Exam 02/19

Q3)

a) $L_1, L_2 \notin REG$. $L_1 \cup L_2$ cannot be in REG

↓

False

Consider $\Sigma = \{a, b\}$

L_1, L_2 are subsets of Σ^*

$L_1 = \{ w \mid \#_a(w) \geq \#_b(w) \}$ → non regular

$L_2 = \{ w \mid \#_a(w) < \#_b(w) \}$ → non regular

$L_1 \cup L_2 = \Sigma^*$ → regular

b) L_1 is CFL. $\forall S \subseteq L_1$ is a CFL

↓

False

$L_1 = \Sigma^*$ → $L_2 = \{ a^n b^n c^n \mid n \geq 1 \} \subseteq L_1$

L_1 CFL

L_2 NON CFL

c) $L_1 \in CFL$, $L_2 \in REG$,

$L_1 \cap L_2$ can be in REG

↓

True

$L_1 = \Sigma^*$ $L_2 = \{ a^i b^j \mid i,j \geq 1 \}$ $L_1 \cap L_2 = L_2$

↳ REG

d) $L_1 \in CFL$, $L_2 \in REG$

$L_1 \cap L_2$ can be outside of REG

↓

True

$L_1 = \{ a^n b^n \mid n \geq 1 \}$ CFL non-REG

$L_2 = \Sigma^*$ REG

$L_1 \cap L_2 = L_1 \rightarrow$ non-REG

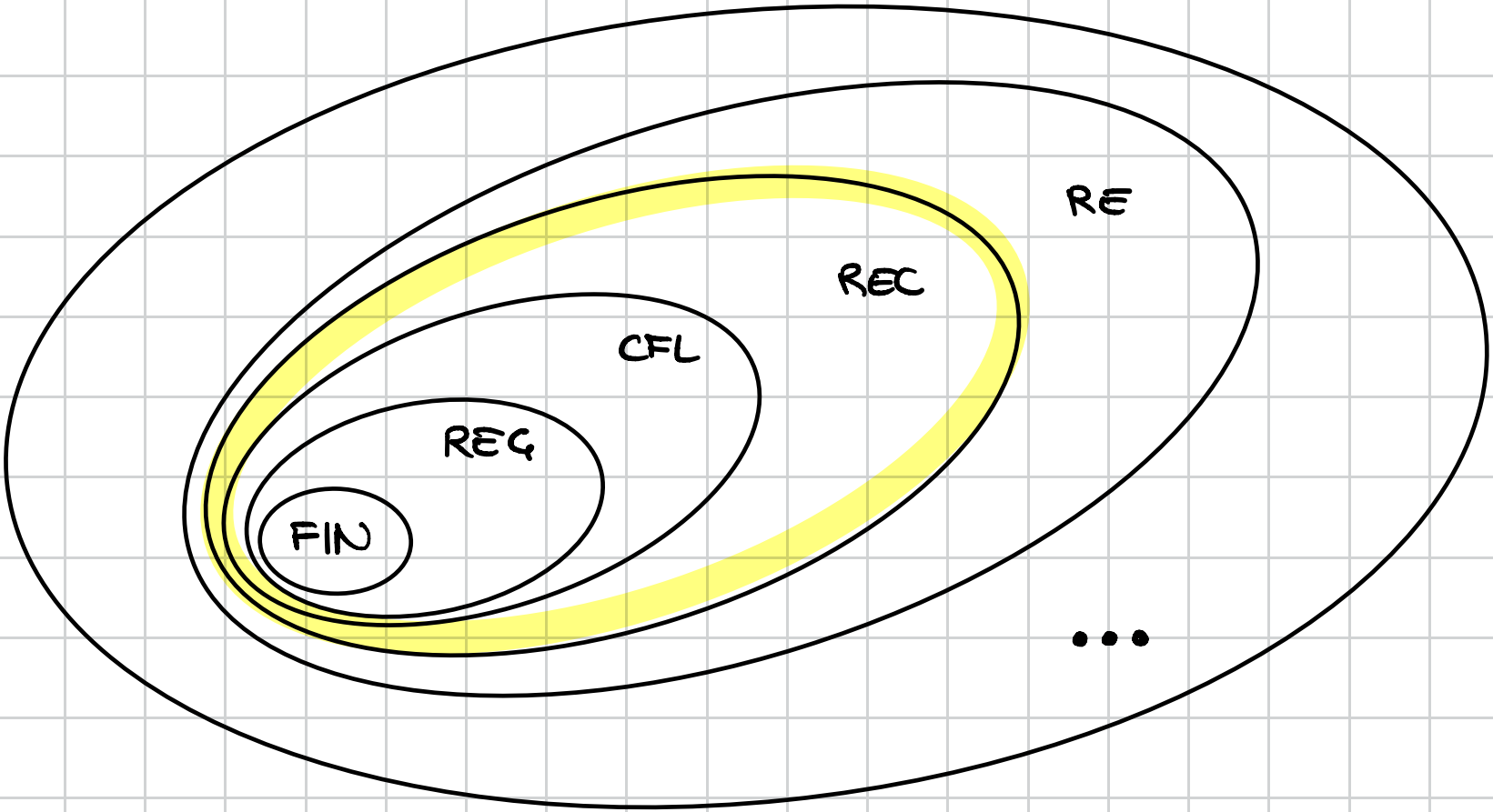
TM and halting

A TM **halts** if it enters a state q with tape symbol X and $\delta(q, X)$ is not defined (there is no next move)

If a TM accepts a string, we can assume that it always halts : just make $\delta(q, X)$ undefined for every final state q

If a TM does not accept, **we can't** assume that it will halt (in a non-final state)

The class of languages accepted by some TM that halts for every input are called **recursive** (REC)


RE
REC
CFL
REG
FIN
...

Decision problems

Let $P(x)$ be a **predicate** expressing some mathematical property of element x

Decision problem associated with P : on input x, decide whether $P(x)$ holds true

everything can be computed as a string

Associated formal language (x viewed as a string) :

$$L_P = \{x \mid P(x) \text{ holds true}\}$$

The decision problem can be reformulated as : Given as input string x, decide whether $x \in L_P$

Example

For natural number x, $P(x)$ is true if x is a prime number. We represent x as a binary string

We define the language of prime numbers

$$L_p = \{10, 11, 101, 111, 1011, \ldots\}$$

Assigned as input the binary string x, decide whether $x \in L_p$

Decision problems

Many mathematical problems are not decision problems, but require instead a computation that constructs an output **result**

Think about search problems, optimization problems, etc.

We can reformulate these problems as decision problems

Example :

- given matrices A, B, construct the matrix $C = A \times B$ → not a decision problem
- associated decision problem : given a triple $\langle A, B, C \rangle$, decide whether $C = A \times B$ → decision problem

Decision problems

The general (non-decision) problem **is no easier** than the associated decision problem

You can solve the decision problem if you have a subroutine for the general problem

Example : Algorithm for decision problem using the algorithm for the general problem as a subroutine (**reduction** technique)

- input $\langle A, B, C \rangle$
- use subroutines on A, B to produce $C' = A \times B$
- if $C' = C$ answer yes, otherwise answer no

If you have enough computational resources to solve the general problem, then you can also solve the decision problem

Recursive and recursively enumerable languages

Recursive language (REC) : the language is accepted by a TM that halts on each input string (in the language or not)

Recursively enumerable language (RE) : the language is accepted by a TM that halts when the string belongs to the language

For strings not in the language, the TM may compute forever

A decision problem P is **decidable** if its encoding L_P (see chapter 1) is a recursive language. Alternatively : if there is a TM M that always halts such that $L(M) = L_P$

Programming techniques for TM

Although the class of TM is very simple, this model has the **same** computational power as a modern computer

We will also see that a TM is able to perform processing on other TMs. This allows us to prove that certain problems are undecidable

Compare with compilers, which take programs as input and produce new programs as output

We present in the following some **notational variants** of the TM that make TM programming easier

Programming techniques for TM

We reformulate the TM definition using

- a finite number of registers with **random access**, which we place inside each state
- a finite number of tape tracks