Automi, Calcolabilità e Complessità

Leonardo Ganzaroli

Indice

Introduzione

Questi appunti sono derivanti principalmente dalle dispense del corso di *Automi, Calcolabilità e Complessità* che ho svolto durante la laurea Triennale di informatica all'università "La Sapienza".

1 Linguaggi

Definizione Un alfabeto è un insieme finito e non vuoto di elementi detti *Simboli/Caratteri*.

Definizione Dato un alfabeto $\sum$. Si dice stringa/parola di $\sum$ una sequenza finita di simboli $\in \sum$.

L'insieme di tutte le possibili stringhe generate da un alfabeto $\sum$ è indicato con $\sum^*$.

Definizione Data una stringa x. La sua lunghezza è data dal numero dei suoi caratteri.

Nel caso la stringa sia vuota si indica con ϵ.

Definizione Data una stringa $x = x_1x_2 \dots x_n$. La sua stringa inversa è definita come $x^R = x_n \dots x_2x_1$.

Definizione Concatenare 2 stringhe $x, y \in \sum^*$ vuol dire creare una nuova stringa composta dagli elementi di x seguiti da quelli di y.

Definizione Dato un alfabeto $\sum$. Un linguaggio di $\sum$ è un sottoinsieme di $\sum^*$.

1.1 Operazioni

Essendo degli insiemi è possibile combinare i linguaggi usando le classiche operazione insiemistiche ($\cup, \cap, \ldots$).

Sono però presenti alcune operazioni aggiuntive (i linguaggi sono sullo stesso alfabeto):

- **Concatenazione**

$$L_1 \circ L_2 = \{xy \mid x \in L_1,\ y \in L_2\}$$

- **Potenza**

$$L^n = \begin{cases} \{\epsilon\} & \text{se } n = 0 \\ L \circ L^{n-1} & \text{se } n > 0 \end{cases}$$

- **Star di Kleene**

$$L^* = \bigcup_{n \geq 0} L^n$$

- **Plus di Kleene**

$$L^+ = L \circ L^*$$

1.2 Hamming

Definizione La distanza di Hamming tra due stringhe è il numero di caratteri per cui differiscono:

$$d_H(x, y) = |\{i \in [1, n] \mid x_i \neq y_i\}| \text{ con } |x| = |y| = n$$

Definizione Dato l'alfabeto $\{0, 1, \ldots, 9\}$ ed x una sua stringa. Il peso di Hamming di x è il suo numero di caratteri diversi da 0:

$$w_h(x) = |\{i \in [1, n] \mid x_i \neq 0\}| \text{ con } n = |x|$$

2 Linguaggi regolari

2.1 Automi

Definizione Un automa è una macchina che segue una serie di istruzioni in modo automatico ed ha una struttura a stati.

2.1.1 DFA

Definizione Un automa a stati finiti deterministico è una quintupla $(Q, \sum, \delta, q_0, F)$ con:

- Q = insieme finito degli stati
- $\sum$ = alfabeto finito dell'automa
- $\delta : Q \times \sum \rightarrow Q$ = funzione di transizione degli stati
- $q_o \in Q$ = stato iniziale
- $F \subseteq Q$ = insieme degli stati accettanti

La funzione di transizione si può esprimere in forma estesa $\delta^* : Q \times \sum^* \rightarrow Q$ definendola ricorsivamente come:

$$\begin{cases} \delta^*(q, \epsilon) = \delta(q, \epsilon) = q \\ \delta^*(q, aw) = \delta^*(\delta(q, a), w) \quad \text{con } a \in \sum,\ w \in \sum^* \end{cases}$$

Un DFA accetta una certa stringa $x = x_1 x_2 \dots x_n$ se esiste una sequenza di stati $s_0 s_1 \dots s_n \in Q$ tali che:

- $s_0 = q_0$
- $\forall\ i \in [0, n-1] \quad \delta(s_i, x_{i+1}) = s_{i+1}$
- $s_n \in F$

Definizione Il linguaggio di un automa è l'insieme di tutte le stringhe accettate da esso, simmetricamente si dice che l'automa riconosce quel linguaggio.

Esempio:

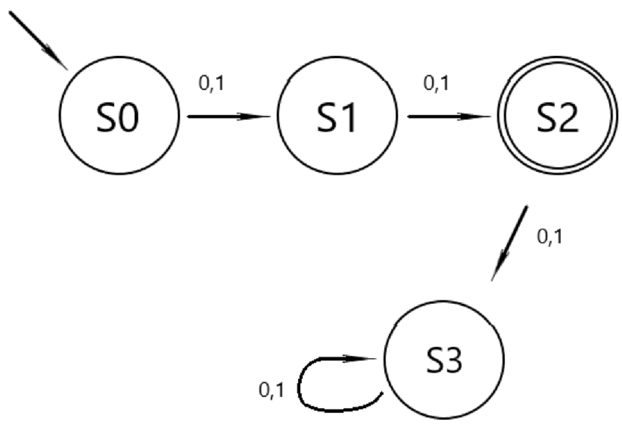


Figura 1: Esempio DFA

- $Q = \{S_0, S_1, S_2, S_3\}$
- $\sum = \{0, 1\}$
- Transizioni:

	0	1
S_0	S_1	S_1
S_1	S_2	S_2
S_2	S_3	S_3
S_3	S_3	S_3

- $q_0 = S_0$
- $F = \{S_2\}$
- $L = \{x \in \sum^* \mid |x| = 2\}$

Definizione Una configurazione di un DFA è una coppia $(q, w) \in Q \times \sum^*$.

Definizione Un passo di computazione è la relazione definita come:

$$(q_i, ax) \vdash_D (q_j, x) \iff \delta(q_i, a) = q_j$$

Definizione Una computazione è detta deterministica se ad ogni passo di computazione segue un'unica configurazione:

$$\forall\ (q, aw)\ \ \exists!\ (p, w) \mid (q, aw) \vdash_D (p, w)$$

2.1.2 NFA

Definizione $\sum_\epsilon = \sum \cup \{\epsilon\}$

Definizione Un automa a stati finiti non deterministico è una quintupla come un DFA, ma la sua funzione di transizione differisce:

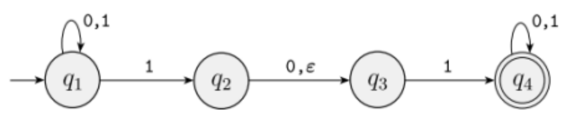
$$\delta : Q \times \sum\nolimits_\epsilon \to P(Q)$$

Quindi una transizione potrebbe portare ad un qualsiasi numero di stati.

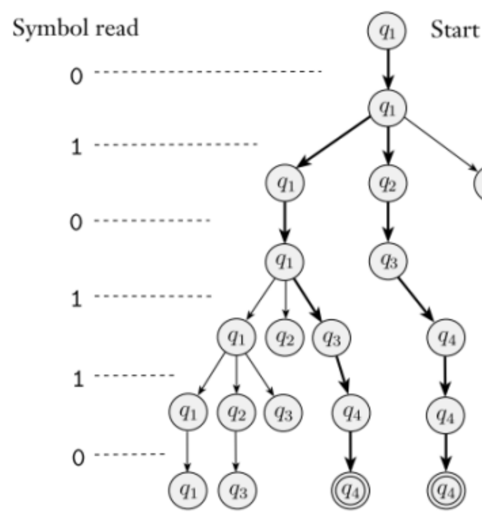
Computazione:

- Quando si presenta una transizione che porta in più stati la macchina crea delle copie di se stessa pari al numero di stati, ogni copia seguirà 1 dei percorsi. Si creano così dei rami di computazione che vengono eseguiti in parallelo
- Se non è possibile eseguire nessuna transizione con il prossimo simbolo il ramo si interrompe
- L'automa accetta la stringa se almeno un ramo finisce in uno stato di accettazione
- Se si raggiunge uno stato che ha una transizione con valore ϵ la macchina crea delle copie che seguono quegli archi ed una che resta nello stato appena raggiunto

Esempio:



La computazione di questo NFA della stringa 010110 sarà:



Equivalenza con DFA

Definizione Due automi si dicono equivalenti se e solo se riconoscono lo stesso linguaggio.

Si nota facilmente che un NFA è una generalizzazione di un DFA, inoltre è possibile costruire un DFA equivalente di un NFA con questi passaggi:

1. $Q_{DFA} = P(Q_{NFA})$
2. Trovo q_o
3. Trovo gli stati accettanti
4. "Traduco" le transizioni

Esempio:

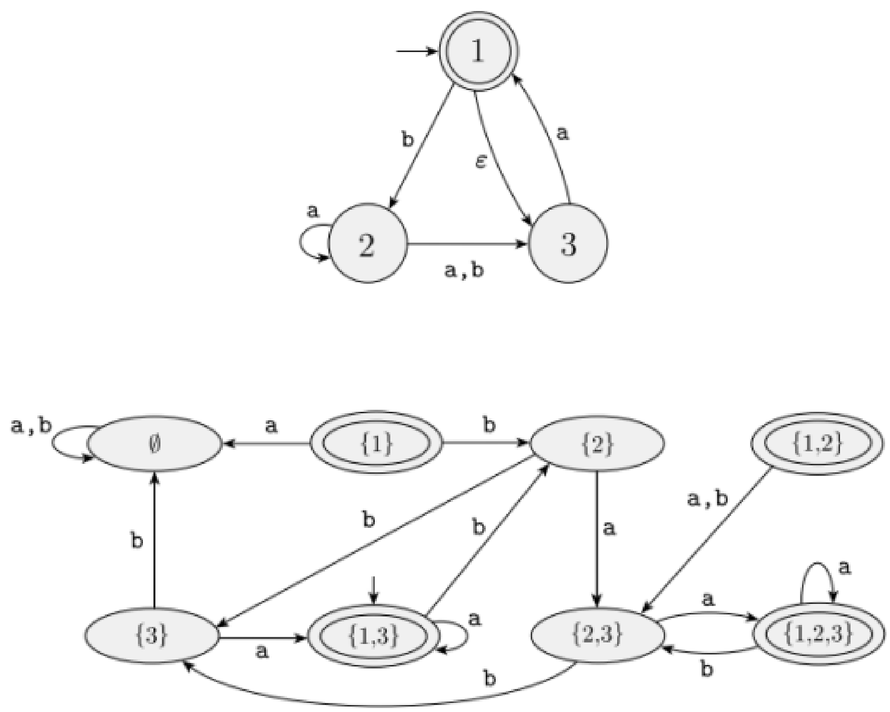


Figura 2: Trasformazione da NFA a DFA

Definizione Un linguaggio si dice regolare se e solo se esiste un DFA che lo riconosce.

L'insieme dei linguaggi regolari è quindi l'insieme:

$$\text{REG} = \{L \subseteq \sum{}^{*} \mid \exists \text{ DFA} \mid L(\text{DFA}) = L\}$$

Vista la loro equivalenza si può anche riscrivere con NFA.

2.2 Espressioni regolari

Definizione Dato un alfabeto $\sum$. Un espressione regolare è una stringa generata da quell'alfabeto che permette di descrivere un intero linguaggio.

Si possono utilizzare le operazioni $\cup, *, \circ$ (quest'ultimo spesso omesso) per combinare tra loro le espressioni regolari.

Alcuni esempi dato $\sum = \{0, 1\}$:

- 0^*10^* → tutte le stringhe con un solo 1
- $\emptyset \rightarrow \ \{\epsilon\}$
- $\sum^*$ → tutte le stringhe possibili
- $((0\sum^* 0) \cup (1\sum^* 1) \cup 0 \cup 1)$ → tutte le stringhe che iniziano e finiscono con lo stesso carattere
- $((0 \cup \epsilon)(1 \cup \epsilon)) \rightarrow \ \{\epsilon, 0, 1, 01\}$

Partendo dai singoli elementi e procedendo a ritroso è possibile trasformare un'espressione regolare in un NFA.

2.2.1 GNFA

Definizione Un automa a stati finiti non deterministico generalizzato è una versione alternativa di un NFA in cui sugli archi sono presenti espressioni regolari invece di singoli caratteri, in questo caso l'input viene gestito "a blocchi".

La funzione di transizione diventa:

$$\delta : (Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \rightarrow \text{re}(\sum)$$

Inoltre è presente un unico stato accettante.

Si può trasformare un DFA in un GNFA seguendo questi passaggi:

1. Aggiungere un nuovo stato iniziale con un arco ϵ uscente verso il vecchio stato iniziale
2. Aggiungere un nuovo stato accettante con un arco ϵ entrante da ogni vecchio stato accettante
3. Sostituire gli archi con caratteri multipli con un unico arco avente l'unione dei caratteri
4. Aggiungere archi $\emptyset$ tra gli stati non collegati, tranne:
   - tra accettante e se stesso
   - tra ogni stato e quello iniziale

Esempio:

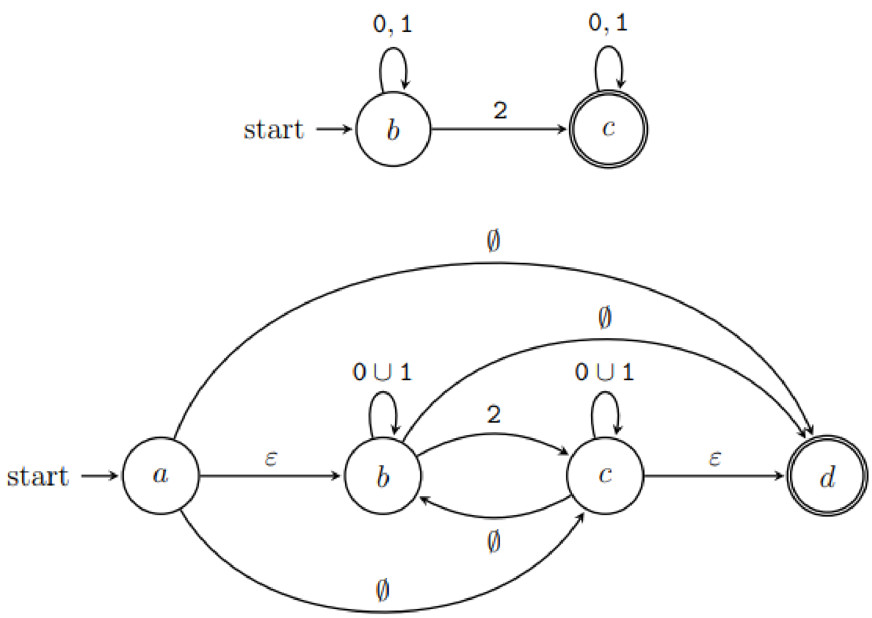


Figura 3: Trasformazione DFA in GNFA

Si può trasformare un GNFA in un'unica espressione regolare seguendo l'algoritmo:

Algorithm 1 Da GNFA a espressione regolare

```
                                                        ▷ Dato G=GNFA
if |Q| == 2 then
    return δ(q_start, q_end)
else
    Q' = Q − {q ∈ Q/{q_start, q_accept}}
    for q_i ∈ Q'/{q_accept} do
        for q_j ∈ Q'/{q_start} do
            δ'(q_i, q_j) = δ(q_i, q)δ(q, q)*δ(q, q_j) ∪ δ(q_i, q_j)
        end for
    end for
    G' = (Q', ∑, δ', q_start, q_accept)
    return questa_funzione(G')
end if
```

Dato che tutti gli automi visti finora e le espressioni regolari sono intercambiabili si può affermare che:

$$REG = L(DFA) = L(NFA) = L(GNFA) = L(REX)$$

2.3 Chiusura di REG

I linguaggi regolari risultano chiusi rispetto alle principali operazioni insiemistiche viste fin'ora, ossia il nuovo insieme che si ottiene applicandole è a sua volta in REG.

Si può verificare direttamente con gli automi (dati 2 linguaggi A e B):

- **Unione** $A \cup B$

  L'automa si forma inserendo un nuovo stato iniziale che ha 2 archi ϵ uscenti verso gli stati iniziali di A e B

- **Intersezione** $A \cap B$

  L'automa avrà le seguenti proprietà:

  - $Q = Q_A \times Q_B$
  - $\forall\ (r_1, r_2) \in Q,\ a \in \sum \quad \delta((r_1, r_2), a) = (\delta_A(r_1, a), \delta_B(r_2, a))$
  - $q_0 = (q_{0A}, q_{0B})$
  - $F = F_A \times F_B$

- **Concatenazione** $A \circ B$

  L'automa si forma aggiungendo ad ogni stato accettante di A un arco ϵ puntante allo stato iniziale di B

- **Potenza** A^n

  Per induzione:

  1. A^0 è regolare
  2. Considerando A^n regolare ho $A^{n+1} = A^n \circ A$
  3. A^n e A sono regolari e come visto prima la concatenazione crea un linguaggio regolare

- **Star** A^*

  Aggiungo un nuovo stato iniziale con arco ϵ puntante al vecchio stato iniziale ed aggiungo ulteriori archi ϵ dagli stati accettanti al vecchio stato iniziale

- **Complemento** $\bar{A}$

  Creo un automa con $F = Q - F_{old}$

2.4 Pumping lemma

Definizione Un linguaggio si dice non regolare se non esiste un DFA che lo riconosce.

Definizione Dato $A \in$ REG. Esiste $p \in \mathbf{N}$ detto lunghezza di pumping tale che $\forall\ s \in A$ con $|s| \geq p\ \ \exists x, y, z \mid s = xyz$, inoltre:

- $\forall\ i \geq 0\ \ xy^iz \in A$
- $|y| > 0$
- $|xy| \leq p$

Dimostro che $\{0^n1^n \mid n \in \mathbf{N}\}$ non è regolare.

Per assurdo esiste p lunghezza di pumping, consideriamo la stringa $s = 0^p1^p$, dividendola in xyz risulta:

- $xy = 0000\ldots0$ dato che $|xy| \leq p$
- y deve contenere almeno uno zero
- se $i = 0 \rightarrow xy^0z = xz$, ma essendo $|y| > 0$ ci si ritrova con almeno uno 0 in meno e di conseguenza la stringa non appartiene al linguaggio

---

3 Linguaggi acontestuali

3.1 Grammatiche acontestuali

Definizione Una grammatica è un insieme di regole di sostituzione di stringhe e puo produrre quest'ultime partendo da una variabile.

Definizione Si definisce acontestualità la caratteristica per cui il lato SX di una grammatica è sempre composto da un solo simbolo.

Definizione Una grammatica acontestuale (CFG) è una quadrupla $(V, \sum, R, S)$ con:

- V insieme finito delle variabili
- $\sum$ insieme finito dei terminali, risulta $(\sum \cap V) = \emptyset$
- R insieme finito delle regole
- $S \in V$ variabile iniziale

Le regole hanno la forma $A \to X$ dove:

- $A \in V$
- $X \in (V \cup \sum_\epsilon)^*$

Definizione Date x, y, z stringhe. Se esiste la regola $A \to y$ allora xAz produce xyz e si indica con $xAz \Rightarrow xyz$.

Definizione Date x, y stringhe. x deriva y se $x = y$ oppure se $\exists x_1 x_2 x_3 \ldots$ tali che $x \Rightarrow x_1 \Rightarrow x_2 \Rightarrow \ldots \Rightarrow y$ e si denota con $x \overset{*}{\Rightarrow} y$.

Definizione Una derivazione a sinistra è una per cui in ogni produzione interna ad essa si valuta la variabile più a sinistra.

Definizione Date x, y stringhe con $x \neq y$. Una grammatica è detta ambigua se una certa stringa z può essere derivata a sinistra sia da x che da y.

Definizione Il linguaggio di una CFG è l'insieme delle stringhe che essa può generare:

$$L(G) = \{w \in \sum^{*} \mid S \overset{*}{\Rightarrow} w\}$$

Se consideriamo la classe dei linguaggi delle CFG (CFL) risulta:

$$REG \subset CFL$$

Esempio:

Una possibile grammatica è:

$$A \to 0A1$$

$$A \to B$$

$$B \to \#$$

Essa ha:

- $V = \{A, B\}$
- $\sum = \{0, 1, \#\}$
- A variabile iniziale

Essa genera stringhe del tipo $000 \ldots 0\#111 \ldots 1$

Si può esprimere graficamente il processo di creazione di una stringa tramite l'albero di derivazione:

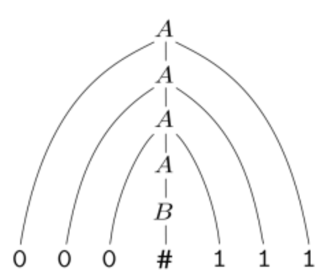


Figura 4: Albero di derivazione di 000#111

3.1.1 Forma normale di Chomsky

Una CFG è in CNF se ogni sua regola è della forma:

- $A \rightarrow BC$
- $A \rightarrow a$
- $S \rightarrow \epsilon$

Ogni CFG si può trasformare in questa forma con questi passaggi:

1. Introdurre S_0
2. Eliminare le ϵ-produzioni
3. Eliminare le regole unitarie
4. Sistemare le regole rimaste

3.2 PDA

Definizione Un automa a pila è un NFA con stack, è una sestupla $(Q, \sum, \Gamma, \delta, q_0, F)$ con:

- Q = insieme finito degli stati
- $\sum$ = alfabeto finito dell'automa
- Γ = alfabeto finito della pila
- $\delta : Q \times \sum_\epsilon \times \Gamma_\epsilon \rightarrow P(Q \times \Gamma_\epsilon)$ = funzione di transizione degli stati
- $q_o \in Q$ = stato iniziale
- $F \subseteq Q$ = insieme degli stati accettanti

Sugli archi si usa la notazione $a; b \rightarrow c$ in cui:

- a è l'input
- b è il valore in cima allo stack
- c è il valore da inserire nello stack

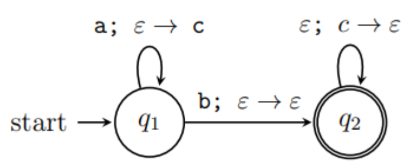
Se ho una transizione $1; b \rightarrow z$ la transizione "scatta" se:

- In input c'è 1
- In cima alla stack c'è b

I valori b, c indicano quindi l'operazione che si va a svolgere sulla stack:

1. $b \neq \epsilon$, **POP**
2. $c \neq \epsilon$, **PUSH** di c

Esempio:



La computazione di "aab" nel seguente PDA sarà:

1. Leggi a, resta in q_1 e fai il push di c
2. Leggi a, resta in q_1 e fai il push di c
3. Leggi b, passa a q_2
4. Fai il pop di c, resta in q_2
5. Fai il pop di c, resta in q_2

Un PDA accetta una certa stringa $x = x_1 x_2 \dots x_n$ se esiste una sequenza di stati $s_0 s_1 \dots s_n \in Q$ ed una sequenza di stringhe $str_0, str_1, \dots, str_n \in \Gamma^*$ tali che:

- $s_0 = q_0$
- $str_o = \epsilon$
- $\forall\ i \in [0, n-1]\ \exists a, b \in \Gamma_\epsilon, t \in \Gamma^* \mid (s_{i+1}, b) \in \delta(s_i, x_{i+1}, a) \land str_i = at \land str_{i+1} = bt$
- $s_n \in F$

3.2.1 Equivalenza con CFG

Si può inserire una stringa nello stack di un PDA tra 2 stati p, q inserendo tra di essi degli stati intermedi in sequenza i cui archi eseguono solamente il push di un carattere alla volta. Graficamente si possono omettere inserendo sull'arco l'intera stringa.

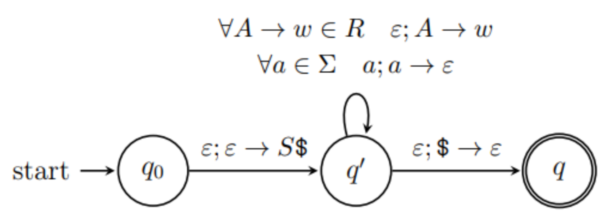
Con questa osservazione si può definire un PDA equivalente di una CFG come:

$$(\{q_0, q', q\} \cup E, \sum, V \cup \sum, \delta, q_0, \{q\})$$

Con E insieme degli stati necessari per usare la notazione vista prima.

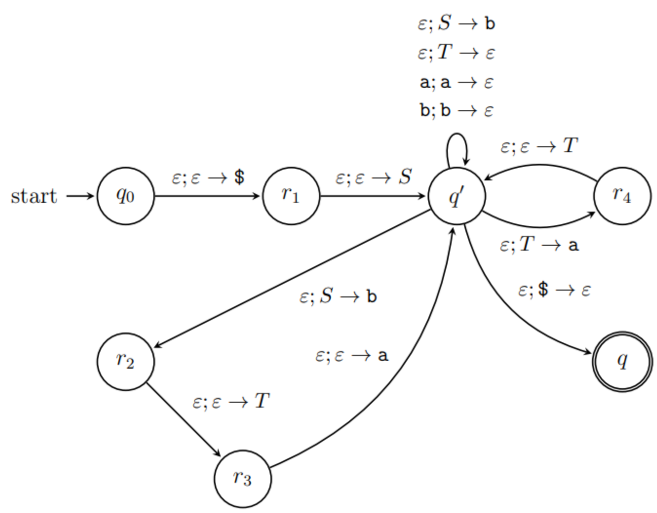
Inoltre:

- $\delta(q_0, \epsilon, \epsilon) = \{(q', S\$)\}$
- $\forall\ A \in V \quad \delta(q', \epsilon, A) = \{(q', w) \mid A \to w \in R,\ w \in \Gamma^*\}$
- $\forall\ a \in \sum \quad \delta(q', a, a) = \{(q', \epsilon)\}$
- $\delta(q', \epsilon, \$) = \{(q, \epsilon)\}$



Data la grammatica:

$$S \rightarrow aTb \mid b$$
$$T \rightarrow Ta \mid \epsilon$$



Ovviamente si può fare anche la trasformazione inversa, partendo da un PDA ne creo uno equivalente $(Q', \sum, \Gamma, \delta', q_0, \{q_{accept}\})$ tale che:

- ogni transizione svolge una sola operazione, indico con Q_δ gli stati aggiunti per svolgere questa operazione
- q_{accept} è l'unico stato accettante
- prima di accettare una stringa si svuota lo stack
- $Q' = Q \cup Q_\delta \cup \{q_{accept}\}$

Creo adesso la CFG in modo che:

- $V = \{A_{p,q} \mid p, q \in Q'\}$
- $S = A_{q_0, q_{accept}}$

Si può quindi affermare che i PDA e le CFG lavorano con la stessa classe di linguaggi (CFL).

3.3 Chiusura CFL

A differenza di REG i CFL sono chiusi solo rispetto ad alcune operazioni:

- **Unione**

  Basta creare una grammatica equivalente partendo dalle grammatiche dei singoli linguaggi:

  - Aggiungo una variabile iniziale S
  - $V = (\bigcup_{i=0}^{n} V_i) \cup \{S\}$
  - $\sum = \bigcup_{i=0}^{n} \sum_i$
  - $R = (\bigcup_{i=0}^{n} R_i) \cup \{S \to S_j \mid j \in [1, n]\}$

- **Concatenazione**

  Come sopra ma l'ultimo punto diventa $R = (\bigcup_{i=0}^{n} R_i) \cup \{S \to S_1 S_2 \dots S_n\}$

- **Star**

  Creo una nuova grammatica equivalente tale che:

  - Inserisco una variabile iniziale S_0
  - $R' = R \cup \{S_0 \to \epsilon, S_0 \to S, S_0 \to S_0 S_0\}$

Date le grammatiche:

$$G_1 : A \to 0A1 \mid \epsilon$$

$$G_2 : A \to 1A0 \mid \epsilon$$

L'unione crea la grammatica:

$$S \to A \mid B$$

$$A \to 0A1 \mid \epsilon$$

$$B \to 1B0 \mid \epsilon$$

La concatenazione invece:

$$S \to AB$$

$$A \to 0A1 \mid \epsilon$$

$$B \to 1B0 \mid \epsilon$$

La star del primo:

$$S \to \epsilon \mid A \mid SS$$

$$A \to 0A1 \mid \epsilon$$

3.4 Pumping lemma 2.0

Definizione Dato $A \in$ CFL. Esiste $p \in \mathbf{N}$ detto lunghezza di pumping tale che $\forall\ s \in A$ con $|s| \geq p\ \ \exists u, v, x, y, z \mid s = uvxyz$, inoltre:

- $\forall\ i \geq 0\ \ uv^i xy^i z \in A$
- $|vy| > 0$
- $|vxy| \leq p$

4 Calcolabilità

4.1 Macchina di Turing

Una macchina di Turing è un modello matematico computazionale che descrive una macchina astratta che manipola (legge e scrive) i dati contenuti su un nastro di lunghezza potenzialmente infinita, secondo un insieme prefissato di regole ben definite.

Definizione Una macchina di Turing è una settupla $(Q, \sum, \Gamma, \delta, q_{start}, q_{accept}, q_{reject})$ con:

- Q = insieme finito degli stati
- $\sum$ = alfabeto finito della macchina
- Γ = alfabeto del nastro, $\sum \subseteq \Gamma$
- $\sqcup$ = cella del nastro vuota, $\notin \sum, \in \Gamma$
- $q_{start} \in Q$ = stato iniziale
- $q_{accept} \in Q$ = stato accettante, se raggiunto la macchina accetta immediatamente
- $q_{reject} \in Q$ = stato rifutante, se raggiunto la macchina rifiuta immediatamente
- $\delta : (Q - \{q_{accept}, q_{reject}\}) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ = funzione di transizione degli stati

Sugli archi si usa la notazione $a; b \rightarrow X$ in cui:

- a è il simbolo letto sul nastro
- b è il simbolo scritto sul nastro al posto di a
- X è lo spostamento a sinistra(L) o a destra(R)

Definizione Una stringa $uqav$ è detta configurazione di una TM se:

- $q \in Q$, stato attuale
- $a \in \Gamma$, simbolo della cella attuale
- $u \in \Gamma^*$, simboli precedenti ad a sul nastro
- $v \in \Gamma^*$, simboli successivi ad a sul nastro

Data una configurazione uaq_ibv:

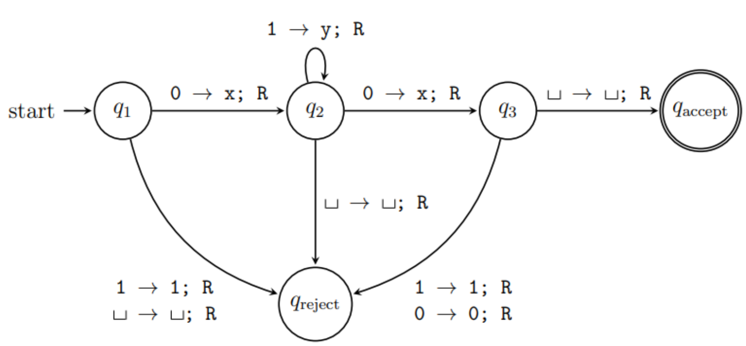
- essa produce $uq_jacv \iff \delta(q_i, b) = (q_j, c, L)$
- essa produce $uacq_jv \iff \delta(q_i, b) = (q_j, c, R)$

Una TM accetta una certa stringa $x \in \sum^*$ se esiste una sequenza di configurazioni $c_1c_2 \dots c_k$ tali che:

- $c_1 = q_{start}w$
- $\forall\ i \in [1, k-1]\ \ c_i$ produce c_{i+1}
- $q_{accept} \in c_k$

Esempio:

La TM che riconosce il linguaggio $\{01^n0 \mid n \in \mathbf{N}\}$ è:



Definizione Una TM è detta decisore se termina sempre l'esecuzione, inoltre si dice che decide il suo linguaggio.

L'insieme dei linguaggi Turing-riconoscibili è definito come:

$$\text{REC} = \{L \subseteq \sum^* \mid \exists \text{ TM} \mid L(\text{TM}) = L\}$$

Quello dei linguaggi coTuring-riconoscibili invece:

$$\text{coREC} = \{L \subseteq \sum^* \mid \bar{L} \in REC\}$$

Sostituendo la TM con un decisore ottengo invece l'insieme DEC dei linguaggi Turing-decidibili, risulta $DEC \subset REC$, inoltre $DEC = REC \cap coREC$.

4.1.1 Varianti

- **Stay-put**

  Oltre ad andare a SX/DX può anche restare sulla cella corrente.

- **Multinastro**

- **Non deterministica**

- **Enumeratore**

  È connessa ad una "stampante" tramite cui stampa tutte le stringhe di un linguaggio in ordine casuale e con possibili ripetizioni, il nastro di input è vuoto.

4.1.2 Tesi di Church-Turing

Data una funzione f:

$$f \text{ è computabile da un algoritmo} \iff f \text{ è computabile da una TM}$$

In alternativa si può dire : *Se un problema è umanamente calcolabile, allora esisterà una macchina di Turing in grado di risolverlo.*

Quindi è possibile affermare che TM ed algoritmi sono equivalenti tra loro, implicando che ogni tipo di computazione si può svolgere tramite TM.

Definizione Una TM è detta universale se può simulare qualsiasi altra TM.

Definizione Un modello di calcolo è detto Turing-completo se è equivalente ad una TM universale.

4.2 Decidibilità

4.2.1 Problemi decidibili

Definizione Dato un oggetto O. La sua codifica $\langle O \rangle$ è una stringa che ne descrive le caratteristiche.

Definizione Il linguaggio per il problema dell'accettazione è definito come:

$$A_x = \{\langle A, w \rangle \mid A \text{ è } X, w \in L(A)\}$$

X può essere:

- DFA

  Se la codifica è giusta la TM simula il DFA con input w ed accetta/rifiuta in base allo stato finale della simulazione.

- NFA

  Si trasforma in un DFA equivalente e si esegue come visto sopra.

- REX

  Si trasforma in un NFA equivalente e si esegue come visto sopra.

- CFG

  Si trasforma in CNF e successivamente:

  - Se $|w| = 0$ e $S \to \epsilon \in G_{CNF}$ allora accetta, altrimenti rifiuta
  - Se $|w| \geq 1$ genera tutte le produzioni con lunghezza $2|w| - 1$ e se w è tra queste accetta, rifiuta altrimenti

Definizione Il linguaggio per il problema del vuoto è definito come:

$$E_x = \{\langle A \rangle \mid A \text{ è } X, L(A) = \emptyset\}$$

X può essere:

- DFA

  Se la codifica è giusta la TM marca lo stato iniziale e tutti quelli che hanno archi entranti da stati marcati in modo iterativo, se alla fine uno stato accettante è marcato accetta, altrimenti rifiuta.

- CFG

  Se la codifica è giusta la TM marca i terminali e tutte le regole che hanno nella parte dx terminali o variabili marcate in modo iterativo, se alla fine S è marcato accetta, altrimenti rifiuta.

Definizione Il linguaggio per il problema dell'equivalenza tra 2 DFA è definito come:

$$EQ_{DFA} = \{\langle A, B \rangle \mid A, B \text{ sono DFA }, L(A) = L(B)\}$$

Se la codifica è giusta la TM crea un DFA C tale $L(C) = L(A) \triangle L(B)$, esegue poi il problema del vuoto su C ed accetta/rifiuta in base al risultato di quella simulazione.

Tutti i linguaggi visti in questa sezione portano inevitabilmente alla fine dell'esecuzione, quindi si considerano Turing-decidibili.

4.2.2 Argomento diagonale di Cantor

Definizione L'argomento diagonale di Cantor è una tecnica dimostrativa atta a dimostrare l'esistenza o meno di una funzione biettiva tra due insiemi A e B disponendo i loro elementi in forma tabellare.

In particolare si può usare per provare che un certo insieme non è numerabile se non esiste una funzione biunivoca da $\mathbf{N}$ all'insieme.

Dimostrare che l'insieme delle stringhe binarie infinite non è numerabile:

- Supponiamo per assurdo che esista una funzione $f : \mathbf{N} \to B$ biunivoca
- Considero la stringa x creata invertendo la diagonale:

$$\begin{array}{c|ccccc} 1 & \mathbf{0} & 1 & 0 & 1 & \dots \\ 2 & 0 & \mathbf{1} & 0 & 1 & \dots \\ 3 & 1 & 1 & \mathbf{1} & 0 & \dots \\ 4 & 0 & 0 & 0 & \mathbf{0} & \dots \\ \dots & \dots & \dots & \dots & \dots & \end{array}$$

In questo caso sarà $x = 1001\dots$

- Data la sua costruzione si avrà che la stringa non può essere immagine di nessun numero, infatti se $f(i) = x$ si dovrà avere che l'elemento sulla diagonale ($x_{i,i}$) è diverso da se stesso
- Quindi la funzione non è suriettiva

Si può usare questo concetto per dimostrare che esistono dei linguaggi non riconoscibili da una TM.

4.2.3 Problemi indecidibili

I linguaggi visti prima (accettazione,vuoto,equivalenza) sono indecidibili nel caso in cui X sia una TM, lo sono anche:

- **Terminazione** $\{\langle M, w\rangle \mid M$ è TM e $M(w)$ termina$\}$
- **Regolarità** $\{\langle M\rangle \mid M$ è TM, $L(M) \in REG\}$

Esempio:

Si può dimostrare che l'accettazione è indecidibile usando un ragionamento simile a quello usato per l'argomento diagonale:

- Per assurdo la considero decidibile da un decisore H
- Definisco la TM D che dato in input $\langle M\rangle$ esegue H con input $\langle M, \langle M\rangle\rangle$ e restituisce l'opposto di quella simulazione
- Passando a D l'input $\langle D\rangle$ risulta che se D accetta $\langle D\rangle$ in H allora D non accetta $\langle D\rangle$ portando ad una contraddizione, infatti D dà il risultato opposto di se stesso
- Quindi il problema non è decidibile

4.2.4 Riducibilità

Definizione Dati due problemi A e B. Si definisce come riduzione il metodo dimostrativo tramite cui sapendo la soluzione di B è possibile risolvere A.

Esempio:

Per provare che il problema della terminazione non è decidibile riduco quello del riconoscimento ad esso:

- per assurdo esiste un decisore H per $HALT$
- costruisco una TM D tale che dato l'input $\langle M, w\rangle$:
  - esegue H con $\langle M, w\rangle$, se rifiuta allora rifiuta
  - altrimenti simula M con w ed accetta se essa accetta

$$\text{Si ha } \langle M, w\rangle \in L(D) \iff \langle M,\ w\rangle \in L(H), w \in L(M) \iff \langle M, w\rangle \in A_{TM}$$

Si nota che $L(D) = A_{TM}$, quindi teoricamente D sarebbe un decisore essendo basato su H ma il problema dell'accettazione non è decidibile e di conseguenza neanche quello della terminazione.

Tramite mappatura

Definizione Una funzione $f : \sum^* \to \sum^*$ è detta calcolabile se esiste una TM F tale che:

$$\forall\ w \in \sum^* \quad F(w) \text{ termina solo con } f(w) \text{ sul nastro}$$

Definizione Dati $A, B \in \sum^*, \neq \emptyset$. Si dice che A è riducibile a B tramite mappatura ($A \leq_m B$) se:

$$\exists\ f : \sum^* \to \sum^* \mid w \in A \iff f(w) \in B$$

Si ha che se $A \leq_m B$:

- $B \in DEC \Rightarrow A \in DEC$
- $B \in REC \Rightarrow A \in REC$
- $A \notin DEC \Rightarrow B \notin DEC$
- $A \notin REC \Rightarrow B \notin REC$

Esempio:

Dimostrare che $L = \{\langle M \rangle \mid M$ è TM ed accetta stringhe di lunghezza dispari$\}$ è indecidibile.

Definisco una funzione tale che $A_{TM} \leq_m L$ in questo modo:

- Dato un input $\langle M, w \rangle$
- Costruisco una TM M' tale che:
  - se l'input $|x|$ è pari rifiuta
  - altrimenti esegui M su w ed accetta solo se M accetta
  - dai in output $\langle M' \rangle$

- se $\langle M, w \rangle \in A_{TM} \Rightarrow L(M')$ contiene solo le stringhe di lunghezza dispari $\Rightarrow$ $\langle M' \rangle \in L$
- altrimenti $\langle M' \rangle \notin L$

---

5 Complessità

Definizione Il limite superiore asintotico (O-grande) di $f(n)$ è una funzione $g(n)$ tale che:

$$\exists c, n_0 \in \mathbf{N_{\geq 0}} \mid \forall n \geq n_0 \quad f(n) \leq c * g(n)$$

Definizione Il limite inferiore asintotico (O-piccola) di $f(n)$ è una funzione $g(n)$ tale che:

$$\forall c \in \mathbf{R_{\geq 0}} \quad \exists n_0 \in \mathbf{N_{\geq 0}} \mid \forall n \geq n_0 \quad f(n) < c * g(n)$$

Date $f, g, h : \mathbf{N} \to \mathbf{R}^+$ si ha (valgono anche per O-grande):

- $\forall\ c \in \mathbf{R} \quad f(n) = c * o(g(n)) \Rightarrow f(n) = o(g(n))$
- $f(n) = o(g(n)) + o(h(n)) \Rightarrow f(n) = o(m(n))$ con $m(n) = max(g(n), h(n))$
- $f(n) = o(g(n)) * o(h(n)) \Rightarrow f(n) = o(g(n) * h(n))$

5.1 Temporale

Definizione Dato un decisore D. La complessità temporale è una funzione $f : \mathbf{N} \to \mathbf{N}$ tale che $f(n)$ è il numero di passi necessari a D per processare una certa stringa lunga n, nel caso sia non deterministico è il massimo numero di passi necessari ad ogni ramo.

5.1.1 Classi

Definizione La classe dei linguaggi decidibili in tempo $O(t(n))$ è l'insieme:

$$DTIME(t(n)) = \{L \in DEC \mid L \text{ è decidibile da una TM in tempo } O(t(n))\}$$

In modo simile definisco $NTIME$, l'unica differenza è che la TM è non deterministica.

P

$$P = \bigcup_{k=0}^{+\infty} DTIME(n^k)$$

EXP

$$P = \bigcup_{k=1}^{+\infty} DTIME(2^{n^k})$$

NP

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

NEXP

$$P = \bigcup_{k=1}^{+\infty} NTIME(2^{n^k})$$

Classi complementari

Definizione $coP = \{A \in DEC \mid \bar{A} \in P\}$

Definizione $coNP = \{A \in DEC \mid \bar{A} \in NP\}$

Definizione $coEXP = \{A \in DEC \mid \bar{A} \in EXP\}$

5.1.2 Riducibilità

Definizione Una riduzione in tempo polinomiale ($A \leq_m^p B$) è una riduzione tra 2 linguaggi tramite mappatura in cui f è calcolabile in tempo polinomiale.

5.1.3 Classi NP-Complete

Definizione NP-Hard$= \{B \subset \sum^* \mid \forall\ A \in NP \quad A \leq_m^p B\}$ con $B \neq \emptyset$.

Definizione Un linguaggio è NP-Completo se $\in (NP \cap NP$-Hard).

5.1.4 Teorema di Cook-Levin

Questo teorema afferma che il problema della soddisfacibilità booleana è NP-Completo.

5.2 Spaziale

Definizione Dato un decisore D. La complessità spaziale è una funzione $f : \mathbf{N} \to \mathbf{N}$ tale che $f(n)$ è il numero di celle usate da D per processare una certa stringa lunga n, nel caso sia non deterministico è il massimo numero di celle usate da ogni ramo.

N.B. Le celle per l'input non vengono considerate.

5.2.1 Classi

Definizione La classe dei linguaggi decidibili in spazio $O(s(n))$ è l'insieme:

$$DSPACE(s(n)) = \{L \in DEC \mid L \text{ è decidibile da una TM in spazio } O(s(n))\}$$

In modo simile definisco $NSPACE$, l'unica differenza è che la TM è non deterministica.

Rapporto con il tempo

Data $f(n)$ con $f(n) \geq n$ risulta:

$$DTIME(f(n)) \subseteq DSPACE(f(n))$$
$$NTIME(f(n)) \subseteq NSPACE(f(n))$$

Inoltre se $f(n) \geq \log n$:

$$DSPACE(f(n)) \subseteq DTIME(2^{O(f(n))})$$

Teorema di Savitch

Data una funzione $f(n) \geq \log n$ si ha:

$$NSPACE(f(n)) \subseteq DSPACE(f^2(n))$$

L

$$L = DSPACE(\log n)$$

PSPACE

$$PSPACE = \bigcup_{k=1}^{+\infty} DSPACE(n^k)$$

EXPSPACE

$$EXPSPACE = \bigcup_{k=1}^{+\infty} DSPACE(2^{n^k})$$

NL

$$NL = NSPACE(\log n)$$

NPSPACE

$$NPSPACE = \bigcup_{k=1}^{+\infty} NSPACE(n^k)$$

NEXPSPACE

$$NEXPSPACE = \bigcup_{k=1}^{+\infty} NSPACE(2^{n^k})$$

Classi complementari

Definizione $coL = \{A \in DEC \mid \bar{A} \in L\}$

Definizione $coPSPACE = \{A \in DEC \mid \bar{A} \in PSPACE\}$

Definizione $coEXPSPACE = \{A \in DEC \mid \bar{A} \in EXPSPACE\}$

5.2.2 Riducibilità

Definizione Una riduzione in spazio logaritmico ($A \leq_m^L B$) è una riduzione tra 2 linguaggi tramite mappatura in cui f è calcolabile in spazio logaritmico.

5.2.3 Classi NL-Complete

Definizione NL-Hard= $\{B \subset \sum^* \mid \forall\ A \in NL\ \ A \leq_m^L B\}$ con $B \neq \emptyset$.

Definizione Un linguaggio è NL-Completo se $\in (NL \cap NL\text{-Hard})$.

5.2.4 Teorema di Immerman-Szelepcsényi

Il teorema afferma che la classe NL è chiusa rispetto al suo complemento, ossia:

$$NL = coNL$$

5.3 Teoremi di gerarchia

Definizione Una funzione $f : \mathbf{N} \to \mathbf{N}$ con $f(n) \geq \log n$ è spazio-costruibile se:

$$g : \sum^* \to \sum^* : 1^n \to f(n)_2 \text{ con } f(n)_2 \text{ codifica binaria di } f(n)$$

si può calcolare in O(f(n)).

In egual modo si può definire una funzione tempo-costruibile.

Data una funzione spazio-costruibile esiste un linguaggio decidibile da una TM in spazio $O(f(n))$ ma non in spazio $o(f(n))$.

Data una funzione tempo-costruibile esiste un linguaggio decidibile da una TM in tempo $O(f(n))$ ma non in tempo $o(\frac{f(n)}{\log(f(n))})$.

5.3.1 Relazioni tra le classi

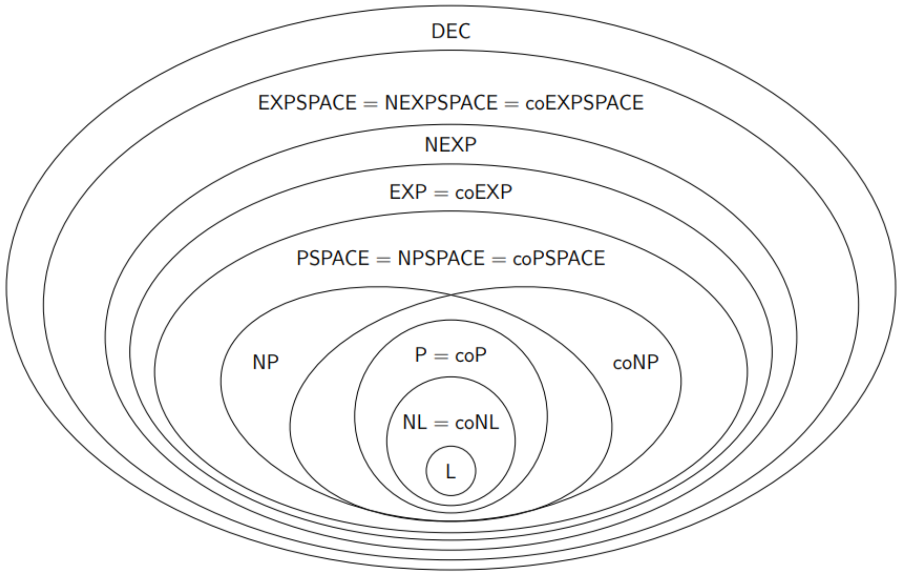


Figura 5: Tutte le classi viste