Appunti di Automi e Linguaggi Formali

17 Agosto 2023

Rosso Carlo

Contents

1 Introduzione

La teoria della computazione si pone una domanda: che cos'è un computer? Idealiziamo un computer come un modello computazionale. Un modello computazionale può essere preciso in qualche aspetto, ma non in altri. Per questo motivo studiamo diversi modelli computazionali, in base alle caratteristiche che ci interessano. Cominciamo con il modello computazionale più semplice: l'automa a stati finiti (DFA, Deterministic Finite Automaton).

2 Automi a stati finiti

2.1 Definizione

Def. 2.1 (Automa a stati finiti) *Un automa finito è una quintupla* $(Q, \Sigma, \delta, q_0, F)$ *dove:*

1. Q *è un insieme finito chiamato insieme degli stati;*
2. Σ *è un insieme finito chiamato alfabeto;*
3. $\delta : Q \times \Sigma \to Q$ *è la funzione di transizione;*
4. $q_0 \in Q$ *è lo stato iniziale;*
5. $F \subseteq Q$ *è l'insieme degli stati finali.*

Se A è l'insieme delle stringhe accetate da un DFA M, allora A è il linguaggio di M, e lo indichiamo con $L(M)$. E diciamo che M accetta A.

Def. 2.2 (Computazione) *Sia* $M = (Q, \Sigma, \delta, q_0, F)$ *un DFA e* $w = w_1 w_2 \dots w_n$ *una stringa su* Σ. *Diciamo che* M *accetta* w *se esiste una sequenza di stati (computazione)* $r_0, r_1, \dots, r_n$ *in* Q *tale che:*

1. $r_0 = q_0$*;*
2. $r_{i+1} = \delta(r_i, w_{i+1})$ *per* $i = 0, 1, \dots, n-1$*;*
3. $r_n \in F$.

Def. 2.3 (Linguaggio regolare) *Un linguaggio si dice regolare se qualche DFA lo accetta.*

2.2 Operazioni sui linguaggi regolari

Def. 2.4 (Unione) *Siano* A *e* B *due linguaggi.*

$A \cup B = \{x \mid x \in A \text{ oppure } x \in B\}$.

Def. 2.5 (Concatenazione) *Siano* A *e* B *due linguaggi.*

$A \circ B = \{xy \mid x \in A \text{ e } y \in B\}$.

Def. 2.6 (Star) *Sia* A *un linguaggio.*

$A^* = \{x_1 x_2 \dots x_k \mid k \geq 0 \text{ e } x_i \in A \text{ per } i = 1, 2, \dots, k\}$.

2.3 Automa a stati finiti non deterministico

Definiamo ϵ come la stringa vuota e $\mathcal{P}(A)$ come l'insieme delle parti di A.

Def. 2.7 (Automa a stati finiti non deterministico) *Un automa a stati finiti non deterministico (NFA, Nondeterministic Finite Automaton) è una quintupla $(Q, \Sigma, \delta, q_0, F)$ dove:*

1. *Q è un insieme finito chiamato insieme degli stati;*
2. *Σ è un insieme finito chiamato alfabeto;*
3. *$\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$ è la funzione di transizione;*
4. *$q_0 \in Q$ è lo stato iniziale;*
5. *$F \subseteq Q$ è l'insieme degli stati finali.*

NB: *δ è una funzione che mappa un elemento di Q e un elemento di $\Sigma \cup \{\epsilon\}$ in un sottoinsieme di Q, quindi anche $\emptyset$.*

Def. 2.8 (Computazione) *Sia $M = (Q, \Sigma, \delta, q_0, F)$ un DFA e $w = w_1 w_2 \dots w_n$ una stringa su Σ. Diciamo che M accetta w se esiste una sequenza di stati (computazione) $r_0, r_1, \dots, r_n$ in Q tale che:*

1. *$r_0 = q_0$;*
2. *$r_{i+1} \in \delta(r_i, w_{i+1})$ per $i = 0, 1, \dots, n-1$;*
3. *$r_n \in F$.*

Def. 2.9 (Equivalenza tra automi) *Diciamo che due automi M_1 e M_2 sono equivalenti se $L(M_1) = L(M_2)$.*

Teorema 2.1 (Equivalenza tra DFA e NFA) *Ogni automa a stati finiti non deterministico ha un equivalente deterministico.*

2.4 Espressioni regolari

Def. 2.10 (Espressione regolare) *R è un espressione regolare se vale una delle seguenti:*

- *a per qualche $a \in \Sigma$;*
- *ϵ;*
- *$\emptyset$;*
- *$(R_1 \cup R_2)$ dove R_1 e R_2 sono espressioni regolari;*
- *$(R_1 \circ R_2)$ dove R_1 e R_2 sono espressioni regolari;*
- *(R_1^*) dove R_1 è un'espressione regolare.*

Teorema 2.2 *Un linguaggio è regolare se e solo se qualche espressione regolare lo genera.*

2.5 Pumping Lemma

Teorema 2.3 (Pumping Lemma) *Sia A un linguaggio regolare. Allora esiste un numero p (detto pumping length) tale che per ogni stringa $s \in A$ con $|s| \geq p$ esiste una suddivisione $s = xyz$ tale che:*

1. $|y| > 0$;
2. $|xy| \leq p$;
3. *per ogni* $i \geq 0$, $xy^iz \in A$.

3 Grammatica context-free

Def. 3.1 (Grammatica context-free) *Una grammatica context-free (CFG, Context-Free Grammar) è una quadrupla (V, Σ, R, S) dove:*

1. V *è un insieme finito chiamato insieme delle variabili;*
2. Σ *è un insieme finito, disgiunto da V, chiamato insieme dei terminali;*
3. R *è un insieme finito di regole, ciascuna delle quali è una coppia (v, w) dove $v \in V$ e $w \in (V \cup \Sigma)^*$;*
4. $S \in V$ *è la variabile iniziale.*

È intuitivo, ma vediamo come costruire una grammatica CF a partire da un DFA:

1. $\forall q_i \in Q \ \exists r_i \in R$;
2. $\Sigma_{CFG} = \Sigma_{DFA}$;
3. $r_i \to ar_j \in R$ per ogni $q_j = \delta(q_i, a)$, tale che $a \in \Sigma_{DFA}$;
4. $r_i \to \epsilon \in R$ per ogni $q_i \in F$;
5. $S = q_0$.

Def. 3.2 (Ambiguità) *Una grammatica è ambigua se esiste una stringa w tale che w ha due alberi di derivazione distinti. In questo caso si dice che w si deriva in modo ambiguo.*

3.1 Forma normale

Def. 3.3 (Forma normale di Chomsky) *Una grammatica è in forma normale di Chomsky se tutte le sue regole sono della forma:*

- $A \to BC$;
- $A \to a$.

dove $A, B, C \in V$ e $a \in \Sigma$ e $B, C \neq S$.

Teorema 3.1 (Forma normale di Chomsky) *Ogni linguaggio context-free è generato da una grammatica context-free in forma normale di Chomsky.*

3.2 Automa a pila

Def. 3.4 (Automa a pila) *Un automa a pila (PDA, pushdown automaton) è una 6-upla* $(Q, \Sigma, \Gamma, \delta, q_0, F)$*, dove* $Q, \Sigma, \Gamma eF$ *sono insiemi finiti e:*

- Q *è l'insieme degli stati;*
- Σ *è l'alfabeto di input;*
- Γ *è l'alfabeto dello stack;*
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \epsilon) \to \mathcal{P}(Q \times (\Gamma \cup \{\epsilon\}))$ *è la funzione di transizione;*
- $q_0 \in Q$ *è lo stato iniziale;*
- $F \subseteq Q$ *è l'insieme degli stati finali.*

Def. 3.5 (Computazione) *Un PDA* $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ *accetta l'input* w *se* w *può essere scritta come* $w = w_1 w_2 \dots w_n$ *dove* $w_i \in \Sigma$*, la sequenza di stati* $r_0, r_1, \dots, r_n \in Q$ *e le stringhe* $s_0, s_1, \dots, s_n \in \Gamma^*$ *esistono tali che valgono le seguenti condizioni:*

1. $r_0 = q_0$ *e* $s_0 = \epsilon$*;*
2. $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$ *e* $s_{i+1} = bt$ *e* $s_i = at$ *per qualche* $t \in \Gamma^*$ *e* $a, b \in \Gamma \cup \{\epsilon\}$*;*
3. $r_n \in F$*.*

Teorema 3.2 (Equivalenza tra PDA e CFG) *Un linguaggio è context-free se e solo se è accettato da un PDA.*

Dal precedente teorema otteniamo il corollario: ogni linguaggio regolare è context-free.

3.3 Pumping Lemma

Teorema 3.3 (Pumping Lemma) *Sia* A *un linguaggio context-free. Allora esiste un numero* p *(detto pumping length) tale che per ogni stringa* $s \in A$ *con* $|s| \geq p$ *esiste una suddivisione* $s = uvxyz$ *tale che:*

1. $|vy| > 0$*;*
2. $|vxy| \leq p$*;*
3. *per ogni* $i \geq 0$*,* $uv^i xy^i z \in A$*.*

4 Macchina di Turing

Def. 4.1 (Macchina di Turing) *Una macchina di Turin (TM, Turing Machine) è una 7-upla* $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$*, dove* Q, Σ, Γ *sono insiemi finiti e valgono le seguenti condizioni:*

1. Q *è l'insieme degli stati;*
2. Σ *è l'alfabeto di input, non contiene il carattere speciale* $\sqcup$*;*
3. Γ *è l'alfabeto del nastro, tale che* $\Sigma \subseteq \Gamma$ *e* $\sqcup \in \Gamma$*;*
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ *è la funzione di transizione;*
5. $q_0 \in Q$ *è lo stato iniziale;*

6. $q_{acc} \in Q$ *è lo stato di accettazione;*

7. $q_{rej} \in Q$ *è lo stato di rifiuto, dove* $q_{rej} \neq q_{acc}$.

Def. 4.2 (Configurazione) *Una configurazione di una TM è una tripla* (q, x, y) *dove* $q \in Q$ *è lo stato corrente,* $x \in \Gamma^*$ *è la stringa sul nastro che precede la posizione della testina e* $y \in \Gamma^*$ *è la stringa non vuota che segue* x*, la testina si trova nel primo carattere di y. Un altro modo per rappresentare una configurazione è* xqy.

NB: la configurazione iniziale è $q_0 w$ dove w è la stringa di input. In ogni configurazione di accetazione, lo stato corrente è q_{acc}. In ogni configurazione di rifiuto, lo stato corrente è q_{rej}. Le configurazioni di accettazione e di rifiuto sono dette configurazioni finali e bloccano la TM.

Def. 4.3 (Computazione) *Si diche che una TM* M *accetta l'input* w *se esiste una sequenza di configurazioni* $C_1, C_2, \ldots, C_k$ *tale che:*

1. $C_1 = q_0 w$*;*
2. C_i *genera* C_{i+1} *per* $i = 1, \ldots, k-1$*;*
3. $C_k = q_{acc}$*;*

L'insieme di strighe che M accetta è detto linguaggio di M e si indica con $L(M)$.

Def. 4.4 (Turing riconoscibile) *Un linguaggio è Turing riconoscibile se è accettato da una TM.*

Quando facciamo partire una TM su un input, ci sono tre possibili risultati:

1. la TM si blocca in uno stato di accettazione;
2. la TM si blocca in uno stato di rifiuto;
3. la TM entra in un loop infinito.

Def. 4.5 (Decisore) *Una TM è un decisore se si blocca in uno stato di accettazione o di rifiuto per ogni input.*

Def. 4.6 (Turing decidibile) *Un linguaggio è Turing decidibile se è accettato da un decisore.*

5 Decidibilità

Teorema 5.1 (A_{DFA}**)** *Sia* $A_{DFA} = \{\langle B, w\rangle \mid B$ *è un DFA che accetta* $w\}$*. Allora* A_{DFA} *è decidibile.*

NB: Si simula la computazione del DFA su w e si controlla se si controlla in quale stato si arriva alla fine della computazione.

Teorema 5.2 (A_{NFA}**)** *Sia* $A_{NFA} = \{\langle B, w\rangle \mid B$ *è un NFA che accetta* $w\}$*. Allora* A_{NFA} *è decidibile.*

NB: Si converte un NFA in DFA e si applica il teorema precedente.

Teorema 5.3 (A_{REX}**)** *Sia* $A_{REX} = \{\langle R, w\rangle \mid R$ *è una regex che genera* $w\}$*. Allora* A_{REX} *è decidibile.*

NB: Si converte la regex in un DFA e si applica il teorema precedente.

Teorema 5.4 (E_{DFA}**)** *Sia* $E_{DFA} = \{\langle A\rangle \mid A$ *è un DFA tale che* $L(A) = \emptyset\}$*. Allora* E_{DFA} *è decidibile.*

Dimostrazione 5.1 *Costruiamo la TM T che decide E_{DFA}:*

$T =$ *"Sulla stringa di input $\langle A \rangle$, dove A è un DFA:*

1. *Segna lo stato iniziale di A;*
2. *Ripeti finché nessuno stato nuovo viene segnato:*
   (a) *Segna ogni stato raggiungibile da uno stato segnato;*
3. *Se uno stato finale non è segnato, allora accetta, altrimenti rifiuta."*

Teorema 5.5 (EQ_{DFA}) *Sia $EQ_{DFA} = \{\langle A, B \rangle \mid A \text{ e } B \text{ sono DFA e } L(A) = L(B)\}$. Allora EQ_{DFA} è decidibile.*

NB: Se $L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B)) = \emptyset$ allora $L(A) = L(B)$.

Teorema 5.6 (A_{CFG}) *Sia $A_{CFG} = \{\langle G, w \rangle \mid G \text{ è una CFG che genera } w\}$. Allora A_{CFG} è decidibile.*

Teorema 5.7 (E_{CFG}) *Sia $E_{CFG} = \{\langle G \rangle \mid G \text{ è una CFG tale che } L(G) = \emptyset\}$. Allora E_{CFG} è decidibile.*

Dimostrazione 5.2 *Costruiamo la TM T che decide E_{CFG}:*

$T =$ *"Sulla stringa di input $\langle G \rangle$, dove G è una CFG:*

1. *Segna tytti i simboli terminali di G;*
2. *Ripeti finché nessuna nuova variabile viene segnata:*
   (a) *Segna ogni variabile V tale che G abbia una regola del tipo $V \to U_1 U_2 \dots U_k$ tali che ogni U_i è già stato seganto;*
3. *Se la variabile di partenza di G non è segnata, allora accetta, altrimenti rifiuta."*

5.1 Problemi indecidibili

Teorema 5.8 (Halting problem) *Sia $A_{TM} = \{\langle M, w \rangle \mid M \text{ è una TM che accetta } w\}$. Allora A_{TM} è indecidibile.*

Teorema 5.9 *Un liguaggio è decidibile se e solo se è Turing riconoscibile e co-Turing riconoscibile.*

6 Riducibilità

Teorema 6.1 ($HALT_{TM}$) *Sia $HALT_{TM} = \{\langle M, w \rangle \mid M \text{ è una TM che si ferma su } w\}$. Allora $HALT_{TM}$ è indecidibile.*

Dimostrazione 6.1 *Si suppone che $HALT_{TM}$ sia decidibile e si costruisce una TM T che decide A_{TM}, tale che R è un desisore di $HALT_TM$:*
$T =$ *"Sulla stringa di input $\langle M, w \rangle$, dove M è una TM e w è una stringa:*

1. *simula R sull'input $\langle M, w \rangle$;*
2. *se R rifiuta, rifiuta;*

3. *simula M sull'input w fino a che non si blocca;*

4. *se M accetta, accetta, altrimenti rifiuta;"*

Dato che R è un decisore di $HALT_{TM}$, se R rifiuta, allora M non si blocca su w, quindi M non accetta w. Se R accetta, allora M si blocca su w, quindi la simulazione di M su w termina, rendendo A_{TM} un decisore. Questo è un assurdo, quindi R non può esistere.

Teorema 6.2 (E_{TM}) *Sia $E_{TM} = \{\langle M\rangle \mid M$ è una TM tale che $L(M) = \emptyset\}$. Allora E_{TM} è indecidibile.*

Dimostrazione 6.2 *Si suppone che E_{TM} sia decidibile e si costruisce una TM T che decide A_{TM}. Sia R è un desisore di E_TM:*
$T =$ "Sulla stringa di input $\langle M, w\rangle$, dove M è una TM e w è una stringa:

1. *usa la descrizione di M e w per costuire M_2 tale che:*
   $M_2 =$ "Sulla stringa di input x:

   (a) *se $x \neq w$ rifiuta;*
   (b) *altrimenti simula M su w, se M accetta, accetta, se M rifiuta, rifiuta;"*

2. *simula R sull'input M_2;*

3. *se R rifiuta, accetta, se R accetta, rifiuta; "*

Dato che M_2 arriva in uno stato di accettazione solo se M accetta w. Questo vuole dire che $L(M_2) = \emptyset$ se M non accetta w ed $L(M_2) = \{w\}$ se M accetta w. Dunque se R, il decisore di E_TM, accetta, vuol dire che M non accetta w. Ma allora T è un decisore di A_TM, questo è assurdo, quindi R non può essere un decisore di E_TM, allora E_TM è indecidibile.

Teorema 6.3 ($REGULAR_{TM}$) *Sia $REGULAR_{TM} = \{\langle M\rangle \mid M$ è una TM tale che $L(M)$ è un linguaggio regolare$\}$. Allora $REGULAR_{TM}$ è indecidibile.*

Dimostrazione 6.3 *Si supponge che $REGULAR_{TM}$ sia decidibile e si costruisce una TM T tale che decide A_{TM}. Sia R un decisore di $REGULAR_{TM}$:*

$T =$ "Sulla stringa di input $\langle M, w\rangle$, dove M è una TM e w è una stringa:

1. *usa M e w per costruire la TM M_2:*

   $M_2 =$ "Sulla stringa di input x:

   (a) *se $x \in 0^n1^n$ accetta;*
   (b) *altrimenti simula M su w, se M accetta, accetta, altrimenti rifiuta;"*

2. *simula R sull'input $\langle M\rangle$;*

3. *se R accetta, accetta; se R rifiuta, rifiuta;"*

Se M accetta w allora $L(M_2) = \Sigma^*$ ed è un liguaggio regolare. Se M rifiuta w allora $L(M_2)$ non è regolare, quindi R rifiuta. T è un decisore di A_TM, questo è assurdo, ma allora R non può essere un decisore di $REGULAR_{TM}$. Dunque $REGULAR_{TM}$ è indecidibile.

Teorema 6.4 (EQ_TM) *Sia $EQ_{TM} = \{\langle M_1, M_2\rangle \mid M_1$ e M_2 sono TM tali che $L(M_1) = L(M_2)\}$. Allora EQ_{TM} è indecidibile.*

Dimostrazione 6.4 *Si supponga che EQ_{TM} sia decidibile e si costruisce una TM T che decide E_{TM}. Sia R un decisore di EQ_{TM}:*

$T =$ *"Sulla stringa di input $\langle M \rangle$, dove M è una TM:*

1. *Simula R sull'input $\langle M, M_0 \rangle$, dove M_0 è una TM che rifiuta ogni stringa;*
2. *se R accetta, accetta; se R rifiuta, rifiuta;"*

Se $L(M) = \emptyset$, allora R accetta, perché $L(M) = L(M_0)$. Altrimenti $L(M) \neq L(M_0) = \emptyset$, quindi R rifiuta. T è un decisore di E_{TM}, questo è assurdo, ma allora R non può essere un decisore di EQ_{TM}. Dunque EQ_{TM} è indecidibile.

7 Complessità temporale

Def. 7.1 (Time) *Sia $t : \mathbb{N} \to \mathbb{N}$ una funzione. Defniniamo la classe di complessità temporale $TIME(t(n))$*

$$TIME(t(n)) = \{L \mid \text{ esiste una TM } M \text{ che decide } L \text{ in tempo } O(t(n))\} \tag{1}$$

Teorema 7.1 (Multitape TM) *Una multitape TM che lavora in tempo $t(n)$ può essere simulata da una singletape TM che lavora in tempo $O(t^2(n))$.*

Teorema 7.2 (TM non deterministica) *Una TM non deterministica che lavora in tempo $t(n)$ può essere simulata da una TM deterministica che lavora in tempo $2^{O(t(n))}$.*

Def. 7.2 (P (Polynomial time)) *P è la classe dei linguaggi che sono decidibili in tempo polinomiale da una TM deterministica. Ovvero:*

$$P = \bigcup_k TIME(n^k) \tag{2}$$

7.1 NP

Def. 7.3 (Verificatore) *Un verificatore per un linguaggio L è una TM V tale che:*

$$L = \{w \mid \text{ esiste una stringa } c \text{ tale che } V \text{ accetta } \langle w, c \rangle\}; \tag{3}$$

c è chiamata certificato.

In sostanza un verificatore è una TM che che verifica che una stringa w sia nel linguaggio L, avendo a disposizione informazioni aggiuntive per effettuare la verifica più velocemente.

Def. 7.4 (NP (Nondeterministic Polynomial time)) *NP è la classe dei linguaggi che hanno un verificatore che lavora in tempo polinomiale. Ovvero:*

Def. 7.5 (NTIME)

$$NTIME(t(n)) = \{L \mid \text{ esiste una TM non deterministica che decide } L \text{ in tempo } O(t(n))\} \tag{4}$$

Dunque: $NP = \bigcup_k NTIME(n^k)$.

7.2 NP-completo

Def. 7.6 (Funzione computabile in tempo polinomiale) *Una funzione $f : \Sigma^* \to \Sigma^*$ è una funzione computabile in tempo polinomiale se esiste una TM M che lavora in tempo polinomiale, tale che si blocca con $f(w)$ sul nastro, quando avendo ricevuto w in input.*

Def. 7.7 (Riduzione polinomiale) *Siano A e B due linguaggi. A si riduce in tempo polinomiale a B se esiste una funzione computabile in tempo polinomiale ($A \leq_p B$) tale che:*

$$\forall w, w \in A \iff f(w) \in B \tag{5}$$

f è detta riduzione polinomiale da A a B.

Teorema 7.3 (Fondamentale) *Se $A \leq_p B$ e $B \in P$ allora $A \in P$.*

Def. 7.8 (NP-completo) *Un linguaggio L è NP-completo se:*

1. $L \in NP$;
2. $\forall L' \in NP, L' \leq_p L$.

Teorema 7.4 *Se B è NP-completo e $B \leq_p A$ allora A è NP-completo.*