quad \psi).$$

Soundness and Completeness of Propositional Logic

Theorem 1.2 (Soundness and Completeness)

Let $\phi_1, \phi_2, \ldots, \phi_n$ and ψ be formulas of propositional logic. Then

$$\phi_1, \phi_2, \ldots, \phi_n \quad \models \quad \psi$$

*holds **iff** the implication*

$$\phi_1, \phi_2, \ldots, \phi_n \quad \Rightarrow \quad \psi$$

is valid.

♡

1.2.4 Proof theory of Propositional Logic

AIM: we want to transform formulas into simpler ones

- which don't contain $\rightarrow$ at all,
- and the occurrences of $\wedge$ and $\vee$ are confined to separate layers

such that **validity checks** are easy!

Definition 1.11 (Conjunctive Normal Form of Propositional Logic Formulas)

♣

A literal L is either an atom p or the negation $\neg p$.

A propositional formula C is in Conjunctive Normal Form (CNF)

if it is a *conjunction* of clauses, where each clause D is a *disjunction* of literals.

In other words, **propositional formulas are recursively defined**, in Backus Naur form, as:

$$L ::= p \,|\, \neg p$$

$$D ::= L \,|\, L \vee D$$

$$C ::= D \,|\, D \wedge C.$$

Example

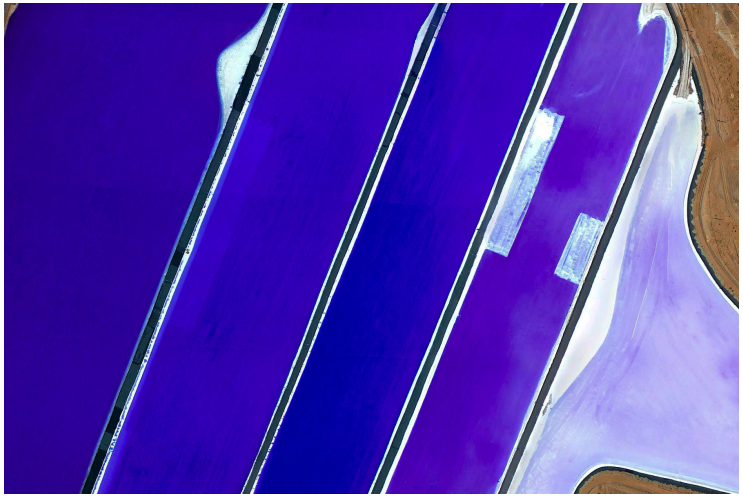
$$(\neg q \vee p \vee r) \wedge (\neg p \vee r) \wedge q$$

is a CNF, but $(\neg(q \vee p) \vee r) \wedge (q \vee r)$ is not! WHY?

Lemma 1.2

A disjunction of literals $L_1 \vee L_2 \ldots L_m$ is valid iff there are $1 \leq i, j \leq m$ such that L_i is $\neg L_j$. ♡

PART III: Predicate Logic- A Primer



1.3 Predicate Logic- A Primer

We discuss the followings in PART III:

1/ Fundamentals of Predicate logic

2/ Natural deduction calculus in Predicate Logic-

Predicate vocabulary

3/ Parse tree of a logic formula

4/ Free and bound variables, in part 1.3.4.

Warming up

Propositional logic dealt quite well and satisfactorily with sentence components like

- **not**
- **and**
- **or**, and
- **if ... then**

but the logical aspects of natural and artificial languages are richer than that!

We have seen *modifiers* like

there exist ...,

all ... ,

among ... and **only** ...

and now develop a theory for using them.

To write predicates and quantifiers symbolically,

we will use capital letters to denote predicates.

1.3.1 Fundamentals of Predicate logic

A/ Predicate: one or many variables

- A predicate P that contains a variable x can be written symbolically as $P(x)$.
- A predicate can contain more than one variable;

  a predicate P with two variables, x and y for example, can be written $P(x, y)$.

  In general, a predicate with n variables, $x_1, x_2, \ldots, x_n$, can be written $P(x_1, x_2, \ldots, x_n)$.

There are several ways to express the notions of **always true** and **sometimes true** in English.

- The expressions '**for all**' $\forall$ and '**there exists**' $\exists$ are called *quantifiers*.

- The process of applying a quantifier to a variable is called *quantifying* the variable.

(**L1**) 'For all n, $P(n)$ is true' can be reduced to $\forall n,\ P(n)$

(**L2**) 'There exists an n such that $P(n)$ is true' can be reduced to $\exists n,\ P(n)$

Always True we use symbol $\forall$	Example
(L1) 'For all n, $P(n)$ is true'.	"For all x, $x^2 \geq 0$.
$\forall\, n$, $P(n)$ is true	
or $P(n)$ is true for every n	$x^2 \geq 0$ for every x
Sometimes True we use symbol $\exists$	
(L2) 'There exists an n such that	There exists an x
$P(n)$ is true'.	such that $5x^2 - 7 = 0$
$\exists n$: $P(n)$ is true	

The above tables give some general formats on the left

and few examples using those formats on the right.

♦ **EXAMPLE 1.6.** *Let $P(x)$ be the statement "$x < 2$".*

I) What is the truth value of the quantification $\forall x P(x)$, where the domain consists of all real number?

- $P(3) = 3 < 2$ is false
- $\Rightarrow \forall x P(x)$ is false
- 3 is a `counterexample` of $\forall x P(x)$

II) What is the truth value of the quantification $\exists x P(x)$, where the domain $D = \mathbb{R}$? ■

A/ Predicate: one or many variables ↙

B/ Quantifying a Predicate: many propositions

For two predicates (i.e. open statements) $p(x), q(x)$ defined for a prescribed universe U, consider the universally quantified statement

$$\forall x \in U\,[\,p(x) \to q(x)\,] \text{ or shortly } \quad \forall x\,[\,p(x) \to q(x)\,].$$

Definition 1.12

If the implication $p(a) \to q(a)$ is true for each a in the universe U then we say that $p(x)$ ***logically implies*** *$q(x)$, and can write*

$$\forall x\,[\,p(x) \to q(x)\,].$$

[a is a value, x is the variable of predicate.]

We similarly define the logically equivalent statement

$$\forall x \in U\,[\,p(x) \Leftrightarrow q(x)\,].$$

♣

♦ **EXAMPLE 1.7.** *Express the statement "Some student in this class comes from Central Vietnam."*

Solutions

- 1/ $M(x)$ = x comes from Central Vietnam

  Domain for x is `the students in the class` . The answer is $\exists x M(x)$
- 2/ Domain for x is `all people ...`

♦ **EXAMPLE 1.8.** *Consider a declarative sentence: Every student is younger than some instructor.*

This is complex, and tell us many things, about:

1. being a student
2. being an instructor, and

3. being younger than somebody else.

Need a mechanism to express them, plus their logical dependence.

Introduce predicates

- $S(x) : x$ is a student
- $I(x) : x$ is an instructor
- (Two-variable) predicate $Y(x, y) :\ x$ is younger than y

We need two quantifiers $\forall$ and $\exists$ to write the above example entirely symbolic way

$$\forall x, [S(x) \to (\exists y : I(y) \wedge Y(x, y))]$$

or $\forall x, [S(x) \to (\exists y : (I(y) \wedge Y(x, y)))\,]$ **? Rephrase this back to daily used sentence**? ■

PRACTICE 1.2.

A) Consider a declarative sentence: Every child is younger than its mother.

Write this example in an entire symbolic way.

B) Write the declarative sentence below in an entire symbolic way:

Andy and Paul have the same maternal grandmother.

Fact 1.1 (Logic equivalences and implications for quantified statements in one variable).

For a prescribed universe and any open statements $p(x), q(x)$, we have

- $\exists x\,[p(x) \wedge q(x)] \Rightarrow [\exists x\, p(x) \wedge \exists x\, q(x)]$

  (here only one way $\Longrightarrow$, the x must be the same).
- $\exists x\,[p(x) \vee q(x)] \Leftrightarrow [\exists x\, p(x) \vee \exists x\, q(x)]$
- $\forall x\,[p(x) \wedge q(x)] \Leftrightarrow [\forall x\, p(x) \wedge \forall x\, q(x)]$ (here the x must be taken in the same domain).

C/ Two useful binary predicates We often use two useful binary predicates below.

- **Equality** $=$: a special predicate, a binary one but usually written in between of its arguments:

$$x = y \quad \text{instead of } = (x, y)$$

- **The function form/symbol** $m(x)$ of a binary predicate $M(x, y)$ allows us to represent the semantic of a sentence elucidatedly.

E.g., for $M(x, y)$: 'x is the mother of y' we write $m(y) \equiv y$'s mother. See EXAMPLE 1.10.

D/ Predicate logic- Quantifiers with many variables

Definition 1.13

*A variable which has been quantified is said to be **bound**.*

*A variable that appears in a predicate but is not bound is said to be **free**.* ♣

Mixing quantifiers. Many mathematical statements involve several quantifiers.

♦ **EXAMPLE 1.9.** ***Goldbach's Conjecture*** *states that*

```
Every even integer greater than 2 is the sum of two primes.
```

Write this more verbosely to make the use of quantification clearer.

ANSWER: $\forall n, \quad [n \in \mathbb{Z} \wedge ?] \longrightarrow [??]$

♦ **EXAMPLE 1.10.** *Let's look at an English sentence '**Every girl is younger than her mother**'.*

Using three predicates

1/ $G(x)$**:** x is a girl;

2/ $M(x, y)$**:** y is x's mother; and the third

3/ $Y(x,y)$**:** x is younger than y

gives us the complete statement $\forall x \forall y\, [G(x) \wedge M(x,y) \rightarrow Y(x,y)]$.

Note that y is only introduced to denote the mother of x. If everyone has exactly one mother, the 2-variable predicate $M(x,y)$ is a function, when reading from right to left.

A function symbol $m(x) \equiv$ "`the mom of` x " then can be applied [in place of $M(x,y)$] to give a better answer

$$\forall x\, [G(x) \longrightarrow Y(x, m(x))] \blacksquare$$

E/ Order of quantifiers is important

Swapping the order of different kinds of quantifiers (either *existential* or *universal*) changes the meaning of a proposition.

♦ **EXAMPLE 1.11.** *Let's return to a confusing statement:* ***Every American has a dream****.*

This sentence is ambiguous because the order of quantifiers is unclear. WHY?

Let A be the set of Americans, let D be the set of dreams, and define the predicate

$$H(a,d) \text{ to be "American } a \text{ has dream } d.\text{"}.$$

Now the sentence could mean either

(I) there is a single dream that every American shares:

$$\exists d \in D : \exists a \in A :\ H(a,d)$$

or (II) every American has an individual dream: $\forall a \in A, \exists d \in D :\ H(a,d).$

PRACTICE 1.3. *(15 min)*

In the specification of a system for booking theatre seats, $B(p, s)$ denotes the predicate "**person *p* has booked seat *s***."

Write the following sentences in symbolic form:

1. Seat *s* has been booked.
2. Person *p* has booked a (that is, at least one) seat.
3. All the seats are booked.
4. No seat is booked by more than one person.

D/ Predicate logic- Quantifiers with many variables

E/ Order of quantifiers is important ↙

F/ Negating a predicate with many variables and quantifiers

Applying **not** to a proposition/predicate is called *negating the proposition/predicate*.

Suppose we want to apply the connective not to the following proposition:

'*All swans are black*.'

The original proposition can be written in symbols: $\forall x, \quad P(x)$

where $P(x)$ is the predicate '*Swan x is black*'.

- Here is one way of forming the negation: '*It is not true that all swans are black*.'

- Or, more simply: '*Not all swans are black*.' $\equiv$ '*There is a swan that is not black*.'

`Rule for negating a predicate`

We **negate the predicate and changing all its quantifiers**.

Statement	Negation	Equivalent form
$\forall x P(x)$	$\neg(\forall x P(x))$	$\exists x \neg P(x)$
$\exists x P(x)$	$\neg(\exists x P(x))$	$\forall x \neg P(x)$

♦ **EXAMPLE 1.12.**

- S = `All` CSE students study Discrete Math 1
- Let C(x) denote "x is a CSE student"
- Let S(x) denote "x studies Discrete Math 1"
- Then S is symbolized as $\forall x : C(x) \rightarrow S(x)$
- Hence, $\neg$ S = $\exists x : \neg(C(x) \rightarrow S(x)) \equiv \exists x : C(x) \wedge \neg S(x)$

  means `There is` a CSE student who does not study Discrete Math 1??

  W

hat is the rule for negating a predicate?

The general principle for negating a predicate is that moving a **not** across a quantifier **changes the kind of quantifier**.

PRACTICE 1.4. *(5 min)*

Now write down the negation of the following proposition:

'For every number x there is a number y such that $y < x$.'

For one variable but compound case:

$$\text{Rule } 4: \ \neg\,[\,\forall x\,(P(x) \rightarrow Q(x))\,] \equiv \exists x\,[\,P(x) \wedge \neg Q(x)\,]$$

WHY? We know the law $[p \rightarrow q \equiv \neg p \vee q]$,

and when negating a complex proposition, we always

- go from the left to the right, apply from inner to outer, and

- negate component by component.

Summary of Quantifier Equivalences

Rule 1: $\neg\,[\,\forall x\,P(x)\,]\equiv\exists x\,[\,\neg P(x)\,]$

Rule 2: $\neg\,[\,\exists x\,P(x)\,]\equiv\forall x\,[\,\neg P(x)\,]$

Rule 3: $\neg\,[\,\forall x\,\exists y\,P(x,y)\,]\equiv\exists x\,\forall y\,[\,\neg P(x,y)\,]$

Rule 4: $\neg\,[\,\forall x\,(P(x)\rightarrow Q(x))\,]\equiv\exists x\,[\,P(x)\wedge\neg Q(x)\,]$

HOMEWORK: Try problems below at home within 120 minutes.

1. Translate these:
   - All lions are fierce.
   - Some lions do not drink coffee.
   - Some fierce creatures do not drink coffee.

   HINT: Let $P(x)$, $Q(x)$ and $R(x)$ be the statements "x is a lion", "x is fierce" and "x drinks coffee", respectively.

2. Translating Nested Quantifiers

$$\forall x\ (C(x)\vee\exists\mathtt{y}\ (\mathtt{C(y)}\wedge\mathtt{F(x,y)})\ )$$

provided that:

- $C(x)$: x has a computer,
- $F(x, y)$: x and y are friends,
- $x, y \in$ all students in your school.

3. For each of the logical formulas, indicate whether or not it is true when the domain of discourse is $\mathbb{N}$ (the natural numbers 0, 1, 2, . . .), $\mathbb{Z}$ (the integers), $\mathbb{Q}$ (the rationals), $\mathbb{R}$ (the real numbers), and $\mathbb{C}$ (the complex numbers).

$$\exists x\,(x^2 = 2)$$

$$\forall x\,\exists y(x^2 = y)$$

$$\forall y\,\exists x(x^2 = y)$$

$$\forall x \neq 0\,\exists y(xy = 1)$$

$$\exists x\,\exists y\ : (x + 2y = 2) \wedge (2x + 4y = 5)$$

4. (Order of quantifiers is important).

   Let $p(x, y)$ be an open statement '$x + y = 25$' where $x, y \in \mathbb{Z}$. Are the statements

   $\forall x\,\exists y\,p(x, y)$ and $\exists y\,\forall x\,p(x, y)$ logically equivalents?

5. Translating into Logical Expressions: **Every people has one best friend**.

6. Inference

   - If I have a girlfriend, I will take her to go shopping.
   - Whenever I and my girlfriend go shopping and that day is a special day, I will surely buy her some expensive gift.
   - If I buy my girlfriend expensive gifts, I will eat noodles for a week.

- Today is March 8.
- March 8 is such a special day.
- Therefore, if I have a girlfriend,...
- Your conclusion is: ??

1.3.2 Natural deduction calculus in Predicate Logic

Two things involved in a predicate logic formula:

- **terms** or **objects** include individuals [2] and variables x, v ...
- **formulas**: $Y(x, m(x))$ is a formula, though x and $m(x)$ are terms [See from EXAMPLE 1.10].

Definition 1.14

*A **predicate vocabulary** consists of 3 sets:*

a) a set of predicate symbols $\mathcal{P}$

b) a set of function symbols $\mathcal{F}$

c) a set of constant symbols $\mathcal{C}$.

** **Arity** of a function* $f(...) \in \mathcal{F}$ *is the number of arguments it expects.*

♣

We view constants are *nullary* functions (don't take any argument), so can view $\mathcal{C} \subset \mathcal{F}$.

Terms and Their Backus Naur forms

Definition 1.15

***Terms** are recursively defined as follows.*

♣

- Any variable x is a term
- If $c \in \mathcal{F}$ is a *nullary* function [having arity 0] then c is a term
- If $t_1, t_2, \ldots, t_n$ are terms and $f \in \mathcal{F}$ has arity $n > 0$, then $f(t_1, t_2, \ldots, t_n)$ is a term.

[2] such as a= Andy, p= Paul ...

- Nothing else is a term.

In Backus Naur form we may write

$$t ::= x \,|\, c \,|\, f(t, \ldots, t)$$

where $x \in$ **Var** a set of variables, $c \in \mathcal{F}$ a *nullary*, and $f \in \mathcal{F}$ with arity $n > 0$.

The choice of sets $\mathcal{P}$ and $\mathcal{F}$ is driven by what we intend to describe.

♦ **EXAMPLE 1.13.**

- Are $g(f(n), n)$ and $f(g(n, f(n)))$ terms? Why?
- Are $g(n)$, $f(f(n), n)$ terms? Explain.

♦ **EXAMPLE 1.14** (Database as a logic model)**.**

We may define a 4-tuples $\mathcal{P} = \{M, F, S, D\}$ referring to

- *M*: being male; *F*: being female
- *S*: being a son of ...
- *D*: being a daughter of ...

if we work on **database** representing relations between human being. Here,

F, M are *unary* predicates (they take one argument), but

S and *D* are *binary* predicates (taking twos). We utilize this instance in Example 1.21. ■

Denoted respectively sets $\mathcal{P}$ be *predicate* symbols, and $\mathcal{F}$ be *function* symbols.

Use the defined set of terms over $\mathcal{F}$ we recursively define the **set of formulas** over $(\mathcal{F}, \mathcal{P})$.

Definition 1.16 (The formulas of predicate logic)

If $P \in \mathcal{P}$ is a predicate symbol of arity $n \geq 1$, and

$t_1, t_2, \ldots, t_n$ are terms over $\mathcal{F}$, then $P(t_1, t_2, \ldots, t_n)$ is a formula. ♣

Furthermore,

- If ϕ is a formula then so is $(\neg\phi)$.
- If ϕ and ψ are formulas, then so are $(\phi \wedge \psi)$, $(\phi \vee \psi)$ and $(\phi \to \psi)$.
- If ϕ is a formula and x is a variable, then $(\forall x\, \phi)$ and $(\exists x\, \phi)$ are formulas.
- Nothing else is a formula.

We summarize the definition with the above components in the following recursion

$$\begin{aligned} \phi \;:=\; & P(t_1, t_2, \ldots, t_n) \mid \\ & (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \to \phi) \mid \\ & (\forall x \phi) \mid (\exists x \phi) \end{aligned}$$

What have been done

- 4.3.1 Fundamentals of Predicate logic
- 4.3.2 Natural deduction calculus in Predicate Logic

- **4.3.3 Parse tree of a logic formula?**

- 4.3.4 Free and bound variables

To construct the Parse tree of a logic formula we follow the rule

Leaves are atomic propositions, and

internal nodes are operations $\neg, \wedge, \vee, \rightarrow$.

1.3.3 Parse tree of a logic formula

RULES:

1. **Leaves** are atomic propositions, and **internal nodes** are operations $\neg, \wedge, \vee, \rightarrow$.
2. We employ the **binding priorities** for logic connectives:

first a/ $\neg$; second b/ the $\vee$ and $\wedge$; finally c/ the $\rightarrow$.

Parse tree of a propositional logic formula

Example, draw the parse tree of a propositional logic formula

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

Parse tree of a predicate logic formula

Convention (for predicate logic): Just like for propositional logic, we introduce convenient conventions to reduce the number of parentheses:

- $\neg, \forall x$ and $\exists x$ bind most tightly;
- then $\wedge$ and $\vee$;
- then $\rightarrow$, which is right-associative.

PRACTICE 1.5.

Draw the parse tree of a predicate logic formula

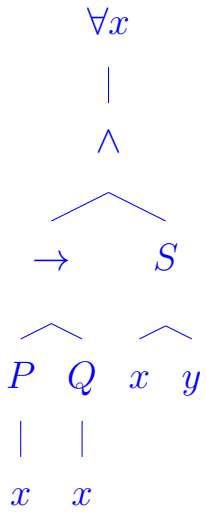
$$\forall x[(P(x) \rightarrow Q(x)) \wedge S(x, y)]$$

HINT: Just remember that:

1. the quantifiers $\forall x$ and $\exists y$ form nodes and have, like negation, just one subtree;

2. predicate expressions, which are generally of the form $P(t_1, t_2, \ldots, t_n)$, [Definition 1.16]
   have the symbol P as a node, but now P has n many subtrees,
   i.e. the parse trees of the terms $t_1, t_2, \ldots, t_n$.

ANSWER



PRACTICE 1.6.

Let $n, f, g \in \mathcal{F}$ be function symbols, respectively nullary, unary and binary.

- Are $g(f(n), n)$ and $f(g(n, f(n)))$ terms? Explain
- Are $g(n)$, $f(f(n), n)$ terms?

- (*) Define a set of function symbols

  $\mathcal{F} = \{0, 1, 2, ...\}(nullary) \cup \{s\}(unary) \cup \{+, -, *\}(binary)$.

  Is $t = *(-(2, +(s(x), y)), x)$ a term? Draw the parse tree of t.

  Rewrite t using infix notation.

EXTRA

1. (About the role of domain-specific knowledge) Translate the sentence

   Every son of my father is my brother

   into predicate logic by two different ways.

2. HINT: we need two predicates

   $S(x, y) : x$ is a son of y; $B(x, y) : x$ is a brother of y.

   and two functions:

   m : constant for “me”; $f(x)$: father of x. ■

Predicate Logic- What for?

Remind that *variables* and *quantifiers* (introduced in **Section** 1.3.1 - Predicate logic (Reminder)) allow us to express the notion

all ... and *some* ...

QUESTION: But how to verify that the claim

$$\forall x\, Q(x)$$

is true?

Method 1: replacing x by any of its possible values and checking that Q holds for each one of them [very inefficient!]

Method 2: Use proof rules (see Section 1.4.1 Proof Theory of Predicate Logic)

Need the concepts of Free and Bound variables to solve these!

1.3.4 Free and bound variables

♣ **OBSERVATION 1.** *Variables in a formula occur at two different sorts of places in parse tree:*

Type 1: they appear next to quantifiers $\forall$ and $\exists$: such nodes always have one subtree, determining their scope to which the respective quantifier applies.

Type 2: they form leaf nodes: variables stand for values that still have to be made concrete. Observe any leaf node, there are two key occurrences:

■ **CONCEPT 1** (Informal description)**.**

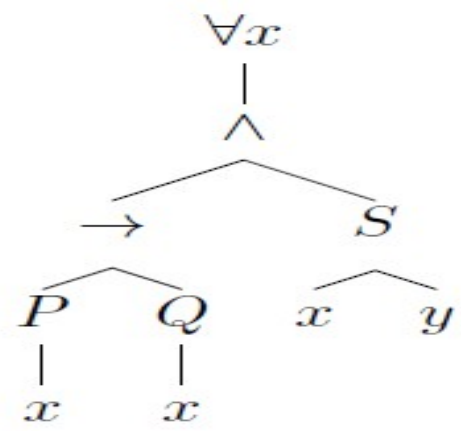
a/ *Bound case*: when walk up the tree from a node x, we run into the quantifier $\forall$:

these nodes represent any value of x.

b/ *Free case*: when walking upwards from a node y, we run into the node $\forall x$,

that x is **not relevant** to y, so y is free from x.

♦ **EXAMPLE 1.15.**



A simple parse tree

a) Does the figure show the parse tree T_1 of the formula

$\forall x[(P(x) \to Q(x)) \wedge S(x, y)]$?

b) Recognize free and bound variables.

c) What is the relationship between variable 'binder' x and occurrences of x? ■

Definition 1.17

Let ϕ be a formula in predicate logic. ♣

* Any occurrence of x in ϕ is **free** in formula ϕ

if it is a leaf node in the parse tree of ϕ

such that there is no path upwards from node x to node $\forall x$ or $\exists x$.

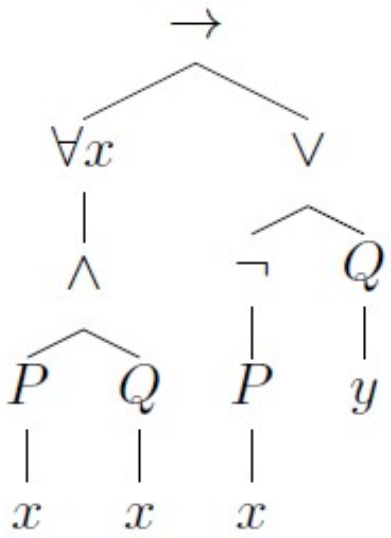
* Otherwise, that occurrence of x is called **bound**.

Hence, if x occurs in formula ϕ, then

- it is bound if, and only if it is in the scope of some nodes $\exists x$ or $\forall x$;

- otherwise it is free.

♦ **EXAMPLE 1.16.** *From the figure could you recover a predicate formula?*



The figure shows the parse tree T_2 of the formula

$$[\forall x(P(x) \wedge Q(x))] \rightarrow [\neg P(x) \vee Q(y)]$$

Which variable occurrences are free; which are bound? ■

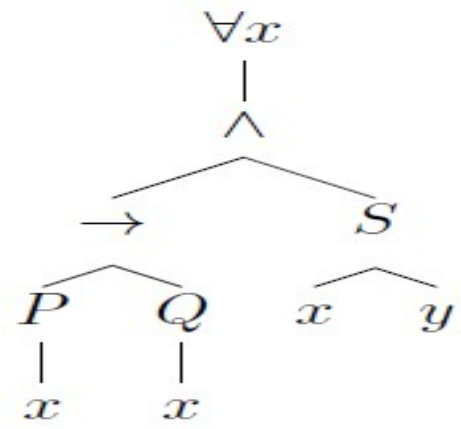
Definition 1.18 (Substitution) ♣

Given a variable x, a term t and a formula ϕ, we define $\phi[t\,/\,x]$ to be

the **formula obtained by replacing each free occurrence of variable** x in ϕ with t.

PRACTICE 1.7.

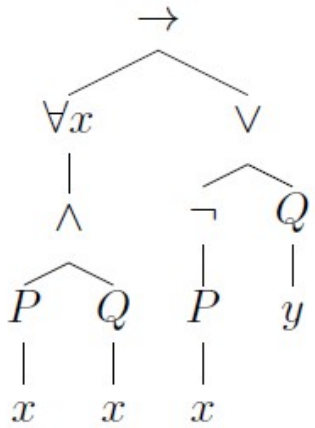
A) Could you substitute occurrences of variable x in the tree T_1 by a term $f(x, y)$?



A simple parse tree

NOTE: Instead of using $[x \Rightarrow t]\phi$ few textbooks also use the notation $\phi[t/x]$

(we find the order of arguments in the latter notation hard to remember).

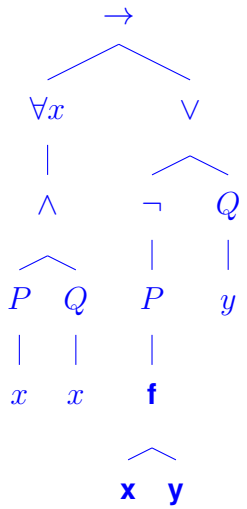


B) Could you substitute occurrences of variable x in the above parse tree T_2 of the formula

$[\forall x(P(x) \wedge Q(x))] \rightarrow [\neg P(x) \vee Q(y)]$ by a term $f(x, y)$?

ANSWER $[x \Rightarrow f(x, y)] \quad ((\forall x(P(x) \wedge Q(x))) \rightarrow (\neg P(x) \vee Q(y)))$

$$= (\forall x(P(x) \wedge Q(x))) \rightarrow (\neg P(f(x, y)) \vee Q(y))$$



When substitution allowed?

To avoid changing action that gives wrong specification of other variable $y \neq x$ in parse trees, we can replace some term t into x only the term t satisfying the following.

Definition 1.19 (Freeness and boundness of a variable)

Given a term t, a variable x and a formula ϕ in predicate logic. We say that:

1. Term t is free for x if no free x leaf in ϕ occurs in the scope of node $\forall y$ or $\exists y$ for any variable y occurring in t.
2. Formula ϕ is free for x if x is a leaf node in the parse tree of ϕ such that there is **no path upwards** from that node x to a node $\forall x$ or $\exists x$.
   Otherwise, that occurrence of x is called **bound** (in ϕ).

3. In terms of parse trees, **the scope of a quantifier is just its subtree**, minus any subtrees which re-introduce a quantifier for x.

Thus, Item 2 equivalently says: if variable x occurs in ϕ, then

it is **bound if, and only if**, it is in the scope of some $\exists x$ or some $\forall x$;

otherwise it is free.

Free-ness as precondition:

To compute $[x \Rightarrow t]\phi$, we demand that t is free for x in ϕ.

What if not? Rename the bound variable!

`Example of Renaming`:

$[x \Rightarrow f(y,y)](S(x) \wedge \forall y(P(x) \rightarrow Q(y)))$

$$\hookrightarrow\ [x \Rightarrow f(y,y)](S(x) \wedge \forall z(P(x) \rightarrow Q(z)))$$

$$\hookrightarrow\ S(f(y,y)) \wedge \forall z(P(f(y,y)) \rightarrow Q(z))$$

NOTE: The logic equivalence notation $\equiv$ in propositional logic

can be replaced by the two-way sequent $\dashv\vdash$. For example,

$\neg\,[\,\forall x\, P(x)\,] \dashv\vdash \exists x\,[\,\neg P(x)\,]$ means $\neg\,[\,\forall x\, P(x)\,] \equiv \exists x\,[\,\neg P(x)\,]$

Fact 1.2. *We so can employ the followings, with any predicate (formula) ϕ and ψ.*

$$\neg\forall x\phi \quad \dashv\vdash \quad \exists x\neg\phi$$
$$\neg\exists x\phi \quad \dashv\vdash \quad \forall x\neg\phi$$
$$\exists x\exists y\phi \quad \dashv\vdash \quad \exists y\exists x\phi$$

Assume further x is **not free** in ψ, the followings are applied:

$$\forall x\phi \wedge \psi \quad \dashv\vdash \quad \forall x(\phi \wedge \psi)$$
$$\exists x(\psi \to \phi) \quad \dashv\vdash \quad \psi \to \exists x\phi$$

BASIC LOGIC THEORY: Reviewed Homework

A) Summarized terms [REF. 4: Chapter 2c]

Write down the explanations (in Vietnamese, or in English if possible) of the following terms, find examples for each term, what are the differences between them:

1. fallacy, contradiction, paradox, counterexample;
2. premise, assumption, axiom, hypothesis, conjecture;
3. tautology, valid, contradiction, satisfiable; soundness, completeness;

4. sequent, consequence, implication, (semantic) entailment;
5. argument, variable, arity.

B) Summarized notation [REF. 4: Chapter 2c]

- What are the differences between the following notations: '$\longrightarrow$', '$\Longrightarrow$', '$\vdash$', '$\models$'?
- What are the differences between the following notations: '$\longleftrightarrow$', '$\Longleftrightarrow$', '$\dashv\vdash$', '$\equiv$', '$=$'?
- Find examples to illustrate these differences.

C) Try the following problem within 1 hour at home.

An **adequate set** of connectives for propositional logic P is a set S such that for every formula of P there is an equivalent formula with only connectives from S.

For example, the set $S_0 = \{\neg, \vee\}$ is adequate for propositional logic.

1. Explain why we conclude that $S_0 = \{\neg, \vee\}$ is adequate for propositional logic.
2. Show that $S_1 = \{\neg, \wedge\}$, $S_2 = \{\neg, \rightarrow\}$ are adequate sets of connectives for propositional logic. Is $\{\leftrightarrow, \neg\}$ adequate? Prove your answer.

THE END of BASIC LOGIC THEORY

PART IV: Advanced Predicate Logic

1.4 Advanced Predicate Logic

The Essence

Syntax: We formalized in Section 1.3 the language of predicate logic, including substitution.

Proof theory: We next extend natural deduction from propositional to predicate logic

Semantics: We then describe models in which predicates, functions, and formulas have meaning.

Further topics: Soundness/completeness

The remaining parts of Section 1.4 include:

Proof Theory of Predicate Logic

The concept of model

Satisfaction Relation

Soundness and Completeness

(See extra info in REF. 4: Lecture DM2- Chapter 2e, 2f.)

1.4.1 *Proof Theory of Predicate Logic*

We discuss

- Natural Deduction Rules
- Quantifier Equivalences

Relationship between propositional and predicate logic:

1. If consider propositions as nullary predicates, then propositional logic is a sub-language of predicate logic. Hence, by `inheriting natural deduction` we can translate the rules for natural deduction in propositional logic directly to predicate logic.
2. The logic implication notation $\Longrightarrow$ in propositional logic can be replaced by the (one-way) sequent $\vdash$ in predicate logic.

Natural Deduction Rules

Most used proof rules in predicate logic are the following.

1. Built-in Rules for Equality : Identical rule and Equality rule
2. Universal quantification
3. Existential Quantification

Explicitly we discuss one by one.

1. Built-in Rules

   with **Identical rule** $= i$, formally written

$$\frac{}{t = t} = i$$

   or more general, **Equality rule** $= e$

$$\frac{t_1 = t_2 \qquad \phi[t_1/x]}{\phi[t_2/x]} = e$$

♦ **EXAMPLE 1.17** (Identical rule and Equality rule)**.**

We show:

$$f(x) = g(x) \vdash h(g(x)) = h(f(x))$$

using

$$\frac{}{t = t}[= i] \quad \text{and} \quad \frac{t_1 = t_2 \qquad [x \Rightarrow t_1]\phi}{[x \Rightarrow t_2]\phi}[= e]$$

1	$f(x) = g(x)$	premise
2	$h(f(x)) = h(f(x))$	$= i$
3	$h(g(x)) = h(f(x))$	$= e\ 1, 2$

1. **Built-in Rules** $\surd$
2. **Universal quantification**

   2a/ Equality case $\forall x\ e$

$$\frac{\forall x\ \phi}{\phi[t/x]}\forall x\ e \iff \frac{\forall x \phi}{[x \Rightarrow t]\phi}[\forall x\ e]$$

In formula ϕ we may replace $\phi[t/x]$ by $[x \Rightarrow t]\phi$, and think of the term t as a more concrete *instance* of x, where x is general term.

♦ **EXAMPLE 1.18.** *for Universal quantification- 2a/ Equality case*

Reuse notion of $M, F \ldots$ in Example 1.14, with the unary function $m(x)$: mother of x, we then prove the logic claim

$$\Big[F(m(\textit{Duong})), \forall x(F(x) \to \neg M(x))\Big] \vdash \neg M(m(\textit{Duong}))$$

1	$F(m(\textit{Duong}))$	premise
2	$\forall x(F(x) \to \neg M(x))$	premise
3	$F(m(\textit{Duong})) \to \neg M(m(\textit{Duong}))$	$\forall x\ e$ 2
4	$\neg M(m(\textit{Duong}))$	$\to e$ 3,1

2b/ Identical case : $\forall x\ i$

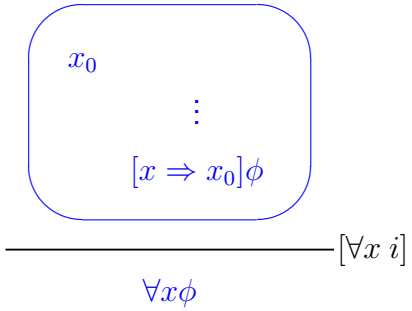
If we manage to establish a formula ϕ about a fresh variable x_0,

we can get the claim $\forall x \quad \phi$.

$$\frac{\begin{array}{c}\boxed{\begin{array}{c}x_0 \\ \vdots \\ [x \Rightarrow x_0]\phi\end{array}} \\ \hline \forall x\phi\end{array}}{}[\forall x\ i]$$

♦ **EXAMPLE 1.19.** *for Universal quantification- 2b/ Identical case*

We prove $\left[\forall x(P(x) \to Q(x)), \forall x P(x)\right] \vdash \forall x Q(x)$ via the logic diagram



1		$\forall x(P(x) \to Q(x))$	premise
2		$\forall x P(x)$	premise
3	x_0	$P(x_0) \to Q(x_0)$	$\forall x\ e$ 1
4		$P(x_0)$	$\forall x\ e$ 2
5		$Q(x_0)$	$\to e$ 3,4
6		$\forall x\ Q(x)$	$\forall x\ i$ 3–5

1. **Built-in Rules** $\surd$
2. **Universal quantification** $\surd$
3. Existential Quantification: two rules $\exists x\ i$ and $\exists\ e$:

$$\frac{[x \Rightarrow t]\phi}{\exists x\ \phi}[\exists x\ i]$$

$$\frac{\exists x\ \phi \qquad \boxed{\begin{array}{cc} x_0 & [x \Rightarrow x_0]\phi \\ & \vdots \\ & \chi \end{array}}}{\chi}[\exists\ e]$$

where χ is a logical conclusion.

♦ **EXAMPLE 1.20.** *for Existential Quantification*

$$\Big[\forall x(P(x) \to Q(x)), \exists x P(x)\Big] \vdash \exists x\ Q(x)$$

1		$\forall x(P(x) \to Q(x))$	premise
2		$\exists x P(x)$	premise
3	x_0	$P(x_0)$	assumption
4		$P(x_0) \to Q(x_0)$	$\forall x\ e$ 1
5		$Q(x_0)$	$\to e$ 4,3
6		$\exists x Q(x)$	$\exists x\ i$ 5
7		$\exists x Q(x) \equiv \chi$	$\exists x\ e$ 2,3–6

Definition 1.20

Fix formulas ϕ_1, ϕ_2 of predicate logic.

A sequent $\phi_1 \vdash \phi_2$ means a provable implication $\phi_1 \to \phi_2$,

in oher words, $\phi_1 \to \phi_2$ is valid (a proof for it can be found).

♣

Fact 1.3.

1/ Two notations $\phi_1 \vdash \phi_2$ and $\phi_1 \Rightarrow \phi_2$ are (equivalently) the same.

2/ We write $\phi_1 \dashv\vdash \phi_2$ (an equivalence of two predicate logic formulas) as

an abbreviation of $\phi_1 \vdash \phi_2$ and $\phi_2 \vdash \phi_1$.

PRACTICE 1.8. *How could we represent the sentence:* **Not all birds can fly**.

Step 1: Represent the claim $p =$ `All birds can fly` by predicate logic

Step 2: Apply **negation** operator to p to get $\neg\, p$. ■

HINT: we could exploit the next theorem.

Quantifier Equivalence

Theorem 1.3

Let ϕ and ψ be formulas of predicate logic. Then we have the following equivalences: ♡

1. $$\neg\forall x\, \phi \;\dashv\vdash\; \exists x\, \neg\phi$$

2. $$\neg\exists x\, \phi \;\dashv\vdash\; \forall x\, \neg\phi$$

3. $$a/ \quad \forall x\, \phi \wedge \forall x\, \psi \;\dashv\vdash\; \forall x\, (\phi \wedge \psi)$$

   $$b/ \quad \exists x\, \phi \vee \exists x\, \psi \;\dashv\vdash\; \exists x\, (\phi \vee \psi)$$

4. Assuming that x is not free in ψ, then $\forall x\, \phi \wedge \psi \;\dashv\vdash\; \forall x\, (\phi \wedge \psi)$.

   [Here we see $\forall x\, \phi \wedge \psi = (\forall x\, \phi) \wedge \psi$.]

We next discuss **semantics of predicate logic** via the concept of model.

1.4.2 *Model*

Definition 1.21 (Model)

*Let $\mathcal{F}$ contain **function** symbols and $\mathcal{P}$ contain **predicate** symbols.*

A model $\mathcal{M}$ for $(\mathcal{F}, \mathcal{P})$ consists of:

1. *A non-empty set A, the* universe*;*
2. *for each nullary function symbol $f \in \mathcal{F}$ a concrete element $f^{\mathcal{M}} \in A$;*
3. *for each $f \in \mathcal{F}$ with arity $n > 0$, a concrete function $f^{\mathcal{M}} : A^n \to A$;*
4. *for each $P \in \mathcal{P}$ with arity $n > 0$, a set $P^{\mathcal{M}} \subseteq A^n$.*

♣

♦ **EXAMPLE 1.21** (See from Example 1.14)**.**

Let $\mathcal{F} = \{e, \cdot\}$ and $\mathcal{P} = \{\leq\}$. Let model $\mathcal{M}$ for $(\mathcal{F}, \mathcal{P})$ be defined as follows:

1. Let A be the set of binary strings over the alphabet $\{0, 1\}$, so $A = \{0, 1\}^*$;
2. let $e^{\mathcal{M}} = \epsilon$, the empty string;
3. let $\cdot^{\mathcal{M}}$ be defined such that $s_1 \cdot^{\mathcal{M}} s_2$ is the concatenation of the strings s_1 and s_2; and
4. let $\leq^{\mathcal{M}}$ be defined such that $s_1 \leq^{\mathcal{M}} s_2$ iff s_1 is a prefix of s_2.

Specific choices for elements of A, as 10001,

- $1010 \cdot^{\mathcal{M}} 1100 \; ? = 10101100$
- $000 \cdot^{\mathcal{M}} \epsilon \; ? = 000$.

Usually, we require that the **equality** predicate $=$ is interpreted as **sameness**.

`Extensionality restriction` means that allowable models are restricted to those

in which $a =^{\mathcal{M}} b$ holds if and only if a and b are the same elements of the model's universe A. E.g., equality in $\mathcal{M}$ is $000 =^{\mathcal{M}} 000$; but $001 \neq^{\mathcal{M}} 100$.

♦ **EXAMPLE 1.22.**

Let $\mathcal{F} = \{z, s\}$ and $\mathcal{P} = \{\leq\}$.

Let model $\mathcal{M}$ for $(\mathcal{F}, \mathcal{P})$ be defined as follows:

1. Let A be the set of natural numbers, so $A = \mathbb{N}$;
2. let $z^{\mathcal{M}} = 0$;
3. let $s^{\mathcal{M}}$ be defined as arity 1 function, from $\mathbb{N}^1 \to \mathbb{N}$, such that $s(n) = n + 1$ [succesor function];
4. let $\leq^{\mathcal{M}}$ be defined such that $n_1 \leq^{\mathcal{M}} n_2$ iff the natural number n_1 is less than or equal to n_2. ■

In both examples we see that

(1) the set $\mathcal{P}$ of predicates has one element $P = \leq$ only, and

(2) predicate $P = \leq$ has arity $n = 2$, could view $P^{\mathcal{M}} = \leq^{\mathcal{M}}$ as binary relation on $\{0, 1\}^*$ and on $\mathbb{N}$.

♣ **OBSERVATION 2.** *Generally, fix model $\mathcal{M}$, let predicate $P \in \mathcal{P}$ has arbitrary arity n then*

(♦) $P^{\mathcal{M}}$ is a n-ary relation on A [see Item 4 of Definition 1.21].

1.4.3 *Satisfaction Relation*

Handling Free Variables

We can give meaning to formulas with *free* variables by providing

an *environment* (lookup table) that assigns variables to elements of our universe:

$$l : \textbf{var} \to A.$$

We define *environment extension* such that notation

$l[x \mapsto a]$ is the environment that maps x to a and any other variable y to $l(y)$.

The last observation (♦) is viewed in the `multivariate` case that, for arity $n \geq 1$

evaluating $(x_1, x_2, \ldots, x_n) \xrightarrow{l} (a_1, a_2, \ldots, a_n)$ implies that $P(a_1, a_2, \ldots, a_n)$ is true.

Definition 1.22 (Satisfaction of a formula by models)

♣

The model $\mathcal{M}$ satisfies formula ϕ with respect to environment l,

written $\mathcal{M} \models_l \phi$ in the following cases.

1. formula ϕ is of the form $P(t_1, t_2, \ldots, t_n)$,
   if the result $(a_1, a_2, \ldots, a_n)$ of evaluating $t_1, t_2, \ldots, t_n$ with respect to l is in $P^{\mathcal{M}}$,
   meaning $P^{\mathcal{M}}(a_1, a_2, \ldots, a_n)$ is true;

2. ϕ has the form $\forall x\, \psi$, if the $\mathcal{M} \models_{l[x \mapsto a]} \psi$ holds **for all** $a \in A$;
3. ϕ has the form $\exists x\, \psi$, if the $\mathcal{M} \models_{l[x \mapsto a]} \psi$ holds **for some** $a \in A$;
4. ϕ has the form $\neg\psi$, if $\mathcal{M} \models_l \psi$ **does not** hold;
5. ϕ has the form $\psi_1 \vee \psi_2$, if $\mathcal{M} \models_l \psi_1$ holds **or** $\mathcal{M} \models_l \psi_2$ holds;
6. ϕ has the form $\psi_1 \wedge \psi_2$, if $\mathcal{M} \models_l \psi_1$ holds **and** $\mathcal{M} \models_l \psi_2$ holds; finally

7. ϕ has the form $\psi_1 \to \psi_2$, if $\mathcal{M} \models_l \psi_1$ holds implying $\mathcal{M} \models_l \psi_2$ holds.

♦ **EXAMPLE 1.23.**

Define model M on two universes $A_1 = \mathbb{R}$ (reals) and $A_2 = \mathbb{C}$ (complexes),

with a predicate-form formula $\psi = P(x) = `x^2 \geq 0'$ with arity 1,

and formulas $\phi_1 = \forall x\ \psi$, and $\phi_2 = \exists x\ \psi$.

Then M satisfies formula ϕ_1 on A_1, since $\mathcal{M} \models_l \psi$ holds **for all** $a \in A_1$.

M **does not** satisfy ϕ_1 on A_2 [with a counter example of $a = i,\ i^2 = -1$?].

But M satisfies formula ϕ_2 on the complexes A_2, with respect to environment l

since $\mathcal{M} \models_l \psi$ holds **for some** $a \in A_2$; ... ■

Satisfaction of Closed Formula (one with no free variables)

If a formula ϕ has **no free variables**, we call ϕ a *sentence*.

$\mathcal{M} \models_l \phi$ either holds or does not hold for sentence ϕ regardless of the choice of l.

Thus we conclude $\mathcal{M} \models \phi$ or $\mathcal{M} \not\models \phi$ for any sentence ϕ.

Semantic Entailment, Satisfiability, and Validity

Let Γ be a (possibly infinite) set of formulas in predicate logic and a formula $\psi \in \Gamma$.

Definition 1.23 (Four concepts motivated by models) ♣

1. **Entailment**: $\Gamma \models \psi$

   if and only if (iff) for all models $\mathcal{M}$ [Definition 1.21] and environments l,

   whenever $\mathcal{M} \models_l \phi$ holds for all $\phi \in \Gamma$, then $\mathcal{M} \models_l \psi$.

2. **Satisfiability of Formula**: formula ψ is satisfiable

   iff there is **some** model $\mathcal{M}$ and some environment l such that $\mathcal{M} \models_l \psi$ holds.

3. **Satisfiability of Formula Sets**: The whole set Γ is satisfiable

   iff there is **some** model $\mathcal{M}$ and some environment l such that $\mathcal{M} \models_l \phi$, for all $\phi \in \Gamma$.

4. **Validity**: Formula ψ is **valid** iff for **all models** $\mathcal{M}$ and environments l, we have $\mathcal{M} \models_l \psi$.

1.4.4 *Soundness and Completeness of Predicate Logic*

- `Entailment ranges over models`

  Semantic entailment between sentences:

  $$\phi_1, \phi_2, \ldots, \phi_n \models \psi$$

  requires that in *all* models that satisfy $\phi_1, \phi_2, \ldots, \phi_n$, the sentence ψ is satisfied.

- `How to effectively argue about all possible models?`

  Usually the number of models is infinite; it is very hard to argue on the semantic level in predicate logic. Can we use natural deduction for showing entailment?

Central Result of Natural Deduction (Kurt Gödel, 1929 in his doctoral dissertation.)

$$\phi_1, \ldots, \phi_n \models \psi \qquad \textbf{iff} \qquad \phi_1, \ldots, \phi_n \vdash \psi$$

[The $\vdash$ is viewed in Definition 1.20, the entailment $\models$ is treated in Definition 1.23.]

We continue in *predicate logic* with some negative results.

Problems of Propositional and Predicate Logic:

Given a formula ϕ in *propositional* logic we can, in principle,

determine whether $\models \phi$ holds:

FACT: If ϕ has n propositional atoms, then the truth table of ϕ contains 2^n lines; and

$\models \phi$ holds if, and only if, the column for ϕ (of length 2^n) contains only `True` entries.

Such a mechanical procedure **cannot** be provided in *predicate logic*.

The decision problem at hand is this: **Validity in predicate logic**.

Given a logical formula ϕ in predicate logic, does $\models \phi$ hold, yes or no?

We now show that this problem is **not** solvable, using a well-known technique called *problem reduction*. That is, we take some other problem, of which we already know that it is **not solvable**, and we then show that the solvability of our problem entails the solvability of the other one.

Definition 1.24

Relevant concepts of ***Decidability*** *include the followings.*

1. **Decision problem**

   It a question in some formal system with a yes-or-no answer*.*

2. ***Decidability****:*

   Decision problems for which there is an algorithm

   that returns "yes" whenever the answer to the problem is "yes",

   and that returns "no" whenever the answer to the problem is "no",

   are called decidable*.*

3. **Decidability of satisfiability**

   The question, whether a given propositional formula is satisifiable*, is* `decidable`*.*

♣

In predicate logic, however, some negative results have been found.

Theorem 1.4 (`On Undecidability of Predicate Logic`)

The decision problem of validity in predicate logic is **undecidable***:*

No program *exists which, given any language in predicate logic and*

any formula ϕ *in that language, decides whether* $\models \phi$*.*

♡

The proof employs two concepts- problems: [follow A. Church's proof]

1. Post Correspondence Problem
2. Turing machines.

STEPS of proving (sketch)

- Establish that the Post Correspondence Problem (PCP) is undecidable
- Translate an arbitrary PCP, say C, to a formula ϕ.
- Establish that $\models \phi$ holds if and only if C has a solution.
- Conclude that validity of pred. logic formulas is undecidable. (See more in Theorem 2.22- REF. 2)

Undecidability of Post Correspondence Problem

Post Correspondence Problem (PCP): Can we line up copies of the cards such that the top row spells out the same sequence as the bottom row?

Formalization: Given a finite sequence of pairs $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$ such that all s_i and t_i are binary strings of positive length, is there a sequence of indices $i_1, i_2, \ldots, i_n$ with $n \geq 1$ such that the concatenations $s_{i_1} s_{i_2} \ldots s_{i_n}$ and $t_{i_1} t_{i_2} \ldots t_{i_n}$ are equal?

Turing machines: Basic abstract symbol-manipulating devices that can simulate in prinicple any computer algorithm [having seen in Chapter 2 of Automata].

The input is a string of symbols on a *tape*, and

the machine "accepts" the input string, if it reaches one of a number of *accepting states*.

Termination of Programs is Undecidable, whether program with input terminates.

`Proof idea`: For a Turing machine with a given input, construct a PCP such that a solution of the PCP exists if and only if the Turing machine accepts the solution.

Proof:

1. **Bits as Functions** Represent bits 0 and 1 by functions f_0 and f_1.
2. **Strings as Terms** Represent the empty string by a constant e.

The string $b_1 b_2 \dots b_l$ corresponds to the term $f_{b_l}(f_{b_{l-1}} \dots (f_{b_2}(f_{b_1}(e))) \dots)$

3. **Formula** $P(s,t)$: from in sequence of pairs $(s_1,t_1),(s_2,t_2),\dots,(s_k,t_k)$, let C be the PCP problem $\begin{matrix} s_1 & s_2 & \dots & s_k \\ t_1 & t_2 & \dots & t_k. \end{matrix}$

   $P(s,t)$ then holds iff $\exists\ (i_1,i_2,\dots,i_m)$ of indices such that $s = s_{i_1} s_{i_2} \dots s_{i_m}$ and $t = t_{i_1} t_{i_2} \dots t_{i_m}$.

4. **Build the formula** ϕ for solution $\phi_C = \phi = \phi_1 \wedge \phi_2 \rightarrow \phi_3$, where

$$\begin{aligned} \phi_1 &= \bigwedge_{i=1}^{k} P(f_{s_i}(e), f_{t_i}(e)) \\ \phi_2 &= \forall v \forall w (P(v,w) \rightarrow \bigwedge_{i=1}^{k} P(f_{s_i}(v), f_{t_i}(w))) \\ \phi_3 &= \exists z P(z,z) \end{aligned}$$

Undecidability of Predicate Logic- SUMMARY

- Post correspondence problem is undecidable.
- Constructed formula ϕ_C for Post correspondence problem C.
- CLAIM: $\models \phi_C$ holds if and only if C has a solution.
- Proof via construction of ϕ_C. Formally construct an interpretation of strings and show that whenever there is a solution, the formula ϕ_C holds and vice versa.

Undecidability of Predicate Logic

The decision problem of validity in predicate logic is **undecidable**:

No program exists which, given any language in predicate logic and

any formula ϕ in that language, decides whether $\models \phi$.

1.4.5 *Compactness of Predicate Calculus*

Proposition 1.1 (`Compactness Theorem`)

Let Γ be a set of sentences of predicate logic.

If all finite subsets of Γ are satisfiable, then Γ is satisfiable. [See Item 3 of Definition 1.23.]

♠

Proof Use Method of Contradiction.

Assume Γ is **not** satisfiable. [See Definition 1.23 Item 3 for `Satisfiability of Formula Set`.]

We thus have $\Gamma \models \bot$ (contradiction). Via completeness, we have $\Gamma \vdash \bot$.

The proof is finite, thus only uses a finite subset $\Delta \subset \Gamma$ of premises.

Thus, $\Delta \vdash \bot$, and $\Delta \models \bot$ via soundness [Kurt Gödel 's Theorem]. ■

REMINDER: Required Texts/Materials

Electronic copies of [2-6] are available on the WWW

1. Handouts (Obtained via emails.)
2. Michael R.A. Huth and Mark D. Ryan.*Logic in Computer Science* (2nd Ed.), Cambridge Uni. Press, 2004. (**Ch. 1, 2**)

3. Michael R.A. Huth and Mark D. Ryan. *Logic in Computer Science: Solutions to designated exercises* (2nd Ed.), Cambridge University Press, 2004. (**Chapters 1, 2**)
4. F.R. Giordano, W.P. Fox & S.B. Horton,
   A First Course in Mathematical Modeling, 5th ed., Cengage, 2014.
5. K. M. Bliss K. R. Fowler B. J. Galluzzo, Math Modeling: getting started & getting solutions. Society for Industrial and Applied Mathematics (SIAM) Handbook, 2014.
6. Peter Linz. *An Introduction to Formal Languages and Automata* (3rd Ed.) Jones and Bartlett, 2001. (**Chap. 1-6**)
   An Introduction to Formal Languages and Automata: Instructors' Manual (**Chapters 1-6**)
7. John Hopcroft et. al. *Introduction to Automata Theory, Languages, and Computation* (**Cha 1-5**)

Homework: You should do as much as you can ALL marked exercises in [2, Sect. 2.8] (notice that sample solutions for these exercises are available in [3]).

[3]: Michael Huth and Mark Ryan. Logic in Computer Science: Solutions

For this lecture, the following are recommended exercises [2]:

- 2.1: 1a); 2a)
- 2.2: 6
- 2.3: 1a); 1b); 6a); 6b); 6c); 7b); 9b); 9c); 13d)
- 2.4: 2); 3); 11a); 11c); 12e); 12f); 12h); 12k)
- 2.5: 1c); 1e).

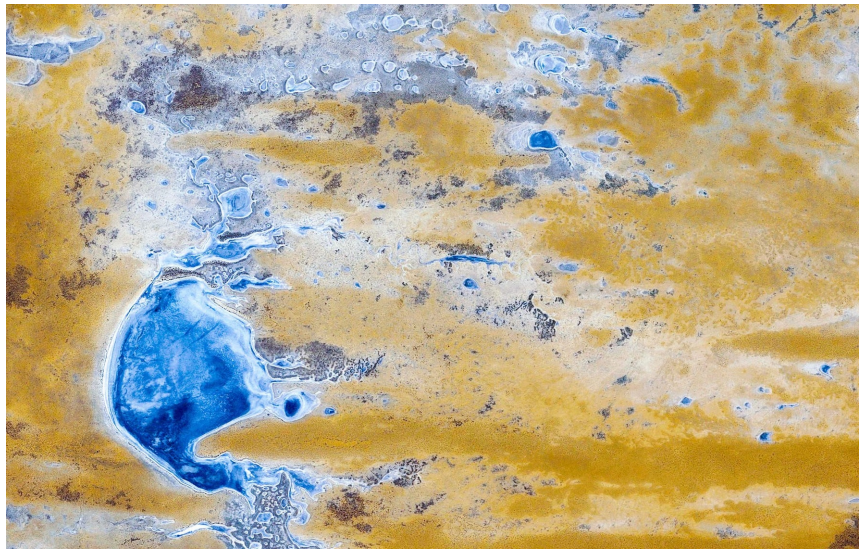
WHAT NEXT?

Logic for Program Verification, in Section 1.5

Hoare Triples and Partial and Total Correctness, Section 1.6

Practical Aspects of Correctness Proofs in Section 1.7

PART V: Logic for Program Verification



[[GoogleMap]]

1.5 Logic for Program Verification

a) Are there expressions in Predicate Logic that **do not** evaluate to TRUE or FALSE? An example?

Ans.: Yes! **Terms**, unlike predicates and formulas, do not evaluate to the distinguished symbols true or false. Examples of terms include:

a, a constant (or 0-ary function);

x, a variable; $f(t)$, a unary function f applied to a term t.

b) How do you represent a propositional variable in a Predicate Logic formula?

Ans.: As a 0-ary predicate.

c) **Fermat's Last Theorem** in number theory says that:

It is impossible to separate any power higher than the second into two like powers.

Or, more precisely: '**If** an integer n is greater than 2,

then the equation $x^n + y^n = z^n$ has no solutions in positive integers x, y, and z.'

Formulate the above statement in Predicate Logic with Equality? DIY

An answer to Fermat's Last Theorem Formulation:

$\forall n.integer(n) \wedge n > 2 \implies \Big[\forall x, y, z.integer(x) \wedge integer(y) \wedge integer(z)$

$\wedge\, x > 0 \wedge y > 0 \wedge z > 0 \longrightarrow x^n + y^n \neq z^n\Big]$.

INITIAL EXAMPLE on imperative programming language

A *binary search* function written in an imperative programming language:

```
bool binarySearch ( int [] a, int l, int u, int e)
{ if (l > u) return false ;
```

```
else {int m = (l + u) div 2;
if (a[m] == e) return true ;
else if (a[m] < e) return binarySearch (a, m + 1, u, e);
else return binarySearch (a, l, m - 1, e);}
}
```

♣ **OBSERVATION 3.**

- As a first step towards determining whether an **implementation** (such as that in the function above) fulfills its **specification**, the specification has to be formalized.

  We do so in terms of *preconditions* and *postconditions.*

1.5.1 *Preconditions and postconditions*

Definition 1.25

- *A precondition specifies what should be true upon entering the function,*

  meaning, under what inputs the function is expected to work.

- *The postcondition is a formula G whose free variables include only*

  the formal parameters and

  the special variable rv representing the return value of the function.

♣

The postcondition relates the function's output (the rv) to its input (the parameters).

Question 1.

Formulate in Predicate Logic the precondition/postcondition for `binarySearch`.

They are

- First precondition: $0 \leq l \wedge u < |a|$
- Second precondition:

  $\forall i, j.\ integer(i) \wedge integer(j) \wedge 0 \leq i \leq j < |a| \longrightarrow a[i] \leq a[j]$
- Postcondition: $rv \longleftrightarrow \exists\, i :\ l \leq i \leq u \wedge a[i] = e$

Brief Program Verification: We need to verify the correctness of computer systems (hardware, software, or a combination). This is most obvious in the case of

safety-critical systems (i.e., petroleum exploration), and also

commercially critical (as industrial manufacturing).

1.5.2 *Program verification: Why (Motivation) and What (Approaches)?*

Favours of program verification include the following:

- **Documentation**: The program's formal specification is important because the logical structure and properties (theorems) typically serve as guiding principles for an implementation later.
- **Time-to-market**: verifying programs with respect to formal specifications can significantly cut down the duration of software development and maintenance
- **Certification - warranty**: verification is required in safety-critical domains [3].

[3] such as nuclear power stations and aircraft cockpits.

Broad Approaches to verification can be classified according to the following criteria:

a) Proof-based checking vs. b) Model-based (model checking)

1. In approach a)- Proof-based checking, the verification method consists of trying
   to find a **proof** that $\Gamma \vdash \varphi$, [provability, defined in Definition 1.20]
   where Γ is a set of formulas for the *system description*, and
   formula φ represents the *system specification*.
2. In approach b)- Model checking- the system is represented by a model M for an appropriate logic. The system specification is again represented by a formula φ and
   the verification method consists of computing whether M satisfies φ,
   written $\mathcal{M} \models \phi$ [i.e. $\mathcal{M} \models_l \phi$, for certain environment l], see Definition 1.23.

`Problems with the` *`model checking`* `approach:`

- Models become infinite.
- Satisfaction/validity becomes undecidable.

In this part, we only cover a **proof-based framework** for program verification [to avoid checking infinitely many models of a set of predicate logic formulas in order to establish the validity of an Entailment $\mathcal{M} \models \phi$].

1.5.3 *Program Verification (PV): Characteristics and Framework*

We consider few characteristics of PV as follows.

- Semi-automatic: not always be carried out algorithmically by a computer

- Property-oriented: verify properties of a program rather than a full specification of its behaviour
- Application domain: use sequential programs (means that we assume the program runs on a single processor and that there are *no concurrency issues)*.

A framework for producing the software

1. **Listen** to all the requirements R [with informality info] from the customer,
2. **Convert** the informal description R into an 'equivalent' formula ϕ_R, of some symbolic logic,
3. **Write** a program P meeting/realizing ϕ_R, in suitable programming environment,
4. **Prove** that P satisfies the formula ϕ_R.

DISCUSSION

Production of a software, targeting to management projects, such as

a) Providing nationally truthful COVID19-related fluctuation, in Health Economics,

b) Providing Airline booking service, in Travel- Hospitality industry.

What are the four steps? Which one is most critical? ■

1.5.4 *Core programming language*

- Popular *imperative* programming languages consists of
  assignments to integer- and boolean-valued variables,
  if-statements, while-statements

and sequential compositions.

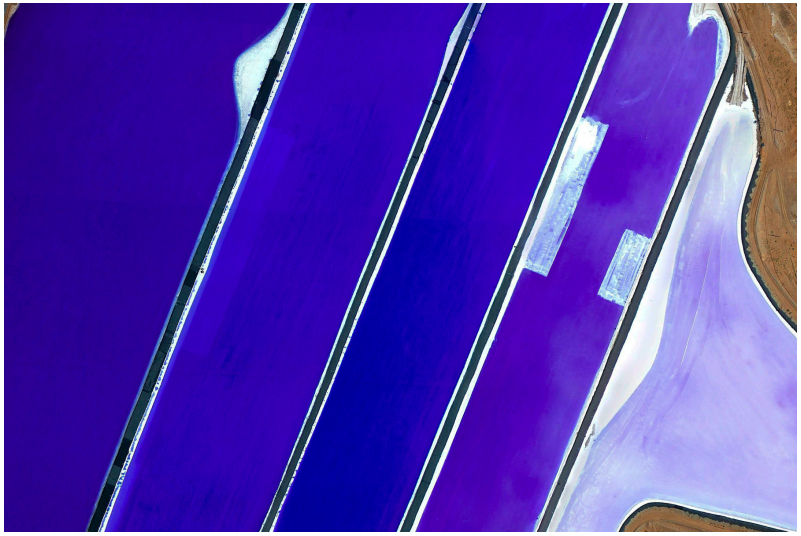
- Core language (discussed in this section) has three syntactic domains:
  Arithmetic (integer) expressions, Boolean expressions and Commands [called programs].

Grammatical Points in Core Language (use Backus Naur form throughout)

1. **Arithmetic expressions** E:
   $E ::= n \mid x \mid (-E) \mid (E + E) \mid (E - E) \mid (E * E)$
2. **Boolean expressions** B:
   $$B ::= \texttt{true} \mid \texttt{false} \mid (!B) \mid (B \& B) \mid (B \| B) \mid (E < E)$$
   NOTE: Boolean expressions are built on top of integer expressions, why?
3. **Command** C [with expressions as its components]:
   $$C ::= x = E \mid C; C \mid \texttt{if}\ B\ \{C\}\ \texttt{else}\ \{C\} \mid \texttt{while}\ B\ \{C\}$$
   where the braces $\{$ and $\}$ are to mark the extent of the blocks of code in the **if**-statement and the **while**-statement, as in languages such as *C* and *Java*.

The atomic command $x = E$ is the usual assignment statement; it evaluates the integer expression E in the current state of the store and then overwrites the current value stored in x with the result of that evaluation.

PART VI: Hoare Triples



[Courtesy Andre Derain]

1.6 Hoare Triples - Partial and Total Correctness

Question 2.

What syntax should we use for ϕ_R, the formal specifications of requirements for such programs? We need to be able to talk **not just** about the state *after* the program executes,

but also about the state *before* it executes.

We make the assertions, therefore of triplet forms,

$$(\!|\phi|\!)\ P\ (\!|\psi|\!)$$

Pre-conditions: ϕ

Post-conditions: ψ

Informal meaning: If the program P is run in a state that satisfies ϕ, then the state resulting from P's execution will satisfy ψ. We may informally say that:

An assertion of the form $(\!|\phi|\!)\ P\ (\!|\psi|\!)$ is called a **Hoare triple**.

- ϕ is called the precondition, ψ is called the postcondition.
- A state of a Core program P is a function l that assigns each variable x in P to an integer $l(x)$.

♦ **EXAMPLE 1.24.** *Consider the factorial function* $n!$.

```
y = 1; z = 0;
```

```
while (z != x) { z = z + 1; y = y * z; }
```

- We need to be able to say that at the end, y is x!

  That means we require a *post-condition* $y = x$!

- Do we need pre-conditions, too?

  Yes, they specify what needs to be the case before execution.

  Example: $x > 0$

♦ **EXAMPLE 1.25.**

Informal specification:

Given a positive number x, the program P calculates a number y whose square is less than x.

Assertion

$$(\!|x > 0|\!)\ P\ (\!|y \cdot y < x|\!)$$

If define program $P : y = 0$ then

Our Hoare triple $(\!|x > 0|\!)$ `y = 0` $(\!|y \cdot y < x|\!)$

Another example for program P

```
y = 0;
while (y * y < x) {
y = y + 1;
}
```

```
y = y - 1;
```

Then we get another Hoare triple

$$(\!|x > 0|\!)\ \mathrm{P}\ (\!|y \cdot y < x|\!)$$

REMINDER 1- Models in Predicate Logic:

Let $\mathcal{F}$ contain **function** symbols and $\mathcal{P}$ contain **predicate** symbols.

A model $\mathcal{M}$ for $(\mathcal{F}, \mathcal{P})$ consists of:

1. A non-empty set A, the *universe*;
2. for each nullary function symbol $f \in \mathcal{F}$ a concrete element $f^{\mathcal{M}} \in A$;
3. for each $f \in \mathcal{F}$ with arity $n > 0$, a concrete function $f^{\mathcal{M}} : A^n \to A$;
4. for each $P \in \mathcal{P}$ with arity $n > 0$, a set $P^{\mathcal{M}} \subseteq A^n$.

REMINDER 2: Satisfaction of a formula by models

The model $\mathcal{M}$ satisfies formula ϕ with respect to environment l, written $\mathcal{M} \models_l \phi$:

- in case ϕ is of the form $P(t_1, t_2, \ldots, t_n)$, if the result $(a_1, a_2, \ldots, a_n)$ of evaluating $t_1, t_2, \ldots, t_n$ with respect to l is in $P^{\mathcal{M}}$, i.e. $P^{\mathcal{M}}(a_1, a_2, \ldots, a_n)$ is true;

- in case ϕ has the form $\forall x\ \psi$, if the $\mathcal{M} \models_{l[x \mapsto a]} \psi$ holds **for all** $a \in A$;
- in case ϕ has the form $\exists x\ \psi$, if the $\mathcal{M} \models_{l[x \mapsto a]} \psi$ holds **for some** $a \in A$;

- in case ϕ has the form $\neg\psi$, if $\mathcal{M} \models_l \psi$ **does not** hold;
- in case ϕ has the form $\psi_1 \vee \psi_2$, if $\mathcal{M} \models_l \psi_1$ holds **or** $\mathcal{M} \models_l \psi_2$ holds;
- in case ϕ has the form $\psi_1 \wedge \psi_2$, if $\mathcal{M} \models_l \psi_1$ holds **and** $\mathcal{M} \models_l \psi_2$ holds; finally
- in case ϕ has the form $\psi_1 \to \psi_2$, if $\mathcal{M} \models_l \psi_1$ holds implying $\mathcal{M} \models_l \psi_2$ holds.

Definition 1.26

*An assertion of the form $(\!|\phi|\!)\ P\ (\!|\psi|\!)$ is called a **Hoare triple**.*

ϕ is called the precondition, ψ is called the postcondition.

♣

- A state of a Core program P is a function l that assigns each variable x in P to an integer $l(x)$.
- A state l satisfies precondition ϕ if $\exists\ \mathcal{M} : \mathcal{M} \models_l \phi$, where model $\mathcal{M}$ contains integers and gives the usual meaning to the arithmetic operations. Write $l \models \phi$.
- Quantifiers in ϕ and ψ bind only variables that do *not* occur in the program P.

♦ **EXAMPLE 1.26.**

Let $l(x) = -2$, $l(y) = 5$ and $l(z) = -1$. Let model $\mathcal{M}$ have the universe $\mathbb{Z}$, we have:

- $l \models \neg(x + y < z)$
- $l \not\models y = x \cdot z < z$
- $l \not\models \forall u\ (y < u \to y \cdot z < u \cdot z)$

1.6.1 Partial and total correctness

Regarding the termination of a program P in the Hoare triple $(\!|\phi|\!)\ P\ (\!|\psi|\!)$ we study

1) *partial correctness* (means that we do not require the program to terminate), and

2) *total correctness* (we insist upon its termination).

Definition 1.27 ♣

1. We say that the triple $(\!|\phi|\!)\ P\ (\!|\psi|\!)$ is *satisfied under partial correctness* if,
   for all states which satisfy ϕ, the state resulting from P's execution satisfies ψ,
   provided that P terminates. We write $\models_{\text{par}} (\!|\phi|\!)\ P\ (\!|\psi|\!)$.
2. We say that the triple $(\!|\phi|\!)\ P\ (\!|\psi|\!)$ is *satisfied under total correctness* if,
   for all states which satisfy ϕ,
   P is guaranteed to terminate and the resulting state satisfies ψ.
   We write $\models_{\text{tot}} (\!|\phi|\!)\ P\ (\!|\psi|\!)$.

♦ **EXAMPLE 1.27** (Partial correctness)**.**

The Hoare triple $(\!|\phi|\!)$ `while true { x = 0; }` $(\!|\psi|\!)$ holds for all ϕ and ψ.

♦ **EXAMPLE 1.28** (Total correctness)**.**

Consider program $P(x) = \texttt{Fac1}(x)$:

```
y = 1;
```

```
z = 0;
while (z != x) { z = z + 1; y = y * z; }
```

Note: $\top$ means `tautology`, [and $\bot$ means `contradiction`].

- $\models_{\text{tot}} (\!|x \geq 0|\!)\ \texttt{Fac1}\ (\!|y = x!|\!)$
- $\not\models_{\text{tot}} (\!|\top|\!)\ \texttt{Fac1}\ (\!|y = x!|\!)$

- $\models_{\text{par}} (\!|x \geq 0|\!)\ \texttt{Fac1}\ (\!|y = x!|\!)$
- $\models_{\text{par}} (\!|\top|\!)\ \texttt{Fac1}\ (\!|y = x!|\!)$

1.6.2 Proof Calculus for Partial Correctness

If the partial correctness of triples $(\!|\phi|\!)\ P\ (\!|\psi|\!)$ can be proved in the *partial-correctness calculus*, we say that the sequent $\vdash_{\text{par}} (\!|\phi|\!)\ P\ (\!|\psi|\!)$ is valid. In other words,

$\vdash_{\text{par}} (\!|\phi|\!)\ P\ (\!|\psi|\!)$ means it is provably correct according to our calculus, being developed now. We look for a **proof calculus** that allows us to establish

$$\vdash_{\text{par}} (\!|\phi|\!)\ P\ (\!|\psi|\!)$$

where

- $\models_{\text{par}} (\!|\phi|\!)\ P\ (\!|\psi|\!)$ holds whenever $\vdash_{\text{par}} (\!|\phi|\!)\ P\ (\!|\psi|\!)$ (correctness), and
- $\vdash_{\text{par}} (\!|\phi|\!)\ P\ (\!|\psi|\!)$ holds whenever $\models_{\text{par}} (\!|\phi|\!)\ P\ (\!|\psi|\!)$ (completeness).

[Read ϕ = phi, ψ = psi, η = eta...]

Rules for Partial Correctness

$$(1)\quad \frac{(\!|\phi|\!)\ C_1\ (\!|\eta|\!) \qquad (\!|\eta|\!)\ C_2\ (\!|\psi|\!)}{(\!|\phi|\!)\ C_1; C_2\ (\!|\psi|\!)}\ [\text{Composition}];$$

$$(2)\quad \frac{}{(\!|[x \to E]\psi|\!)\ x = E\ (\!|\psi|\!)}\ [\text{Assignment}];$$

$$(3)\quad \frac{(\!|\phi \wedge B|\!)\ C_1\ (\!|\psi|\!) \qquad (\!|\phi \wedge \neg B|\!)\ C_2\ (\!|\psi|\!)}{(\!|\phi|\!)\ \texttt{if}\ B\ \texttt{\{}\ C_1\ \texttt{\}}\ \texttt{else}\ \texttt{\{}\ C_2\ \texttt{\}}\ (\!|\psi|\!)}\ [\text{If-statement}];$$

$$(4)\quad \frac{(\!|\psi \wedge B|\!)\ C\ (\!|\psi|\!)}{(\!|\psi|\!)\ \texttt{while}\ B\ \texttt{\{}\ C\ \texttt{\}}\ (\!|\psi \wedge \neg B|\!)}\ [\text{Partial-while}];\ \text{and}$$

$$(5)\quad \frac{\vdash_{AR} \phi' \to \phi \qquad (\!|\phi|\!)\ C\ (\!|\psi|\!) \qquad \vdash_{AR} \psi \to \psi'}{(\!|\phi'|\!)\ C\ (\!|\psi'|\!)}\ [\text{Implied}].$$

- Rule Implied: a sequent $\vdash_{AR} \phi' \to \phi$ is valid **iff** there is a proof of ϕ in the natural deduction calculus for predicate logic, where ϕ' and standard laws of arithmetic are *premises*.
- Note that the rule Implied allows the precondition to be *strengthened* (we assume more than we need to), while the postcondition is *weakened* (i.e. we conclude less than we are entitled to).
- The rule Implied acts as a link between *program logic* and a suitable extension of *predicate logic*. It allows us to import proofs in predicate logic [enlarged with the basic facts of arithmetic, which are required for reasoning about integer expressions], into the

proofs in program logic.

$$(5)\quad \frac{\vdash_{AR} \phi' \to \phi \qquad (\!|\phi|\!)\ C\ (\!|\psi|\!) \qquad \vdash_{AR} \psi \to \psi'}{(\!|\phi'|\!)\ C\ (\!|\psi'|\!)}\,[\text{Implied}].$$

♦ **EXAMPLE 1.29** (On Assignment)**.**

Let P be the program `x = x + 1`. Using rule (2) of Assignment

we can prove:

- $(\!|x+1=2|\!)\ P\ (\!|x=2|\!)$
- $(\!|x+1=y|\!)\ P\ (\!|x=y|\!)$

1.7 Practical Aspects of Correctness Proofs

Proofs have tree shape: All rules have the structure below, named **Proof Tableaux**

$$\frac{\text{something}}{\text{something else}}$$

As a result, all proofs can be written as a tree.

These trees, however, tend to be very wide when written out on paper.

1.7.1 *Proof Tableaux*

Thus we are using a linear format, called *proof tableaux*, providing 2 patterns.

I) Interleave Formulas with Code:

$$\frac{(\!|\phi|\!)\ C_1\ (\!|\eta|\!) \qquad (\!|\eta|\!)\ C_2\ (\!|\psi|\!)}{(\!|\phi|\!)\ C_1; C_2\ (\!|\psi|\!)}\text{[Composition]}$$

Shape of rule suggests format for proof of series of programs $C_1; C_2; \ldots; C_n$:

$(\!|\phi_0|\!)$

$C_1;$

$(\!|\phi_1|\!)$ justification

$C_2;$

$\vdots$

$(\!|\phi_{n-1}|\!)$ justification

$C_n;$

$(\!|\phi_n|\!)$ justification

II) Working Backwards: Need to find a proof that at the end of executing a program P, some condition ψ holds. If program P has the shape $C_1; \ldots; C_n$, we need to find the weakest formula ψ' such that $(\!|\psi'|\!)\ C_n\ (\!|\psi|\!)$.

Terminology : The weakest formula ψ' is called *weakest precondition*.

♦ **EXAMPLE 1.30.** *Consider the following proof tableaux:*

$(\!|y < 3|\!)$

$(\!|y + 1 < 4|\!)$ Implied

```
y = y + 1;
```

$(\!|y < 4|\!)$ Assignment

1. Can we claim $u = x + y$ after `z = x; z = z + y; u = z;` ?

$(\!|\top|\!)$

$(\!|x + y = x + y|\!)$ Implied

`z = x;`

$(\!|z + y = x + y|\!)$ Assignment

`z = z + y;`

$(\!|z = x + y|\!)$ Assignment

`u = z;`

$(\!|u = x + y|\!)$ Assignment

1.7.2 *An Alternative Rule for If*

We have:

$$\frac{(\!|\phi \wedge B|\!)\ C_1\ (\!|\psi|\!) \qquad (\!|\phi \wedge \neg B|\!)\ C_2\ (\!|\psi|\!)}{(\!|\phi|\!)\ \texttt{if}\ B\ \texttt{\{}\ C_1\ \texttt{\}}\ \texttt{else}\ \texttt{\{}\ C_2\ \texttt{\}}\ (\!|\psi|\!)}\ [\text{If-statement}]$$

Sometimes, the following *derived rule* is more suitable:

$$\frac{(\!|\phi_1|\!)\ C_1\ (\!|\psi|\!) \qquad (\!|\phi_2|\!)\ C_2\ (\!|\psi|\!)}{(\!|(B \to \phi_1) \wedge (\neg B \to \phi_2)|\!)\ \texttt{if}\ B\ \texttt{\{}\ C_1\ \texttt{\}}\ \texttt{else}\ \texttt{\{}\ C_2\ \texttt{\}}\ (\!|\psi|\!)}\ [\text{If-Statement 2}]$$

♦ **EXAMPLE 1.31.** *Consider this implementation of program* `Succ`*: [input* x*?]*

```
a = x + 1;
```

```
if (a - 1 == 0) {y = 1;}
else {y = a;}
```

Can we prove $(\!|\top|\!)$ `Succ` $(\!|y = x + 1|\!)$?

GUIDANCE for solving.

- This program is the `sequential composition` of an *assignment* and an *if-statement*.
- We push the **postcondition** (here $y = x + 1$) upwards through branches C_i of the if-statement, get the results ϕ_i. Thus, we need to obtain a suitable *midcondition* to put between the if-statement and the assignment. Indeed, let

$$\begin{cases} \phi_1 : \ 1 = x + 1; \\ \phi_2 : \ a = x + 1; \end{cases}$$

  we obtain the midcondition $(\!|(a - 1 = 0 \to 1 = x + 1) \wedge \ (\neg(a - 1 = 0) \to a = x + 1)|\!)$.
  We just calculated the weakest $\phi = (B \to \phi_1) \wedge (\neg B \to \phi_2)$
  such that we get the **postcondition** by the If-Statement 2

$$\frac{(\!|\phi_1|\!)\ C_1\ (\!|\psi|\!) \qquad (\!|\phi_2|\!)\ C_2\ (\!|\psi|\!)}{(\!|(B \to \phi_1) \wedge (\neg B \to \phi_2)|\!)\ \texttt{if}\ B\ \texttt{\{}\ C_1\ \texttt{\}}\ \texttt{else}\ \texttt{\{}\ C_2\ \texttt{\}}\ (\!|\psi|\!)}\,[.]$$

- The partial proof now looks like this:

$\vdots$

```
if ( a - 1 == 0 ){
```

$(\!|1 = x + 1|\!)$ If-Statement 2

```
  y = 1;
```

$(\!|y = x + 1|\!)$ Assignment

```
} else {
```

$(\!|a = x + 1|\!)$ If-Statement 2

```
  y = a;
```

$(\!|y = x + 1|\!)$ Assignment

```
}
```

$(\!|y = x + 1|\!)$ If-Statement 2

- We finally push the long formula (midcondition) above the if-statement through the assignment $a = x + 1!$, to obtain the proof

$(\!|\top|\!)$

$(\!|(x+1-1=0 \to 1=x+1) \wedge (\neg(x+1-1=0) \to x+1=x+1)|\!)$ Implied

```
a = x + 1;
```

$(\!|(a-1=0 \to 1=x+1) \wedge (\neg(a-1=0) \to a=x+1)|\!)$ Assignment

```
if ( a - 1 == 0 ){
```

$(\!|1=x+1|\!)$ If-Statement 2

```
  y = 1;
```

$(\!|y=x+1|\!)$ Assignment

```
} else {
```

$(\!|a=x+1|\!)$ If-Statement 2

```
  y = a;
```

$(\!|y=x+1|\!)$ Assignment

■

1.7.3 *Correctness of the Factorial Function*

Partial-while Rule Revisited

$$\frac{(\!|\psi \wedge B|\!)\ C\ (\!|\psi|\!)}{(\!|\psi|\!)\ \texttt{while}\ B\ \{\ C\ \}\ (\!|\psi \wedge \neg B|\!)}\ [\text{Partial-while}]$$

ELUCIDATION: In the premise $(\!|\psi \wedge B|\!)\ C\ (\!|\psi|\!)$, the formula ψ is chosen to be an *invariant* of the body C of the while-

statement: provided the boolean guard B is true, if ψ is true before we start C, and C terminates, then it is also true at the end.

♦ **EXAMPLE 1.32 (Factorial Example).**

We shall show that the following Core program `Fac1(x)` meets this specification:

```
y = 1;
z = 0;
while (z != x) { z = z + 1; y = y * z; }
```

Thus, to show $(\!|\top|\!)$ `Fac1` $(\!|y = x!|\!)$ we begin with

$\vdots$

$(\!|y = z!|\!)$

`while ( z != x ){`

$(\!|y = z! \wedge z \neq x|\!)$	Invariant
$(\!|y \cdot (z+1) = (z+1)!|\!)$	Implied
`z = z + 1;`	
$(\!|y \cdot z = z!|\!)$	Assignment
`y = y * z;`	
$(\!|y = z!|\!)$	Assignment

`}`

$(\!|y = z! \wedge \neg(z \neq x)|\!)$	Partial-while
$(\!|y = x!|\!)$	Implied

then pushing the postcondition ψ (here $y = x!$) upwards through a while- statement

to meet the precondition ϕ (here $(\!|\top|\!)$), we have

$(\!|\top|\!)$

$(\!|(1 = 0!)|\!)$ Implied

```
y = 1;
```

$(\!|y = 0!|\!)$ Assignment

```
z = 0;
```

$(\!|y = z!|\!)$ Assignment

```
while ( z != x ){
  ⋮
}
```

$(\!|y = z! \wedge \neg(z \neq x)|\!)$ Partial-while

$(\!|y = x!|\!)$ Implied ■

1.7.4 *Proof Calculus for Total Correctness*

We just developed a calculus for proving partial correctness of triples

$$(\!|\phi|\!)\ P\ (\!|\psi|\!).$$

Partial correctness does not tell us if P 'loops' indefinitely. We now extend the proof calculus for partial correctness so that it also proves that programs terminate.

Ideas for Total Correctness

- The only source of *non-termination* is the `while` command. Therefore, the proof calculus for total correctness is the same as for

partial correctness for all the rules except the rule for while-statements.

- If we can show that the value of an integer expression decreases in each iteration, but never becomes negative, we have proven termination.

  Why? **Well-foundedness of natural numbers**

- We shall include this argument in a new version of the `while` rule.

PROOF STRUCTURE:

- A **proof of total correctness for a while-statement** will consist of two parts:

  the proof of partial correctness and

  a proof that the given while-statement terminates.

- The proof of termination usually has the following form.

  We identify an **integer expression** whose value can be shown to decrease every time we execute the body of the while-statement: which is always non-negative.

  If we can find an expression with these properties, it follows that the while-statement *must terminate*; because the expression can only be **decremented a finite number of times** before it becomes 0.

  Such integer expressions are called **variants**. ■

♦ **EXAMPLE 1.33** (**Factorial revisited).**

```
y = 1; z = 0;
while (z != x) { z = z + 1; y = y * z; }
```

What could be a good variant E?

E must strictly decrease in the loop, but not become negative. Answer: $E \equiv x - z$

When it is 0, the while-statement terminates. ■

RULES: We describe this intuition in the following rules.

$$\frac{(\!|\psi \wedge B|\!)\ C\ (\!|\psi|\!)}{(\!|\psi|\!)\ \texttt{while}\ B\ \{\ C\ \}\ (\!|\psi \wedge \neg B|\!)}\ [\text{Partial-while},\ recalled]$$

$$\frac{(\!|\psi \wedge B \wedge 0 \leq E = E_0|\!)\ C\ (\!|\psi \wedge 0 \leq E < E_0|\!)}{(\!|\psi \wedge 0 \leq E|\!)\ \texttt{while}\ B\ \{\ C\ \}\ (\!|\psi \wedge \neg B|\!)}\ [\text{Total-while}]$$

The ψ is called the **invariant**, the expression E is called **variant**.

In the 2nd rule, E is the expression whose value decreases with each execution of the body C. This is coded by saying that,

if its value equals that of the logical variable E_0 before the execution of C,

then it is strictly less than E_0 after it – yet still it remains non-negative.

* NOTE: in practical computation, as in Econometric or Data analytics, with real numbers, E does not need to be integer expression, E can be real expression.

Total Correctness of `Fac1`

We use the rule `Total-while` in tableaux similarly to how we use `Partial-while`, but note that the body of the rule C must now be shown to satisfy

$$(\!|\psi \wedge B \wedge 0 \leq E = E_0|\!)\ C\ (\!|\psi \wedge 0 \leq E < E_0|\!)$$

Let us illustrate this rule by proving that $\vdash_{\text{tot}} (\!|x \geq 0|\!)\ \texttt{Fac1}\ (\!|y = x!|\!)$ is valid.

We knew that $E \equiv x - z$ is a suitable variant. The invariant ($y = z!$) of the partial correctness proof is retained. We obtain the following proof

$\vdots$

$(\!|y = z! \wedge 0 \leq x - z|\!)$

```
while ( z != x ) {
```

$(\!|y = z! \wedge z \neq x \wedge 0 \leq x - z = E_0|\!)$	Invariant
$(\!|y \cdot (z+1) = (z+1)! \wedge 0 \leq x - (z+1) < E_0|\!)$	Implied
`z = z + 1;`	
$(\!|y \cdot z = z! \wedge 0 \leq x - z < E_0|\!)$	Assignment
`y = y * z;`	
$(\!|y = z! \wedge 0 \leq x - z < E_0|\!)$	Assignment

```
}
```

$(\!|y = z! \wedge \neg(z \neq x)|\!)$	Total-while
$(\!|y = x!|\!)$	Implied

Now imposing the precondition **invariant** $\psi = (\!|x \geq 0|\!)$, we get the complete proof for total correctness:

$(\!|x \geq 0|\!)$

$(\!|(1 = 0! \wedge 0 \leq x - 0)|\!)$ Implied

```
y = 1;
```

$(\!|y = 0! \wedge 0 \leq x - 0|\!)$ Assignment

```
z = 0;
```

$(\!|y = z! \wedge 0 \leq x - z|\!)$ Assignment

```
while ( z != x ) {
```

$\vdots$

```
}
```

$(\!|y = z! \wedge \neg(z \neq x)|\!)$ Total-while

$(\!|y = x!|\!)$ Implied

Hence, $\vdash_{\text{tot}} (\!|x \geq 0|\!)$ `Fac1` $(\!|y = x!|\!)$ is valid. ■

FINAL COMMENTS

1. The precondition $x \geq 0$ is crucial in securing the fact that
   $0 \leq x - z$ holds right before the while-statements gets executed: it implies the precondition $(\!|(1 = 0! \wedge 0 \leq x - 0)|\!)$ computed, and the program must terminate because z certainly meets x.

2. The application of `Implied` within the body of the while-statement is valid, but it makes vital use of the fact that **the boolean guard is true**.

This is an example of a while-statement whose boolean guard is needed in reasoning about the correctness of **every** iteration of that while-statement.

3. Finding a working variant E is a creative activity which requires skill, intuition and practice.

1.8 HOMEWORK and Chapter Problems

1. Do all HWs which have not been done in previous lectures.
2. Try to understand deeply the following notations/terms

   arity, expression, term, formula, atomic formula, sentence, clause, Backus Naur form (BNF), parse tree, precondition, postcondition, binding priorities, provability, witness, scope, bound, verification, model checking, Hoare triple, and their other related notation/terms.
3. Do exercise 1.5.14 on page 89 in [2].
4. Consider the following program

   ```
   temp := x
   x := y
   y := temp
   ```

   What does this tinny program do? Find preconditions, postconditions and verify its correctness?
5. **Reviewed problems: [REF. 4: Chapter 2c]**

   It is recommended that you should do as much as you can ALL marked exercises in [2, Sect. 2.8] (notice that sample solutions for these exercises are available in [3]). For this lecture, the following are recommended exercises [2]:

   - 2.1: 1a); 2a)
   - 2.2: 6

- 2.3: 1a); 1b); 6a); 6b); 6c); 7b); 9b); 9c); 13d)
- 2.4: 2); 3); 11a); 11c); 12e); 12f); 12h); 12k)
- 2.5: 1c); 1e).

PROBLEM 1.1.

Find appropriate predicates and their specification to translate the following into predicate logic:

1. Only red things are in the box.
2. No animal is both a cat and a dog.
3. Every prize was won by a boy.
4. A boy won every prize.
5. Any difference between the last two cases, logically?

HINT - SOLUTION 1.

1. $\forall x\,[Inb(x) \longrightarrow Red(x)]$

3. $\forall x\,[Prize(x) \longrightarrow \exists y\,(Boy(y) \wedge Win(y,x))]$

4. and 5. DIY

PROBLEM 1.2.

Let $F(x,y)$ mean that x is the father of y; $M(x,y)$ denotes x is the mother of y.

Similarly, $H(x,y)$, $S(x,y)$, and $B(x,y)$ say that x is the husband/sister/brother of y, respectively. You may also use constants to denote individuals, like 'Ed' and 'Patsy.'

However, you are **not allowed** to use any predicate symbols other than the above to translate the following sentences into predicate logic:

1. Everybody has a mother.
2. Whoever has a mother has a father.

3. All fathers are parents.
4. No uncle is an aunt.

5. Ed and Patsy are husband and wife.
6. Carl is Monique's brother-in-law.

PROBLEM 1.3.

The following sentences are taken from the `RFC3157 Internet Task-force Document` 'Securely Available Credentials – Requirements.'

Specify each sentence in predicate logic, defining predicate symbols as appropriate:

1. An attacker can persuade a server that a successful login has occurred, even if it hasn't.
2. An attacker can overwrite someone else's credentials on the server.
3. All users enter passwords instead of names.
4. Credential transfer both to and from a device MUST be supported.
5. Credentials MUST NOT be forced by the protocol to be present in cleartext at any device other than the end user's.
6. Different end user devices MAY be used to download, upload, or manage the same set of credentials.