## **Problem (Language):**

A problem Q is a binary relation on a set I of instances and a set S of solutions Q ⊆ I x S

* every problem can be expressed as deciding whether an object is a member of a set.

[w ∈ L]?

## **There are three types of problems:**

### **1- General problem:**

A general problem G is a total relation:

G ⊆ I × S

One or more solutions can be produced for each problem instance. It also called search problem or optimal solution problem. It can be expressed as deciding whether an object is a member of a set,

[(I, s1, s2,………sn)∈{ (I, s1, s2,………sn): ∀i,i∈I ∃sn,sn∈S(I,sn)∈G}]

inG = lambda i,ss: all((i,s) in G for s in ss)

### **2- Function problem:**

A function problem F is a function F ⊆ I x S

F: I 🡪 S

One solution is produced for each problem instance. It is also called computation problem or optimal value problem.

It can be expressed as deciding whether an object, (i,s) where i ∈ I, s ∈ S is a member of the set F

[(i,s) ∈ F]?

### **3- Decision Problem**

A decision problem Q is a function Q ⊆ I x {no, yes}

Q: I 🡪 {no, yes}

It can be expressed as deciding whether an object is a member of a set:

Y = {i ∈ I: Q[i] == yes}

[i∈Y]?

Since every problem can be expressed as deciding whether an object is a member of a set, every problem can be casted as a decision problem.

### **Encoding:**

An encoding of a set A of abstract objects is a mapping ‘e’ from A to a set L of strings.

e: A 🡪 L

### **Semantics Encoding:**

An encoding of a language A into language B is a mapping of all terms of A into B

e: A 🡪 B

### **Language:**

set of words

### **Alphabet:**

An alphabet is a finite nonempty set of symbols (characters).

Binary alphabet

𝛴 = {0,1}

Lower-case English letters

𝛴 = {a,b,c,...,z}

### **String**

A string (word) is a finite sequence of symbols chosen from some alphabet.

"101"

The length of a string is the number of positions for symbols in the string, |w|.

|"101"|

3

The empty string, ε, is the string with zero occurrences of symbols.

|''|

0

Zk = {string of length k: each of whose symbols is in 𝛴}

𝛴3 = {000, 001, 010, 011, 100, 101, 110, 111}

where 𝛴 = {0,1}

𝛴0 = {ε}

The set of all strings over an alphabet 𝛴 is denoted as 𝛴\*

𝛴\* = 𝛴0 ⋃ 𝛴1 ⋃ 𝛴2 ⋃....

𝛴+ = 𝛴1 ⋃ 𝛴2 ⋃ 𝛴3 ⋃....

### **Language:**

#### **A language is a set of strings, all of which are chosen from 𝛴\***

L is a language over 𝛴

L ⊆ 𝛴\*

𝛴 is a set of alphabets.

𝛴\* is a set of all possible combination of the alphabet (Some of them could be empty or one character).

L is a set of 𝛴\*.

### **Encoding a decision Problem:**

A decision problem can be encoded in a language as a function from the set 𝛴\* to the set {0,1}

Q: 𝛴\* 🡪 {0,1}

0: no

1: yes

The decision problem Q is entirely characterized by those problem instances that produce a 1 (yes) answer:

L = {i ∈ 𝛴\*: Q[i] == 1}

Assign semantics to L, such as a set of prime numbers, even integers, logical expressions, or graphs.

The decision problem Q is: Given a string W ∈ 𝛴\*, decide whether or not w ∈ L:

For w ∈ 𝛴\* [w ∈ L]?

### **The problem Q can be viewed as the language L.**

### **Encoding a decision Problem:**

A decision problem can be encoded in a language as a function from the set 𝛴\* to the set {0,1}

Q: 𝛴\* 🡪 {0,1}

0: no

1: yes

The decision problem Q is entirely characterized by those problem instances that produce a 1 (yes) answer

L = {i ∈ 𝛴\* : Q[i] == 1}

Assign semantics to L, such as a set of prime numbers, even integers, logical expressions, or graphs.

The decision problem Q is: Given a string W ∈ 𝛴\*, decide whether or not w ∈ L

For w ∈ 𝛴\* [w ∈ L]?

### **The problem Q can be viewed as the language L**

**Another Problem of Language:**

Output:

{'0000', '0001', '0010', '0011', '0100', '0101', '0110', '0111', '1000', '1001', '1010', '1011', '1100', '1101', '1110', '1111'}

**Describe how a decision problem is encoded in a language.**

A decision problem can be encoded in a language as a function from the set ∑\* to the set {0, 1}

Q : ∑\* → {0, 1}

The decision problem Q is entirely characterized by those problem instances that problem instances that produce a 1(yes) answer.

L = {i ∈ ∑\* : Q[i] == 1}

Assign semantics to L, such as a set of prime numbers, even integers, logical expressions, or graphs.

The decision problem Q is: Given a string w ∈ ∑\*, decide whether or not w ∈ L

For w ∈ ∑\*, [w ∈ L]?

The problem Q can be view as the language L

def Prob1(z: set, n:int) -> set:

P = [‘ ‘]

For\_in range(n):

P = [p+str(z) for p in P for z in Z]

Return P

I = set(range(16))

### **Encoding:**

An encoding of a set A of abstract objects is a mapping 'e' from A to a set L of strings.

e: A 🡪 L

### **Semantics Encoding:**

An encoding of a language A into language B is a mapping of all terms of A into B

e: A 🡪 B

**Lecture – 2**

## **Formally define sorting problem**

##### **input is a set of tuple (x,y,z) where each of x, y, z is in {'A', 'B', 'C'}**

##### **output is set of tuple of the sorted input**

##### **Formally def I, S, IxS, Q**

## **Computational Model:**

### **Turing Machine:**

A Turing machine is a 7-tuple,

(Q, 𝛴, 𝝘, 𝞭, q0, qaccept, qreject)

Q is the set of states

𝛴 is the input alphabet not containing the special blank symbol \_

𝝘 is the alphabet, where \_ ∈ 𝝘 and 𝛴 ⊆ 𝝘

𝞭 is the transition function

𝞭: Q x 𝝘 🡪 Q x 𝝘 x {L, R}

q0 ∈ Q is the start state

qaccept ∈ Q is the accept halt state

qreject ∈ Q is the reject halt state, where qreject != qaccept

Function 𝞭 is the program of the machine.

For a k multitape Turing machine,

𝞭: Q x 𝝘k 🡪 Q x 𝝘k x {L, R}k

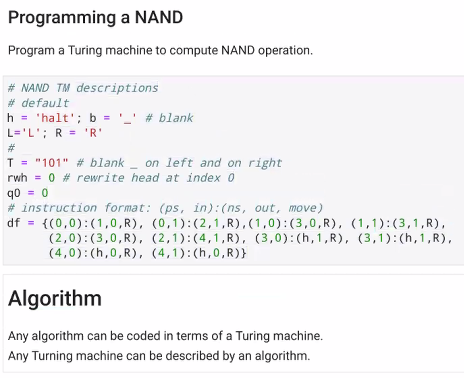
For a non-deterministic Turing machine

𝞭: Q x 𝝘 🡪 P[Q x 𝝘 x {L, R}}

### **Algorithm**

🧲 Any algorithm can be coded in terms of a Turing machine.

🧲 Any Turing machine can be described by an algorithm.



**Universal Turing Machine**

A Universal Turning machine can simulate any Turing machine form the description of that machine.

Let U be three-tape univerival Turing machine

U = (Q, 𝛴, 𝝘, 𝞭, q0, qaccept, qreject)

𝞭: Q x 𝝘3 🡪 Q x 𝝘3 x {L, R}3

Let M be a one-tape Turing machine to be simulated by U

M = (Q', 𝛴', 𝝘', 𝞭', q'0, q'accept, qreject')

𝞭': Q' x 𝝘' 🡪 Q' x 𝝘' x {L', R'}

where 𝝘' ⊆ 𝝘

The definition of 𝞭' is stored on tape p as program code. Initially head p is on the leftmost non-blank.

The q'0 is stored on tape q. initially head q is on the leftmost non-blank.

The input string of M is stored on tape t. Initially head t is on the left most input symbol.

**The 𝞭 of U is defined as:**

**Repeat unit halt:**

* the following macrostep of U to simulate one state transition of M
* instruction format for each state transition M:d(ps, in) = (ns, out, move)

1. read present state and input of M

* head q moves to the leftmost non-blank to read present state of M
* If the present state is qaccept' or q'reject
* then U goes to the corresponding qaccept or qreject
* head t reads the present input of M

1. Search program for the present state and input

* head p moves to leftmost non-blank and search right the pair of present state and input of M until found

1. Copy next state

* head p moves right to read the next state
* head q moves back to leftmost no-bank and writes the next state

1. Copy output

* head p moves right to read the present output
* head t writes the present output

1. move head as specified

* head p moves right to read the movement
* head t moves as specified by the movement

**Computer:**

A computer can be simulated on a Universal Turing machine. A Universal Turing machine can be programmed to run on a computer.

### **Most powerful model:**

Turing machine (and equivalent formalisms) is the most powerful formalism to model and mechanical calculus.

### **Beyond Turing Machine:**

Exploring theoretical computational models beyond the limit of Turing machine and the possibility for hypercomputer.

**Computer:**

A computer can be simulated on a Universal Turing machine. A Universal Turing machine can be programmed to run on a computer.

**Most powerful model:**

Turing machine (and equivalent formalisms) is the most powerful formalism to model and mechanical calculus.

**Beyond Turing Machine:**

Exploring theoretical computational models beyond the limit of Turing machine and the possibility for hypercomputer.

See S:pp.125-147 MG;pp.151-162

**Computability Theory**

**Solvable Problems**

A problem is decidable if it has either a **yes** or a **no** answer

computable, Decidable (or recursive), Semidecidable (recursively enumerable, or acceptable)

**Computable function**

A function f:𝛴\* ⇒ 𝛴\* is computable if there exist Turing machine, on every input *w*, halts with just f(w) on its tape.

The class of computable functions is equivalent to the class of functions defined by

* Recursive function, lambda calculus, or Markov algorithms.

**Recursive functions** are precisely the function that can be computed by Turing machines

**Membership Question**

[x ∈ S]?

The characteristic function 𝑐𝑠: 𝛴\* ⟹ {0,1} of a set *S*, is defined by

𝑐𝑠 (x):= 1 if x ∈ *S* else 0

A set S is **decidable** (or recursive) ⇔ its characteristic function 𝑐𝑠 is computable

A set S is **semidecidable** (or recursive enumerable) ⇔ S = ⦰ or it is the image of a total and computable function gs, that is

S = {y: ∀x ∈ 𝛴\* (y == gs(x))}

When a problem P is represented as a set *Sp*, then

(P is **solvable**) if (*Sp* is recursive)

(P is **partially solvable**) if (*Sp* is recursively enumerable)

**Decidable Languages:**

**∀x ∈ Σ∗ [x ∈ L] ?**

Let L be a regular language and 𝑥∈Σ\*, it is decidable whether x is a member of L

* Regular language is decidable.
* Context free language is decidable.
* Language accepted by pushdown automata is decidable.

**Enumeration**

An enumeration of a set S of abstract objects is a mapping g from S to the set natural number N.

A **godel numbering** is a function g form a countable se *S* to N

With both g and the inverse of g a computable function

g: S ⟹ N

The set g: {TM𝛴} of all the Turing Machines over a given alphabet 𝛴 can be enumerated algorithmically. That is, an algorithmic bijection g: {TM𝛴} ⟹ N can always be stated

g: {TM𝛴} ⟹ N

g: g is bijective

g and g-1 are computable

The number of *all* possible TM based on Z is:

|{TM𝛴}| = N0

* All algorithmically computable function can be algorithmically enumerated.
* All algorithmically recognizable language can be algorithmically enumerated.

Turing Machines can be referred as devices computing function form N ⟹ N

**All computable function f: N ⟹ N can be enumerated**

**Rice's Theorem (solvable case)**

Let *F* be any set of computable functions.

The set *S* = {*x*: *fx* F} is decidable ⇔ *F* = ⦰ or *F* is the set of all computable functions.

**Only trivial property of computable functions are decidable.**

EG: all function for one binary variable

**Rice's Theorem (solvable case)**

Let *F* be any set of computable functions.

The set *S* = {*x*: *fx* F} is decidable ⇔ *F* = ⦰ or *F* is the set of all computable functions.

**Only trivial property of computable functions are decidable.**

EG: all function for one binary variable

EG2: all functions for two binary variables

### **Unsolvable Problems:**

Algorithmic problem **unsolvability may** occur only in the case where **infinitely** many cases (arguments of a function, strings of a language) are to be considered.

* e.g. One machine may not predict the behavior of *any* other machines.

When a problem *P* is represented as a function *f*, then:

*P* is unsolvable if *f* is nonrecursive

When a problem *P* is represented as a set *Sp* then:

*P* is unsolvable if *Sp* in nonrecursively enumerable

### **Uncomputable function**

A vast majority of function on ℕ cannot be computed.

Since all computable function *f*: ℕ ⇒ ℕ can be enumerated, the cardinality of the class of **all computable** functions is at most

*N*0

However, the cardinality of the class *F* of **all definable function** on ℕ, where *F* = {*f*:(*f*: ℕ ⇒ℕ)} is

2*N*0

### **Non-recursively enumerable set**

A set S is **not recursively enumerable** if

((i *S*) ⟹ (*fi* is total)) ∧ (∀*f, f* is total and computable ∃*i* ∈ *S* (*f* == *fi*))

* Total computable function are not recursively enumerable

(*S*tot = {*i*: *fi* is total}) is not recursively enumerable

### **Rice's Theorem (non-solvable case)**

Let F be any set of computable function.

The set *S* = {*x*: *fx* ∈ *F*} is **undecidable** ⇔ ((F ≠ ø) ∧ (F≠{the set of all computable functions}))

* For any non-trivial property of partial function, the question of whether a given algorithm computes a partial function with this property is undecidable.

### **Halting problem is undecidable**

The *halting* problem is a decision problem for determining, when given a description of a program and its initial input, whether **any** program executing on the input will eventually *halt*.

**Halting problem is undecidable**

Proof of sketch: Proof by contradiction. Assume, to the contrary, that there exits a program halt(progStr, inStr), which takes any program encoded as progStr and any input string *inStr*, is able to decide whether or not the program will halt when given the input string.

halt(progStr, inStr) ::= True if prog(inStr) is halt else False

trouble(xSr ::= Ture if halt(xStr, xStr) == False else loopForever

tStr ::= encoding(trouble program)

Dose trouble(tStr) halts?

Assume that trouble(tStr) halts:

Then, halt(tStr, tStr)==False,. That is, tStr program given tStr (trouble(tStr)) does not halts, resulting in a contradiction.

Assume that trouble(sStr) does not halts:

Then, trouble(tStr) program must reach loopForever. Then, halt(tStr, tStr) != False, that is, halt(tSt, tStr) is true. Then, tStr program given tStr (trouble(tStr)) halts, resulting in a contradiction.

Therefore, there does not exist such a halt(progStr, inStr) program.

HaltTM = {<M,w>: (M is a Turing Machine) and (M halts on input w)}

HaltTM is undecidable