Theory of Computation fiche

Pierre Colson

Contents

Cours **1**
Automata and Languages . . . 1
Turing Machine . . . 2
Decidable languages . . . 2
Time complexity . . . 3

Example **4**

Markdown version on *github*
Compiled using *pandoc* and *`gpdf` script*
More fiches *here*

Cours

Automata and Languages

- A **deterministic finite automaton (DFA)** is a 5-tuples $(Q, \Sigma, \delta, q_0, F)$, where :
  1) Q is a finite set called the *states*
  2) Σ is a finite set called the *alphabet*
  3) $\delta : Q \times \Sigma \longrightarrow Q$ is the *transition function*
  4) $q_0 \in Q$ is the *start state*
  5) $F \subseteq Q$ is the *set of accept states*
- A **nondeterministic finite automaton (NFA)** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where :
  1) Q is a finite set called the *states*
  2) Σ is a finite set called the *alphabet*
  3) $\delta : Q \times \Sigma_\epsilon \longrightarrow P(Q)$ is the *transition function*
  4) $q_0 \in Q$ is the *start state*
  5) $F \subseteq Q$ is the *set of accept states*
- Deterministic and nondeterministic finite automata recognize the same class of languages. Say that two machines are **equivalent** if they recognize the same language.
- A language is called a **regular language** if some finite automaton recognizes it.
- Every nondeterministic finite automaton has an equivalent deterministic finite automaton.
- Let A and B be languages. We define the regular operations **union**, **intersection**, **concatenation** and the **star** as follows:
  1) **Union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
  2) **intersection** $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$
  3) **Concatenation**: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$

4) **Star**: $A^* = \{x_1x_2...x_k \mid k \geq 0 \text{ and each } x_i \in A\}$

- The class of regular language is closed under the union operation, intersection operation, concatenation operation, start operation.
- The **complement** ($\bar{L} = \{w \in \Sigma^* : w \text{ is not in } L\}$) of a regular language is also regular.
- Say that R is a **regualr expression** is R is:

  1) a for some a in the alphabet Σ
  2) ϵ
  3) $\emptyset$
  4) $\bar{R_1}$ where R_1 is aregular language
  5) $(R_1 \cup R_2)$ where R_1 and R_2 are regular expressions
  6) $(R_1 \cap R_2)$ where R_1 and R_2 are regular expressions
  7) $(R_1 \circ R_2)$ where R_1 and R_2 are regular expressions
  8) (R_1^*) where R_1 is a regular expression

- A language is relular if and only if some regular expressions describes it.
- **Pumping Lemma**: If A is a regular language, then there is a number p (the pumping lemma) where if s is any string in A of length a least p, then the s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

  1) for each $i \geq 0, xy^iz \in A$
  2) $|y| > 0$, and
  3) $|xy| \leq p$

Turing Machine

- A **Turing Machine** is a 7-tuples, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite set and :

  1) Q is the set of *states*,
  2) Σ is the input *alphabet* not containing the blank symbol $\_$
  3) Γ is the tape *alphabet* where $\_ \in \Gamma$ and $\Sigma \subset \Gamma$
  4) $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ is the *transition function*,
  5) $q_0 \in Q$ is the *start state*,
  6) $q_{accept} \in Q$ is the *accept state*, and
  7) $q_{reject} \in Q$ is the *reject state*, where $q_{reject} \neq q_{accept}$.

- Call a language **Turing reconizable** or simply **reconizable** if some Turing machine recognizes it. A Turing machine M recognizes a language $L \subseteq \Sigma^*$ if and only if al inputs $w \in \Sigma^*$:

  1) if $w \in L$ then M accepts w and
  2) if $w \notin L$ then M either rejects w or never halts

- Call a language **Turing decidable** or simply **decidable** if some Turing machine decides it. A Turing machine M decides a language $L \subseteq \Sigma^*$ if and only if al inputs $w \in \Sigma^*$:

  1) M halts on w, and
  2) M accepts w if and only if $w \in L$

- Every multitape Turing machine has an equivalent single-tape Turing machine
- Every nondeterministic Turing machine has an equivalent deterministic Turing machine

Decidable languages

- $A_{DFA} = \{\langle B, w\rangle \mid B \text{ is a DFA that accpets input string } w\}$. A_{DFA} is a decidable language.
- $A_{NFA} = \{\langle B, w\rangle \mid B \text{ is and NFA that accepts input string } w\}$. A_{NFA} is a decidable language.

- $EQ_{DFA} = \{\langle A, B\rangle \mid A, B \text{ are DFAs s.t. } L(A) = L(B)\}$. EQ_{DFA} is decidable.
- $E_{DFA} = \{\langle D\rangle : L(D) =\}$. E_{DFA} is decidable. (not verify yet)
- $L_{DIAG} = \{\langle M_i\rangle : M_i \text{ doesn't accept } \langle M_i\rangle\}$. L_{DIAG} is undecidable
- $A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$. A_{TM} is undecidable.
- $REG_{TM} = \{\langle N\rangle \mid L(N) \text{ is regular}\}$. REG_{TM} is undecidable.
- $EQ_{TM} = \{\langle M_1, M_2\rangle \mid M_1, M_2 \text{ are TMs s.t. } L(M_1) = L(M_2)\}$. EQ_{TM} is undecidable.
- $E_{TM} = \{\langle M\rangle : L(M) =\}$. E_{TM} is undecidable. (not verify yet)
- **Halting problem**, let $HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$. $HALT_{TM}$ is undecidable.
- A language is **decidable** if and only if it is Turing reconizable and co Turing reconizable. In other words, a language is decidable exactly when both it and its complement are Turing reconizable.
- A function $f : \Gamma^* \longrightarrow \Gamma^*$ is a **computable function** is some Turing machien M, on every input w, halts with just $f(w)$ on its tape.
- Language A is **mapping reducible** to language B, written $A \leq_m B$, if ther is a computable function $f : \Gamma^* \longrightarrow \Gamma^*$, where for every w, $w \in A \iff f(w) \in B$. The function f is called the **reduction** from A to B.
- If $A \leq_m B$ and B is decidable, then A is decidable (same for reconizable)
- If $A \leq_m B$ and A is undecidable, then B is undicidable (same for reconizable)
- If $B \leq_m \bar{B}$ then $\bar{B} \leq_m B$

Time complexity

- P is the calss of languages that are *decidable* in *polynomial time* on a *deterministic* Turing machine :

$$P = \bigcup_{k=1}^{\infty} TIME(n^k)$$

- A **verifier** for a language A is an algorithm V, where:

$$A = \{w \mid V \text{ accepts } \langle w, c\rangle \text{ for some string } c\}$$

- NP is the class on languages that have a *polynomial time verifiers*
- **NP-complete** A language B is NP-complete if it satifies two conditions :
  1) B is in NP, and
  2) every A in NP is polynomial time reductible to B
- **SAT** Conjunctive Normal Form (CNF) Formula :

$$\varphi = (\bar{X} \vee \bar{y} \vee z_0) \wedge (x \vee \bar{y} \vee z_2) \wedge (x \vee y \vee z_3)$$

$$SAT = \{\langle \phi\rangle \mid \phi \text{ is a satisfiable Booolean formula}\}$$

  SAT is $NP - complete$

- **CLIQUE** The CLIQUE problem is to determine whether a graph contains a clique (fully connected subgraph) of a specified size. Let :

$$CLIQUE = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a} k-\text{clique}\}$$

  CLIQUE is $NP-complete$

- **SUBSET-SUM** We want to determine whether the collection contains a subcollection that adds up to t. Thus :

$$SUBSET\text{−}SUM = \{\langle S, t\rangle \mid S = \{x_1, \dots, x_k\} \text{ and for some } \{y_1, \dots, y_l\} \subseteq \{x_1, \dots, x_k\} \text{ we have } \sum y_i = t\}$$

  SUBSET-SUM is $NP-complete$

- **3SAT** *3cnf-formula*: all the clauses have three literals. Let :

$$3SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable 3cnf-formula}\}$$

  3SAT is polynomial time reductible to CLIQUE. Thus it is $NP-complete$

- **INDESET**

$$INDSET = \{(G, k) : G \text{ has an independent subset of size } k\}$$

  INDSET is $NP-complete$

- **VERTEX-COVER** If G is an undirected graph, a *vertx cover* of G is a subset of the nodes where every edges of G touches one of thoses nodes. The vertex cover problem asks whether a graph contains a vertex cover of a specified size.

$$VERTEX-COVER = \{\langle G, k\rangle \mid G \text{ is an undirected graph taht has a} k\text{-node vertex cover}\}$$

  VERTEX-COVER is $NP-complete$

- **SET-COVER** Let $U = \{1, \dots, N\}$ and $F = \{T_1, \dots, T_m\}$ be a family of subsets $\forall i, T_i \subseteq U$. A subset $\{T_{i_1}, \dots, T_{i_k}\} \subseteq F$ is called a *set cover* of size k if $\bigcup_{j=1}^{k} T_{i_j} = U$
- If $A \leq_p B$ and $B \in P$, then $A \in P$
- If B is $NP-complete$ and $B \leq_p C$ for C in NP, then C is $NP-complete$

Example

- **Pumping Lemma** : Let B be the languages $\{0^n1^n \mid n \geq 0\}$. We use the pumping lemma to prove that B is not regular. The proof is by contradiction. Assume to the contrary that B is regular. Let p be the pumping length given by the pumping lemma. Choose s to be the string 0^p1^p. Because s is a member of B and s has length more than p, the pumping lemma guarantees that s can be split into three pieces, $s = xyz$ where $|xy| \leq p, |y| \geq 1$ and for any $i \geq 0$ the string xy^iz is in B. We consider three cases to show that this result is impossible:
  1) The string y consists only of 0s. In this case, the string $xyyz$ has more 0s than 1s and so is not a member of B, violating condition 1 of the pumping lemma. This case is a contradiction.
  2) The string y consists only of 1s. This case also gives a contradiction.
  3) The string y consists of both 0s and 1s. In this case, the string $xyyz$ may have the same number of 0s and 1s, but they will be out of order with some 1s before 0s. Hence it is not a member of B, which is a contradiction.