Math. For CS
Lecture 1

Chong-Kwon Kim

2022.

What is CS?

- CS is science of

The study of ***complexity***

- How can it be done?
- How well can it be done?
- Can it done at all?

it : a process that transforms information from one from to another

What is CS?

AI

How can it be done?
How well can it be done?
Can it done at all?

LCS (Longest Common Subsequence)
the problem of finding the longest subsequence common to all sequences in a set of sequences (often just two)

CHIMPANZEE
HUMAN

Can you solve the problem?

Can you create a process to solve the problem?

What is CS?

AI

How can it be done?
How well can it be done?
Can it done at all?

How quickly can you find a solution?

Is your solution the “best” possible?

What is CS?

AI

How can it be done?
How well can it be done?
Can it done at all?

Is the problem solvable?

How can I tell?

Many problems are uncomputable

AI

How can it be done?
How well can it be done?
Can it done at all?

Can you solve the problem?

Can you create a process to solve the problem?

How quickly can you find a solution?

Is your solution the “best” possible?

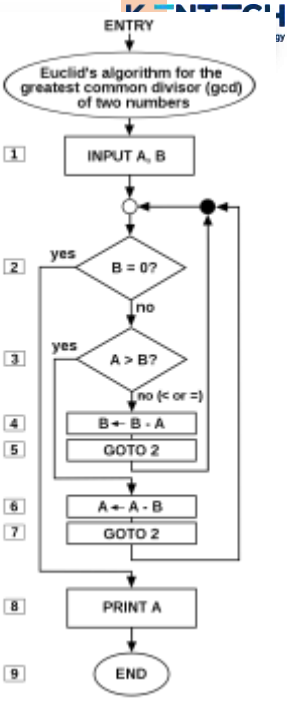
Is the problem solvable?

How can I tell?

Algorithm - LCS

- Given two strings of length m and n
  - ➔ There is O(mn) ***method*** to find the LCS
    - DP (Dynamic programming)

Algorithm : a finite sequence of well-defined instructions, typically used to solve a class of specific problems or to perform a computation



LCS

AI

- Example

ALIEN & ALIEN
INLINE INLINE

- How to find an algorithm
- ➔ Require deep understanding of the problem and *creativeness*

Observations
Let S1 and S2 be two strings of lengths m and n, respectively
Let LCS(S1, S2) be the LCS of S1 and S2

- If both S1 and S2 end with a character 'A'
  ➔ LCS(S1, S2) = LCS(S1-'A', S2-'A') + A
- If S1 and S2 have different last characters (Let them be 'A' and 'B', respectively)
  ➔ LCS(S1, S2) = LCS(S1-'A', S2) or LCS(S1, S2-'B')

Complexity

- (Computational) Complexity
  - **complexity** of an algorithm is the amount of resources (time or memory) required to run it.
- Notation: O(f(N))

Size of the problem

- O(1), O(N), O(NlogN), O(N^2), ….



Logic

- Propositional logic
  - ≈ Boolean algebra
  - Reasoning about Boolean values

George Boole (1815~1864)
was English mathematician, philosopher, and logician. He is best known as the author of *The Laws of Thought* (1854) which contains Boolean algebra.

- First-Order Logic
  - Reasoning about properties of multiple objects
- Higher Order Logic
  - Second/Third… Order Logic

Propositional Logic

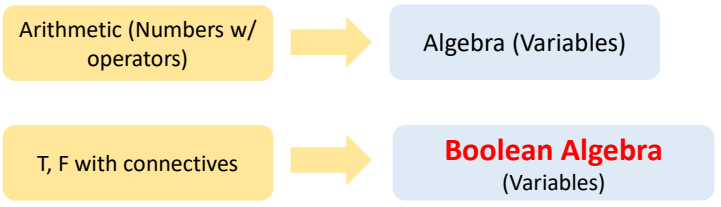
AI

- **Proposition** is a statement that is either True (T) or False (F)
  - English sentences can be propositions
  - "Today is rainy", "I got an A+ in Math."
- Propositional logic
  - A mathematical system for reasoning about propositions and how they relate each other
- Every statement in propositional logic consists of propositional variables and (propositional, logical) connectives that combine propositional variables
  - Each variable represents a proposition, such as "Today is rainy"
  - Connectives encode how propositions are related
    - "Today is rainy" and (∧) "Today is cold"

Boolean Algebra

AI

- Arithmetic deals with numbers and Algebra deals with variables
  - $3^2 + 4^2 = 5^2$
  - $a^2 + b^2 = c^2$
  - We can focus on analysis and manipulation of the structure
- Mathematicians and logicians did the same thing that algebra does for arithmetic



For the analysis of the structure of arguments

Propositional Variables

- Each proposition will be represented by a propositional variable
- Variables are usually represented as lower-case italic letters, such as *p, q, r, s*
  - *p* : "Today is rainy"
- As a proposition, each variable can take one of two values: True or False

Propositional Connectives

- There are seven (Or five) connectives

Connective	Read as	C version	Fancy name
T	True	true	Truth
⊥	False	false	Falsity
¬	not	!	Negation
∧	and	&&	Conjunction
∨	or	||	Disjunction
→	imply		Implication
↔	if and only if		Biconditional

Propositional Connectives

AI

- Logical "NOT": ¬
  - Change T to F and vice versa
    - ¬*p* is F if *p* is T
- Logical "AND": ∧
  - *p* ∧*q* is T if both *p* and *q* are T
- Logical "OR": ∨
  - *p* ∨*q* is T if at least one of *p* or *q* *is* T

P	q	*p* ∧*q*
T	T	T
T	F	F
F	T	F
F	F	F

Truth table

Inclusive/Exclusive OR

AI

- The ∨ connective is an "inclusive OR'
  - True if at least one of two operands is true
- Exclusive OR : ⊕
  - True if and only if exactly one of two operands is True

You can express the "exclusive OR' with other connectives
p ⊕ *q* = (not(p) and q) or (p and not(q)) , not(p iff q)

P	q	*p iff q*
T	T	T
T	F	F
F	T	F
F	F	T

Implication

- p implies q : $p \rightarrow q$
  - If p is True, then q is True
- Construct the truth table for $p \rightarrow q$

P	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

Vacuously True

Vacuous Truth

Implication

- If you work hard, then you will get an A+
- p: you work hard
- q: Get an A+

p	q	$p \rightarrow q$
Work hard	Get an A+	T
Work hard	Get a D	F
Play hard	Get an A+	T
Play hard	Get a D	T

Biconditional

- "p if and only if q": $p \leftrightarrow q$
- $p \leftrightarrow q = (p \rightarrow q) \wedge (q \rightarrow p)$
- Truth table

P	q	$p \leftrightarrow q$
T	T	T
T	F	?
F	T	?
F	F	T

Biconditional

- You work hard if and only if you will get an A+
- p: you work hard
- q: Get an A+

p	q	$p \leftrightarrow q$
Work hard	Get an A+	?
Work hard	Get a D	?
Play hard	Get an A+	?
Play hard	Get a D	?

One interpretation is to think of it as equality: the two propositions must have the same values (both T or both F)

Precedence

- How to parse this statement?

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

- Operator precedence

Connective
¬
∧
∨
→
↔

- All operators are right-associative
- Use parentheses to disambiguate

How to parse this statement?
$(\neg p) \rightarrow ((q \vee r) \rightarrow (p \vee (q \wedge r)))$

De Morgan's Laws

- Use truth table to show

1. $\neg(p \wedge q)$ is equivalent to $\neg p \vee \neg q$
2. $\neg(p \vee q)$ is equivalent to $\neg p \wedge \neg q$

Important Equivalences

- $p \rightarrow q$ is equivalent to $\neg(p \wedge \neg q)$ = not(p) or not(not(q) = not(p) or q
- $\neg(p \rightarrow q)$ is equivalent to not(p imply q) = not (not(p) or q) = p and not(q)

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

  is equivalent to $\neg p \vee \neg\neg q$

  is equivalent to $\neg p \vee q$

If p is False, then $\neg p \vee q$ Is True. If p is True, then q must be True for the whole expression to be True

Cautions

- English sentences and logical connectives sometimes may have different meanings
- p if q is equivalent to q → p

- (Women passengers) ***and*** (Passengers less than 15 years old) will take lifeboats
- p: "It rained"

  q: "there is some sunshine"

  r: "I will see rainbow"
- If it rained, ***but*** there is no sunshine, then I will not see rainbow
- (p and not(q)) implies not(r)

First-Order Logic (FOL)

- Definition: A logical system for reasoning about properties of objects

- Tools that augment the logical connectives from propositional logic
  - ***Predicates***: Describe properties of objects
  - ***Function***: Map objects to another object
  - ***Quantifier***: Allow to reason about multiple objects

(Constant) Symbol, Predicate

Likes (*You, Bacon*) ∧ ***Likes*** (*You, Tomato*) → ***Likes*** (*You, BLT*)

Gets (*You, Optimization, A+*) ∧ ***Gets*** (*You, Probability, A+*) → ***PreparedFor*** (*You, ML*)

Constant Symbols: Refer to objects, not propositions

Predicates: Take objects as arguments and evaluate to T or F

➔ Evaluation results of predicates can be considered as propositions

➔ Connect them with propositional connectives

Reasoning about Objects

- Predicate is to reasoning about objects
- Example
  - *Delicious* (*Macaron*)
  - *BelongsTo* (*Seoul, Korea*)
- Applying a predicate to arguments produces a proposition, which is either True or False
- In FOL, a list of predicates with specifications will be provided
- Specifications
  - Definitions, functionalities
  - Types and numbers of arguments they take

First-Order Sentences

- Sentences in FOL can be constructed from predicates applied to objects

$Delicious(a) \rightarrow IsMacaron(a) \vee IsIceCream(a) \vee IsDonut(a)$

$Smart(You) \wedge WorkLifeBal(You) \leftrightarrow IsHappy(You)$

$x < 2022 \rightarrow x < 2050$ lessThan(x, 2022)

< is a predicate
Binary predicates may written in an ***infix*** notation

Numbers and strings are constant symbols like “You” and “a”

Equality

- FOL is equipped with a special predicate "="
  - Check whether two objects are the same or not
- Equality is a connective like → and ∨

- Example
  - MichaelJordan = BasketballPlayer
  - MichaelJordan = MLResearcher
  - Sun = Moon
- Note: Equality can be applied to objects only

↔ for equality of two propositions

Functions

- Function takes objects as input and produces an object as output
- Examples
  - ***YoungestOf*** (*SNU*) ≠ ***YoungestOf*** (*Kentech*)
  - ***SteepestAscentOf*** (***FavorateTrailOf*** (*You*))
  - ***MinimumOf*** (*x, y, z*)
  - *x* + *y* = *plus(x, y)*

- As with predicates, functions can take in any number of arguments, but always return a single value (Object)
- Functions evaluate to objects, not propositions

Objects & Predicates

- Caution: always keep objects (actual things) and propositions (True or False) separate
- Cannot apply connectives to objects

  Moon ↔ *Sun*
- Cannot apply functions to propositions

  YoungestOf (*IsMarcaron* (*a*) = *IsIceCream* (*a*))

Type Checking Table

	Input	Output
Connectives	Propositions	Single proposition
Predicates	Objects	Single proposition
Functions	Objects	Single Object

Existential Quantifier: ∃

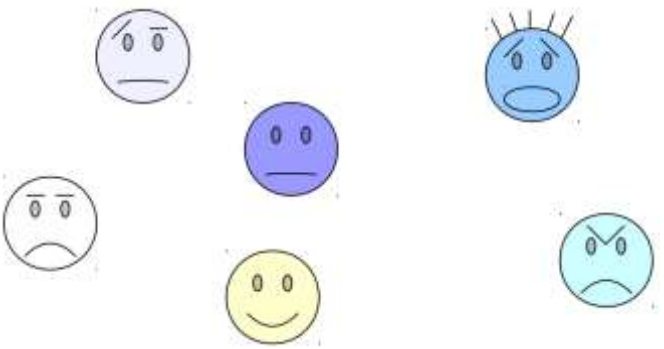
- A statement of the form

  ∃*x*. *some-formula*

  is True if there exist a choice of *x* where *some-formula* is True when that x is plugged into it

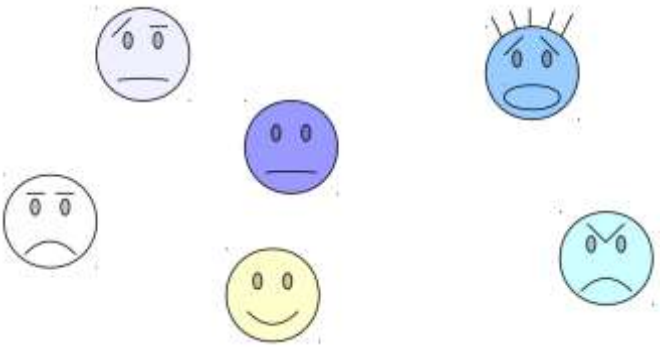
- Examples
  - ∃ *x*.(*Even* (*x*) ∧ *Prime* (*x*))
  - ∃ *x*. (*SmarterThan* (*x, Einstein*) ∧ *LessFamousThan* (*x, Einstein*))
  - (∃ *x*. *WorkHard* (*x*)) → (∃ *y*. *GetA* (*y*))

Existential Quantifier



(∃ *x*. *Smiling* (*x*)) → (∃ *y*. *WearingHat* (*y*))

Existential Quantifier



$(\exists\, x.\, Smiling\ (x)) \rightarrow (\exists\, y.\, WearingHat\ (y))$

How to make the statement to True?

Variables & Quantifiers

- Each quantifier has two parts
  - Variable
  - Statement that is being quantified
- The variable is scoped just to the statement being quantified

$(\exists\, x.\, Loves\ (You, x)) \rightarrow (\exists\, y.\, Loves\ (y, You))$

The variable *x* just lives here

The variable *y* just lives here

Precedence

- Quantifiers have precedence just below ¬
- How to parse the statement

  ∃ *x*. *Even* (*x*) ∧ *Prime* (*x*) ∨ *Odd* (*x*)

  As
  ∃ (*x*. *Even* (*x*) ∧ *Prime* (*x*) ∨ *Odd* (*x*))

  Or as
  ∃ (*x*. *Even* (*x*)) ∧ *Prime* (*x*) ∨ *Odd* (*x*)

- Explicitly put parentheses around the region to quantify

Universal Quantifier: ∀

- A statement of the form

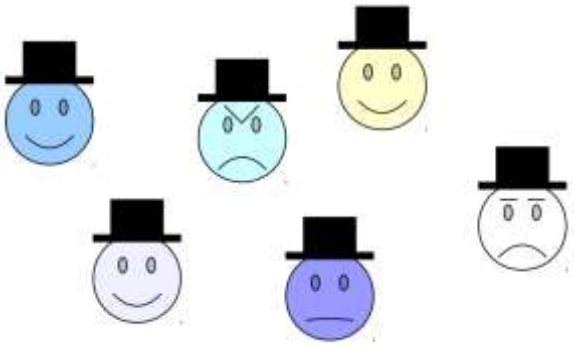
  ∀*x*. *some-formula*

  is True if for every choice of *x*, the statement *some-formula* is True when that x is plugged into it

- Examples
  - ∀*x*. (*IsMacaron* (*x*) → *Delicious* (*x*))
  - For any natural number *n*, *n* is even if and only if n^2 is even

    $\forall n.\ (n \in \mathbb{N} \rightarrow (Even\ (n) \leftrightarrow Even\ (n^2)))$

Universal Quantifier



$(\forall \boldsymbol{x}.\ Smiling\ (x)) \rightarrow (\forall \boldsymbol{y}.\ WearingHat\ (y))$

Universal Quantifier

$\forall \boldsymbol{x}.\ Smiling\ (x)$

Vacuously True

How about $(\exists \boldsymbol{x}.\ Smiling\ (x))$

English & Logic

- FOL is an excellent logic for manipulating definitions and theorems
- Need to take negation?
  - Translate your stamen into FOL, negate it, then translate it back
- Want to prove an implication by contrapositive?
  - Translate the implication into FOL, take the contrapositive, then translate it back
- When translating from English into FOL,

Think of FOL as a mathematical programming language

- The goal is to learn how to combine basic concepts (quantifiers, connectives, etc) together in ways that say what you mean