Esercizi di complessità e calcolabilità

versione 1.2.1

Guido Sciavicco

Giovanni Pagliarini

CAPITOLO 1

Linguaggi Regolari

Esercizio 1

Scrivere le espressioni regolari corrispondenti ai seguenti linguaggi, con $\Sigma = \{0, 1\}$:

- $\{w \mid w$ contiene esattamente un $1\}$;

0^*10^*

Una parola che contiene esattamente un 1 è composta da un numero qualsiasi (anche nullo) di 0, seguito da un 1, seguito nuovamente da un numero qualsiasi di 0.

- $\{w \mid w$ contiene almeno un $1\}$;

$0^*1(0+1)^*$

Una parola che contiene un 1 o più è composta da un numero qualsiasi (anche nullo) di 0, seguito da un 1, seguito nuovamente da una qualsiasi stringa di 1 e 0. Si noti che anche $(0+1)^*1(0+1)^*$ costituisce una risposta corretta.

- $\{w \mid w$ contiene la stringa $001\}$;

$(0+1)^*001(0+1)^*$

Una parola che contiene la sequenza 001 è composta una qualsiasi stringa da 1 e 0 seguita dalla sequenza 001, seguita da un'altra qualsiasi stringa di 1 e 0.

- $\{w \mid |w|$ è un multiplo di $3\}$

$((0+1)(0+1)(0+1))^*$

Le parole di lunghezza 3 sono catturate dall'espressione $(0+1)(0+1)(0+1)$. La chiusura di Kleene di questa espressione è esattamente l'insieme di parole

di lunghezza multipla di 3 (si noti che ϵ è una parola di lunghezza multipla di 3, ed è infatti inclusa nel linguaggio).

Esercizio 2

Si dica se sono regolari i linguaggi sull'alfabeto $\Sigma = \{a, b\}$ formati da:

- tutte le parole con non più di tre a;

Sì.

L'espressione regolare $b^*+b^*ab^*+b^*ab^*ab^*+b^*ab^*ab^*ab^*$ cattura esattamente il linguaggio richiesto. Si noti che si può scrivere anche $b^*(\epsilon+ab^*(\epsilon+(ab^*(\epsilon+ab^*))))$.

- tutte le parole con esattamente una occorrenza della sottostringa aa;

Sì.

L'espressione regolare $(b + ab)^*aa(ba + b)^*$ cattura esattamente li linguaggio richiesto. Si noti che l'espressione si può riscrivere in diversi modi; ad esempio, $(b^*ab^+)^*aa(b^+ab^*)^*$.

- tutte le parole con lo stesso numero di a e di b.

No.

Più avanti nel corso vedremo più in dettaglio perché non questo linguaggio non è regolare. Nel frattempo, si focalizzi l'attenzione sul perché falliscono i tentativi di dare un automa o una espressione regolare per tale linguaggio.

Esercizio 3

Quali tra le seguenti affermazioni sono vere?

- $baa \in a^*b^*a^*b^*$;

Vero.

La chiusura di Kleene cattura anche zero occorrenze dell'espressione a cui viene applicata; dunque, il primo a^* e l'ultimo b^* possono annullarsi, mentre la sotto-espressione b^*a^* può catturare baa.

- $b^*a^* \cap a^*b^* = a^* \cup b^*$;

Vero.

Ovviamente, a^* e b^* sono catturate sia da b^*a^* che a^*b^*. Inoltre, se una parola con zero o più b viene catturata da b^*a^*, allora le sue eventuali a si presentano solamente in coda, e quindi la parola può essere al contempo a^*b^* solo se essa è esattamente b^*. Un argomento simile può essere fatto per parole con zero o più a^*.

- $(a+b)^* = a^* + b^*$;

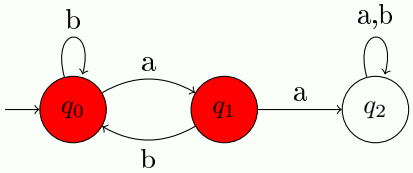
Falso.

$abab$ è una parola $\in (a+b)^*$ ma $\notin (a^* + b^*)$.

Esercizio 4

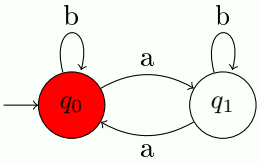
Dato $\Sigma = \{a, b\}$, costruire automi DFA che accettino i linguaggi formati da:

- tutte e sole le parole che non contengono la parola aa;



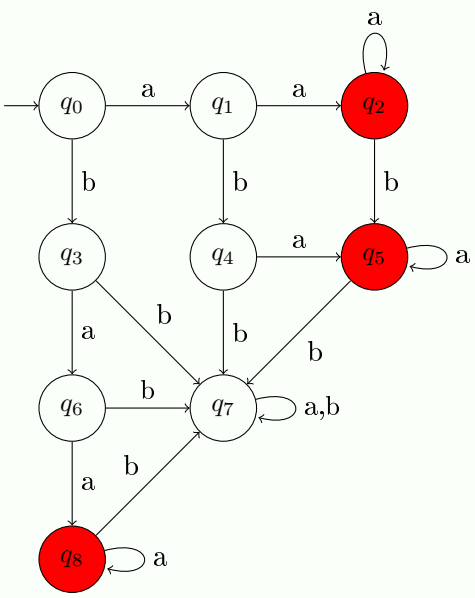
L'automa proposto è progettato in un modo per il quale, non appena una sequenza di due a viene rilevata, la computazione finisce su un *pozzetto*, ovvero uno stato non finale dal quale non è possibile raggiungere nessuno stato finale.

- tutte e sole le parole con un numero pari di a;



I due stati dell'automa proposto rappresentano rispettivamente le condizioni di aver consumato un numero di a pari e dispari; ovviamente, si passa da uno stato all'altro nel momento in cui si consuma una a, e solamente il primo dei due stati è finale.

- tutte e sole le parole che contengono almeno due a ed al massimo un b;



In questo caso, per costruire un automa, è conveniente considerare una espressione regolare per il linguaggio. Considerando le casistiche, un'espressione intuitiva per il linguaggio richiesto è $aa^+ + aa^+ba^* + aba^+ + baa^+$, che può essere riscritta in maniera strutturata e deterministica come $a(a^+(\epsilon + ba^*) + ba^+) + baa^+$, e che in maniera naturale induce il DFA proposto.

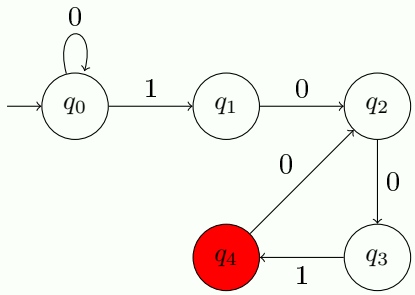
Esercizio 5

Si consideri il DFA $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_4\}, \Sigma = \{0,1\}, \delta, q_0, F = \{q_4\})$, dove le seguenti tuple costituiscono δ:

- ➔ $((q_0, 0), q_0)$; $((q_0, 1), q_1)$;
- ➔ $((q_1, 0), q_2)$;
- ➔ $((q_2, 0), q_3)$;
- ➔ $((q_3, 1), q_4)$;
- ➔ $((q_4, 0), q_2)$;

Adesso:

- si disegni l'automa;



- si dica quali delle seguenti parole sono riconosciute:
  - 0100;

No.

  - $w \in 0^*10^*1^*0^*$;

No.

$1 \in 0^*10^*1^*0^*$ ma 1 non è accettata dall'automa.

  - $w \in 0^*10^*1^*0^+$;

No.

$10 \in 0^*10^*1^*0^+$ ma 10 non è accettata dall'automa.

  - $w \in 0^*1001$;

Sì.

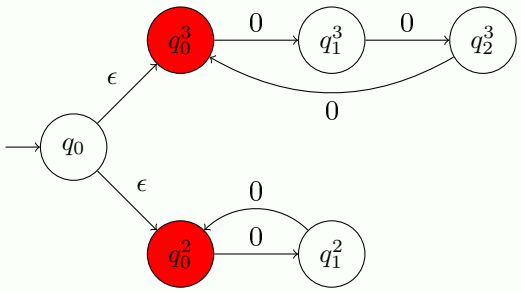
- si ricavi una espressione regolare che catturi esattamente il linguaggio dell'automa.

$0^*1(001)^+$.

Esercizio 6

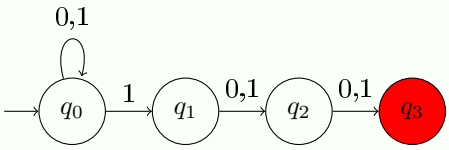
Costruire automi NFA che accettino i linguaggi formati da:

- tutte e sole le parole dell'alfabeto $\Sigma = \{0\}$ di lunghezza multiplo di 2 o di 3;



Dopo un passo non-deterministico per indovinare se la parola può avere lunghezza multiplo di 2 oppure di 3, l'automa proposto prevede due sotto-routine per riconoscere parole rispettivamente di un tipo o dell'altro.

• tutte e sole le parole dell'alfabeto $\Sigma = \{0, 1\}$ tali che contengono un 1 in terzultima posizione;



L'automa proposto ha un solo stato accettante, e utilizza il non-determinismo sul primo stato per indovinare se, consumato un carattere 1, questo si trova oppure meno in terzultima posizione. Ipotizzando di consumare un 1 ad un certo punto della stringa, e quindi raggiungere lo stato q_2, una serie di due transizioni forza il raggiungimento dello stato finale q_3, contestualmente consumando due caratteri. Per la definizione di NFA data a lezione, la computazione termina con accettazione solo una volta consumati tutti i caratteri, pertanto condizione necessaria per l'accettazione è che il terzultimo carattere sia un 1.

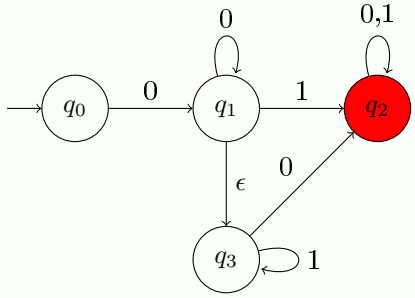
Esercizio 7

Si consideri l'NFA $\mathcal{A} = (\{q_0, q_1, q_2, q_3\}, \Sigma = \{0, 1\}, \delta, q_0, F = \{q_2\})$, dove le seguenti tuple costituiscono δ:

- ➔ $((q_0, 0), q_1)$;
- ➔ $((q_1, 0), q_1)$; $((q_1, 1), q_2)$; $((q_1, \epsilon), q_3)$;
- ➔ $((q_2, 0), q_2)$; $((q_2, 1), q_2)$;
- ➔ $((q_3, 1), q_3)$; $((q_3, 0), q_2)$;

Adesso:

• si disegni l'automa;



- si dica quali delle seguenti parole sono riconosciute:
  - 0;

    No.

  - $w \in 0^*$;

    No.

    $\epsilon \in 0^*$, ma ϵ non è accettata dall'automa.

  - $w \in 00^+$;

    Sì.

  - $w \in 00^*10^*$;

    Sì.

  - $w \in 00^*(10)^*$.

    No.

    $0 \in 00^*(10)^*$, ma 0 non è accettata dall'automa.

- si ricavi una espressione regolare che catturi esattamente il linguaggio dell'automa.

  $0(0^*1 + 0^*1^*0)(0 + 1)^*$, semplificabile in $0^+(1 + 1^*0)(0 + 1)^*$.

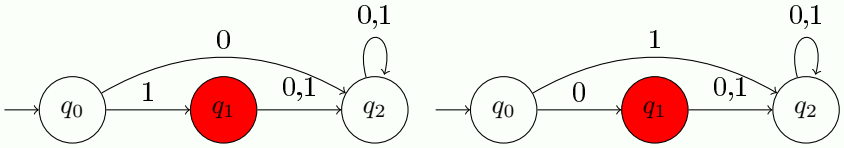
Esercizio 8

Usando il teorema REG $\rightarrow$ NFA, trovare gli automi non-deterministici per le espressioni:

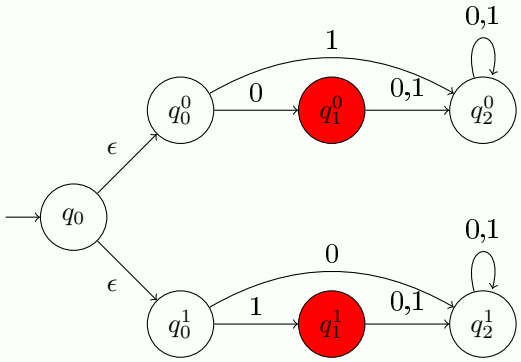
A seconda di ciò che conviene, il teorema, che è definito in maniera induttiva, può essere usato in maniera 'top-down' oppure 'bottom-up'; nel primo caso, l'NFA viene costruito partendo dalla struttura esterna, prima astraendo alcune sue componenti, e poi espandendole; nel secondo caso, viene costruito componendo NFA più semplici.

- $(0+1)^*000(0+1)^*$;

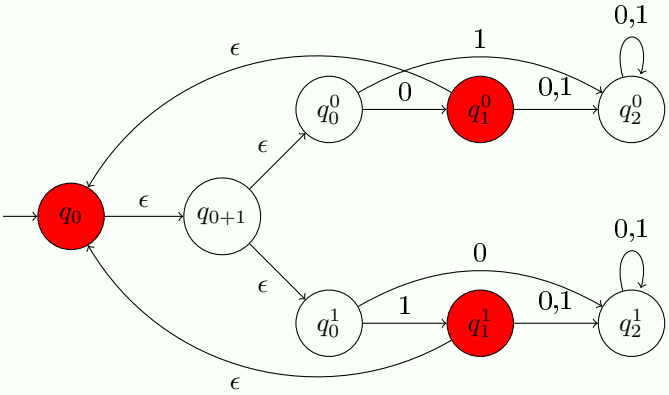
Usando metodo bottom-up, si costruiscono gli automi che riconoscono rispettivamente 0 e 1:



Si costruisce l'automa che riconosce $(0+1)$ per unione:

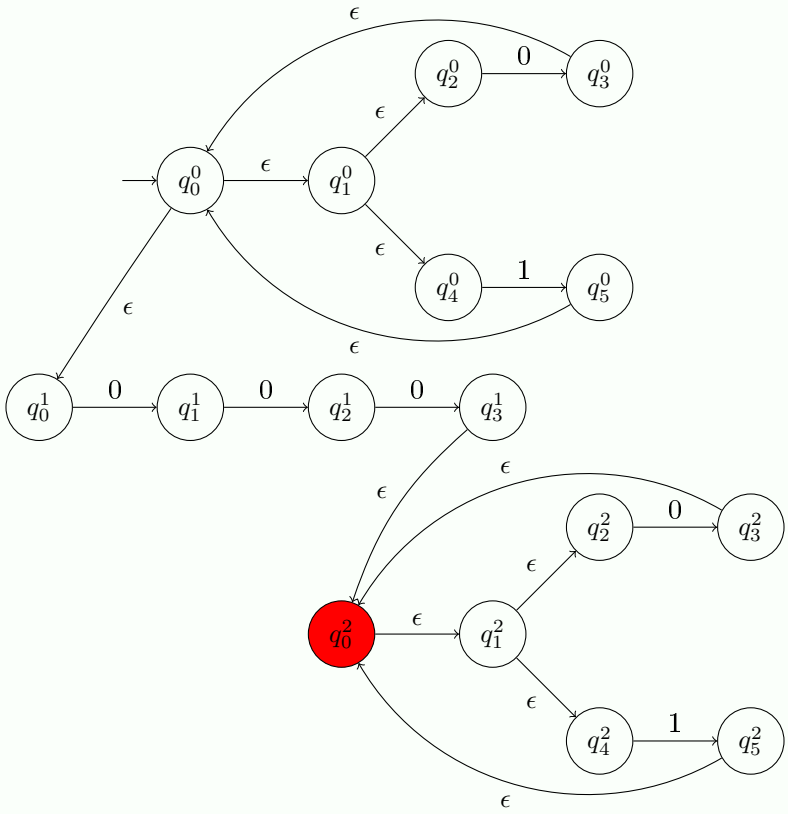


L'automa che riconosce $(0+1)^*$:



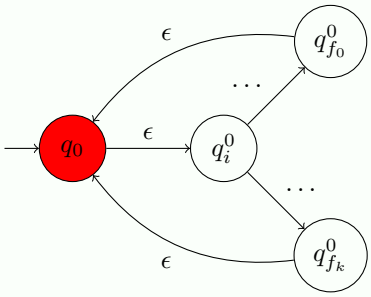
Concatenando due di questi ultimi con un automa (triviale) per 000,

l'automa finale risulta:



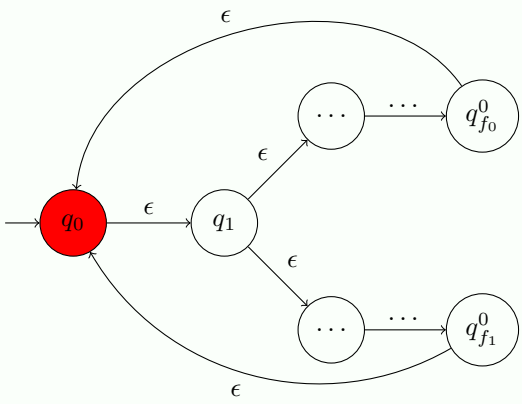
- $(((00)^*(11)) + 01)^*$;

Procedendo in maniera top-down, il livello più esterno dell'espressione presenta una chiusura di Kleene che, per il teorema di equivalenza, dà luogo ad una struttura del tipo:

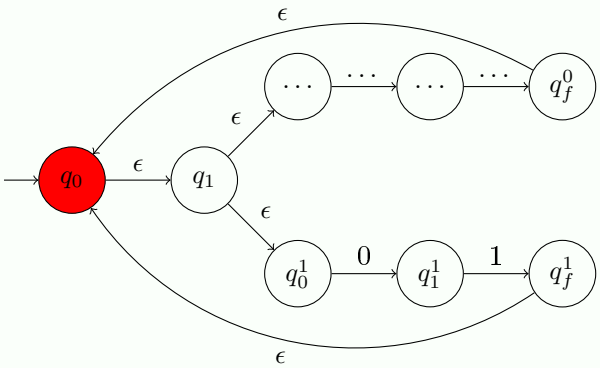


dove q_i^0 è lo stato iniziale di un automa che riconosce l'espressione interna, e $q_{f_1}^0, \ldots, q_{f_k}^0$ sono gli stati finali dello stesso. Scendendo di livello, troviamo una disgiunzione di due termini, $(00)^*(11)$ e 01, pertanto il sotto-automa

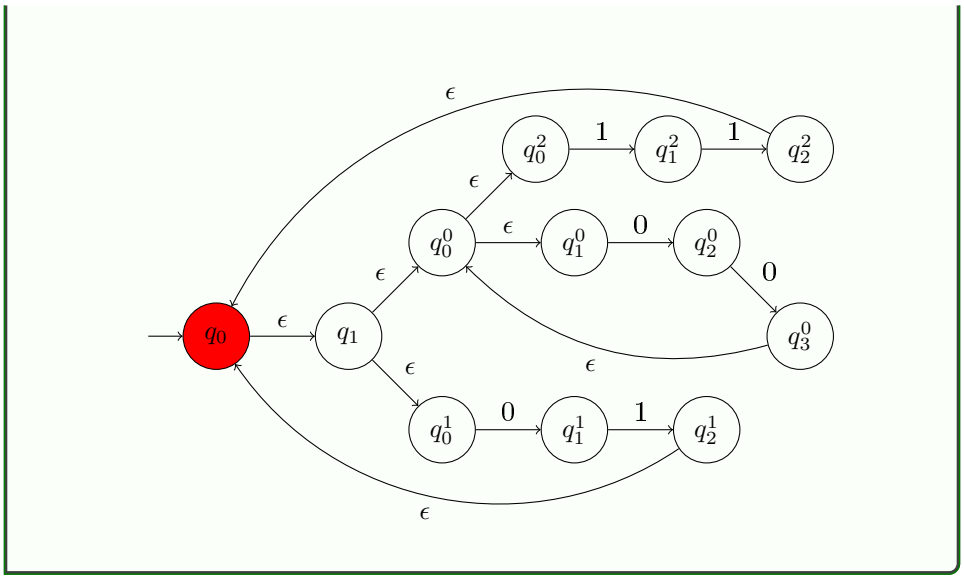
sarà a sua volta unione di due sotto-automi:



Si espandono i sotto-automi per il riconoscimento di $(00)^*(11)$ e 01, che sono due concatenazioni:

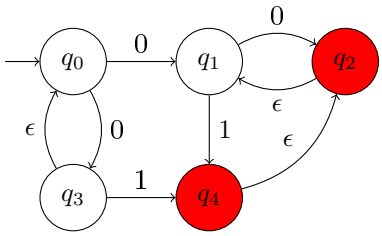


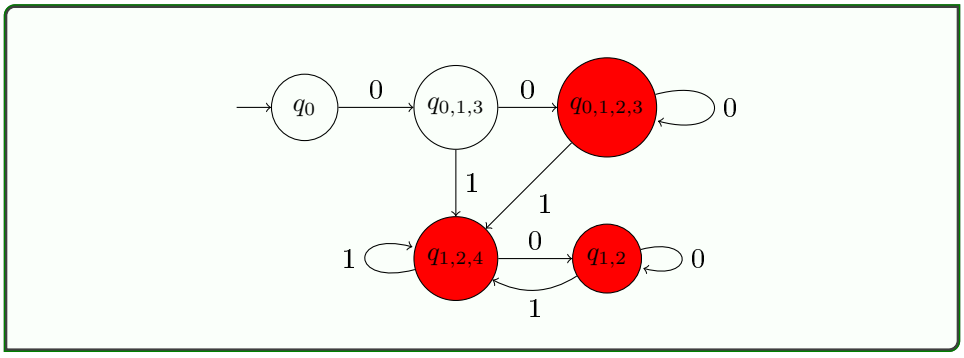
Infine, si considerano gli automi per 00^* e 11. Il primo consiste nella composizione di una lettera e una chiusura di Kleene di una lettera, mentre il secondo in una composizione di lettere. Svolgendo tutte le espansioni rimanenti, si ottiene:



Esercizio 9

Usando il teorema NFA $\rightarrow$ DFA, trovare l'automa deterministico equivalente al seguente NFA:





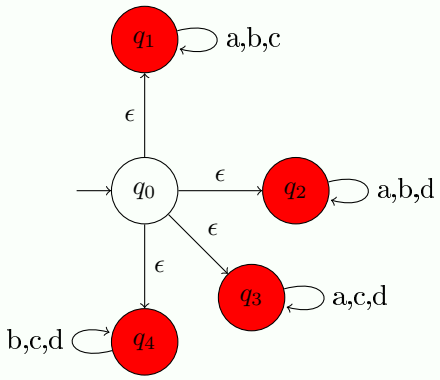
Esercizio 10

Considerare il linguaggio L sull'alfabeto $\{a, b, c, d\}$ dato da tutte e sole le parole nelle quali almeno un simbolo dell'alfabeto è assente. Per esempio, le parole *abcccaca* e *bdadadaddb* sono ammesse, ma la parola *aaabbbcddd* è esclusa. Adesso:

- costruire una espressione regolare per catturare esattamente L;

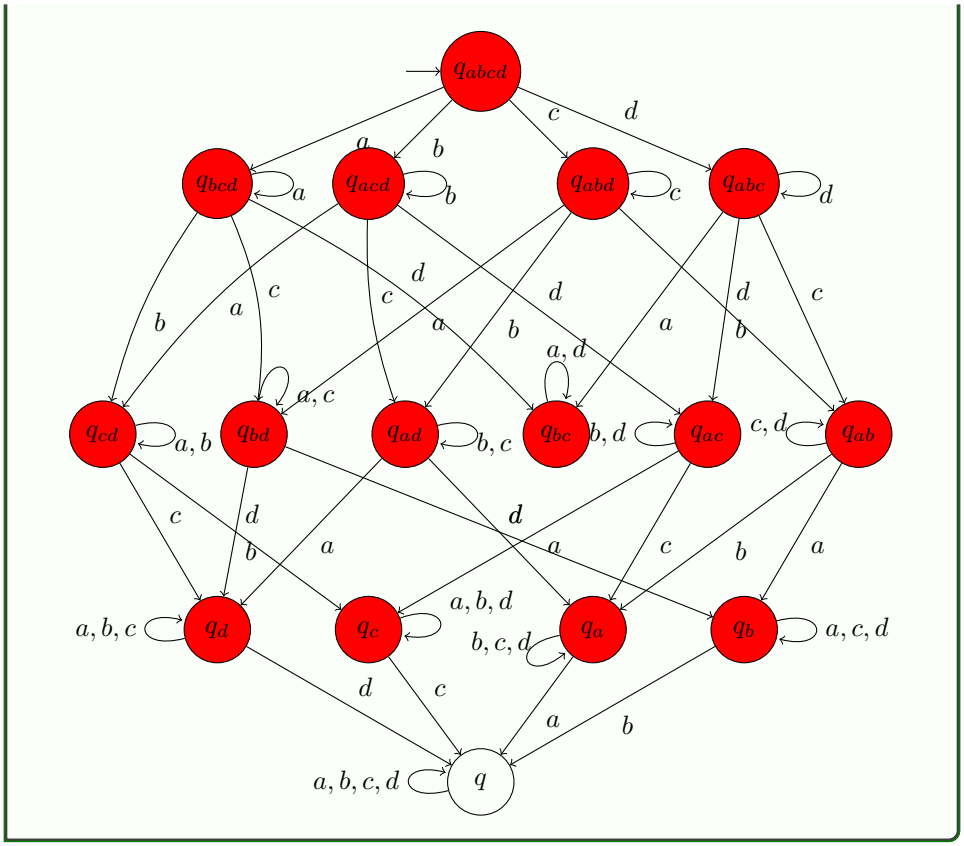
$$(a+b+c)^* + (a+b+d)^* + (a+c+d)^* + (b+c+d)^*$$

- costruire un automa non-deterministico per L;



- Applicando il teorema NFA $\rightarrow$ DFA, costruire un automa deterministico per L. Quanti stati ha?

L'automa risultante ha $2^4 = 16$ stati:

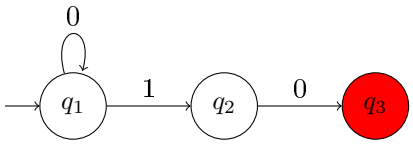


• cosa è possibile dedurre da questo esercizio, riguardo alla relazione in termini di dimensioni di automi deterministici e non-deterministici?

A parità di linguaggio riconosciuto,un automa deterministico può essere *esponenzialmente* più grande di uno non-deterministico. Questo è infatti un caso in cui lo sbalzo esponenziale nel numero di stati avviene *in maniera necessaria*.

Esercizio 11

Usando il teorema DFA → REG, si ricavi l'espressione regolare corrispondente al seguente DFA:



Ricordando che $R(i,j,k)$ è l'insieme di parole che fanno passare A dallo stato q_i allo stato q_j passando, al più, per gli stati $q_1, \ldots, q_k$, calcoliamo $R(0,3,3)$ compilando i valori di R per k crescente.
$k = 0$:

	1	2	3
1	0	1	$\emptyset$
2	$\emptyset$	$\emptyset$	0
3	$\emptyset$	$\emptyset$	$\emptyset$

$k = 1$:

	1	2	3
1	0^+	0^*1	$\emptyset$
2	$\emptyset$	$\emptyset$	0
3	$\emptyset$	$\emptyset$	$\emptyset$

$k = 2$:

	1	2	3
1	0^+	0^*1	0^*10
2	$\emptyset$	$\emptyset$	0
3	$\emptyset$	$\emptyset$	$\emptyset$

$k = 3$:

	1	2	3
1	0^+	0^*1	0^*10
2	$\emptyset$	$\emptyset$	0
3	$\emptyset$	$\emptyset$	$\emptyset$

Quindi, l'espressione regolare ottenuta è 0^*10.

Esercizio 12

Mostrare, usando il Pumping Lemma, che non sono regolari i seguenti linguaggi:

- $\{0^n1^n2^n \mid n \geq 0\}$, $\Sigma = \{0,1,2\}$;

Si consideri una parola $0^p1^p2^p$ per un qualsiasi p; dividendola in tre parti, poiché $|xy| \leq p$, la parte y è limitata ad essere 0^+. In tal caso, per un qualsiasi $i > 1 \mid xy^iz \notin \mathcal{L}$, perché risulta $0^p0^+1^p2^p$.

- $\{www \mid w \in (0+1)^*\}$, $\Sigma = \{0,1\}$;

Si consideri una parola $(0^p1^p)(0^p1^p)(0^p1^p)$ per un qualsiasi p; dato che $|xy| \leq p$, y è sicuramente 0^+. Per qualsiasi $i > 1$, xy^iz è dunque qualcosa del tipo $0^+(0^p1^p)(0^p1^p)(0^p1^p)$, che non appartiene al linguaggio.

- $\{x = y + z \mid x, y, z \in (0 + 1)^*, B(x, y, x)\}$, $\Sigma = \{0, 1, +, =\}$, dove $B(x, y, z)$ significa "x è la somma binaria di y e z";

Si consideri una parola $1^p = 1^p + 0^p$ per un qualsiasi p; dividendola in tre parti, poiché $|xy| \leq p$, la parte y è limitata ad essere 1^+. In tal caso, per un qualsiasi $i > 1 \mid xy^iz \notin \mathcal{L}$, perché risulta $1^+1^p = 1^p + 0^p$ che, non può appartenere al linguaggio contemporaneamente a $1^p = 1^p + 0^p$ (implicherebbe l'uguaglianza $1^p = 1^+1^p$).

- $\{a^{2^n} \mid n \geq 0\}$, $\Sigma = \{a\}$.

Si consideri una parola del linguaggio $w = a^{2^p}$. Chiaramente, $|w| = 2^p$. Siano n, m, l le dimensioni delle parole x, y, z rispettivamente, per qualsiasi divisione si ha $xy^iz = a^n a^{i \cdot m} a^l$. È vero che per ogni divisione esiste almeno un i per cui xy^iz non appartenga al linguaggio? Dimostrare che il linguaggio non sia regolare significa dimostrare che non esistono n, m, l tali che $\forall i, n + i \cdot m + l = 2^q$ per un qualche q, con $n + m + l = 2^p$. Considerando il caso $i = 0$:

$$n + im + l = n + l = 2^r$$

per un qualche r. Mettendo questa equazione a sistema con $n + im + l = 2^q$

$$\begin{cases} n + im + l = 2^q \\ n + l = 2^r \end{cases}$$

si ottiene

$$im = 2^q - 2^r$$

che sappiamo essere pari, e che possiamo dunque scrivere come

$$im = 2k$$

Siccome $n + im + l = 2^q$, si ha che per ogni $i \geq 0$

$$n + i2k + l = 2^q$$
$$i2k + 2^r = 2^q$$
$$i2k = 2^q - 2^r$$

Se questo vale per ogni i, allora deve valere anche per $i = n + l$:

$$(n + l)2k = 2^q - 2^r$$
$$2^r 2k = 2^q - 2^r$$
$$2k = 2^{q-r} - 1$$

Questa equazione è però falsa, poiché uguaglia un numero sicuramente pari $(2k)$ con uno sicuramente dispari $2^{q-r} - 1$. Questo prova che il pumping può generare parole al di fuori del linguaggio.

Esercizio 13

Si mostri che il seguente linguaggio con $\Sigma = \{a, b, c\}$ non è regolare:

$$\{a^i b^j c^k \mid i, j, k \geq 0, i = 1 \rightarrow j = k\}.$$

Se, per assurdo, L fosse regolare, allora dato che REG è chiuso per intersezione, $L \cap ab^*c^* = ab^nc^n$ dovrebbe essere regolare. Se però quest'ultimo fosse regolare, allora dovrebbe esserlo anche la sua intersezione con b^*c^*, ovvero b^nc^n, che invece sappiano essere $\notin REG$; dunque il linguaggio ab^nc^n non è regolare, e pertanto nemmeno L.

Esercizio 14

Trovare l'errore nella seguente dimostrazione. Vogliamo mostrare che $\{0^n1^m\}$ non è regolare. Usiamo il Pumping Lemma, e scegliamo una lunghezza p. Consideriamo la stringa $w = 0^p1^p$. Si vede che la stringa w non può essere divisa in tre parti xyz in maniera da rispettare le condizioni del lemma, e, quindi, il linguggio non può essere regolare.

L'errore sta nell'affermare che la parola $w = 0^p1^p$ non possa essere divisa rispettando le condizioni del lemma. Infatti, L è regolare, la divisione della parola è possibile e il lemma è rispettato; è possibile dare una divisione xyz di una generica parola $w = 0^n1^m, n + m \geq p$ distinguendo due casi:

- Se $n = 0$, la parola è 1^*, e, naturalmente, qualsiasi divisione genera pompaggi che rimangono 1^*, e quindi parte nel linguaggio;
- Se $n \geq 1$, allora la parola si divide in modo tale che $x = \epsilon$ e $y = 0$, e si ha $\forall i \geq 0, xy^iz = 0^{n+i-1}1^m$, che rimane nel linguaggio.

CAPITOLO 2

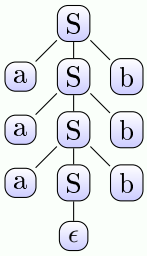
Linguaggi liberi dal contesto

Esercizio 15

Si consideri la grammatica libera

$$S \quad ::= \quad \epsilon \mid aSb$$

e si disegni l'albero di derivazione della parola $aaabbb$.

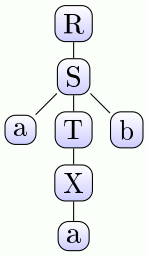


Esercizio 16

Si consideri la grammatica libera

$$\begin{array}{lll} S & ::= & XSX \mid R \\ R & ::= & aTb \mid bTa \\ T & ::= & XTX \mid X \mid \epsilon \\ X & ::= & a \mid b \end{array}$$

e si disegni l'albero di derivazione della parola aab.



Esercizio 17

Si trovi la grammatica libera dal contesto corrispondente ai seguenti linguaggi:

- $\{w \mid w$ contiene più 1 che $0\}$;

$$\begin{cases} S & ::= T1T \\ T & ::= 1T0T \mid T0T1 \mid T1T \mid \epsilon \end{cases}$$

- $\{w\#v \mid w, v \in (0+1)^*,\ w^{-1} \sqsubset v\}$;

$$\begin{cases} S & ::= ST \mid C \\ T & ::= 0 \mid 1 \\ C & ::= 0C0 \mid 1C1 \mid \# \end{cases}$$

- $\{a^i b^j c^k \mid i = j \text{ o } j = k\}$.

$$\begin{cases} S & ::= RT \mid MN \\ R & ::= aRb \mid \epsilon \\ T & ::= cT \mid \epsilon \\ M & ::= aM \mid \epsilon \\ N & ::= bNc \mid \epsilon \end{cases}$$

Esercizio 18

Si converta la seguente grammatica in una equivalente in forma di Chomsky:

$$\begin{array}{lll} S & ::= & XSX \mid R \\ R & ::= & aTb \mid bTa \\ T & ::= & XTX \mid X \mid \epsilon \\ X & ::= & a \mid b \end{array}$$

Per prima cosa, si esplicitano le disgiunzioni:

$$\begin{array}{lll} S ::= XSX, & S ::= R, & \\ R ::= aTb, & R ::= bTa, & \\ T ::= XTX, & T ::= X, & T ::= \epsilon, \\ X ::= a, & X ::= b & \end{array}$$

Ora, T ha una ϵ-produzione non iniziale, quindi la si rimuove introducendo

nuove regole:

$$\begin{array}{llll} S ::= XSX, & S ::= R, & & \\ R ::= aTb, & R ::= bTa, & R ::= ab, & R ::= ba, \\ T ::= XTX, & T ::= X, & T ::= XX, & \\ X ::= a, & X ::= b & & \end{array}$$

Si applica la funzione Ch, partendo dalla prima regola:

$$\mathrm{Ch}(S ::= XSX) = \{S ::= XB_1,\ B_1 ::= SX\}$$

La seconda regola è già in forma normale. Si procede con la terza:

$$\begin{aligned} \mathrm{Ch}(R ::= aTb) &= \{S ::= A_1B_2,\ A_1 ::= a\} \cup \mathrm{Ch}(B_2 ::= Tb) \\ &= \{S ::= A_1B_2,\ A_1 ::= a\} \cup \{B_2 ::= TB_3,\ B_3 ::= b\} \\ &= \{S ::= A_1B_2,\ A_1 ::= a,\ B_2 ::= TB_3,\ B_3 ::= b\} \end{aligned}$$

Continuando con le produzioni rimanenti, rimuovendo regole duplicate, e operando opportune rinominazioni per ottenere una grammatica più chiara, si ottiene:

$$G = \begin{cases} S & ::= XB_1 \mid A_1B_2 \mid A_2B_3 \mid A_1A_2 \mid A_2A_1 \\ T & ::= XB_4 \mid a \mid b \mid XX \\ X & ::= a \mid b \\ A_1 & ::= a \\ A_2 & ::= b \\ B_1 & ::= SX \\ B_2 & ::= TA_2 \\ B_3 & ::= TA_1 \\ B_4 & ::= TX \end{cases}$$

Esercizio 19

Dimostrare che se G è una grammatica libera in forma di Chomsky e $w \in L(G)$, allora esiste una derivazione di w in G di al più $2 \cdot |w| - 1$ passi.

Una parola generata da una grammatica libera può essere rappresentata come la frontiera dell'albero di derivazione, e tutte le grammatiche di Chomsky danno luogo ad un albero che è, nel caso peggiore, un albero che è binario fino al penultimo livello, e dove l'ultimo livello è unario. Infatti, da un simbolo non terminale la produzione in simbolo terminale non genera un branch, ma una singola foglia, e pertanto gli ultimi due livelli avranno lo stesso numero di foglie. Per una parola di lunghezza $|w|$, l'albero di derivazione avrà quindi $|w|$ foglie e altezza massima $\lceil \log_2 |w| \rceil + 1$, dove il $+1$ è dovuto alla 'anomalia' che l'ultimo livello presenta rispetto ad un normale albero binario. Il numero di passi di derivazione, pari al numero di nodi interni dell'albero, è pari al numero totale di nodi in un albero binario di altezza $\lceil \log_2 |w| \rceil$, ovvero $2^{\lceil \log_2 |w| \rceil} - 1 \leq 2^{\log_2 |w| + 1} - 1 = 2 \cdot 2^{\log_2 |w|} - 1 = 2 \cdot |w| - 1$.

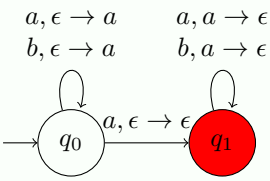
Esercizio 20

Si consideri il PDA $\mathcal{A} = (\{q_0, q_1\}, \Sigma = \{a, b\}, \Gamma = \{a\}, \delta, F = \{q_1\})$, dove le seguenti tuple costituiscono δ:

- ➔ $((q_0, a, \epsilon), (q_0, a))$;
- ➔ $((q_0, b, \epsilon), (q_0, a))$;
- ➔ $((q_0, a, \epsilon), (q_1, \epsilon))$;
- ➔ $((q_1, a, a), (q_1, \epsilon))$;
- ➔ $((q_1, b, a), (q_1, \epsilon))$.

Adesso:

- si disegni l'automa;



- si dica quali delle seguenti parole sono riconosciute:
  - aba;

No.

  - aa;

No.

  - baa;

Sì.

  - bab;

Sì.

  - abb.

No.

- si spieghi in linguaggio naturale la struttura delle parole accettate;

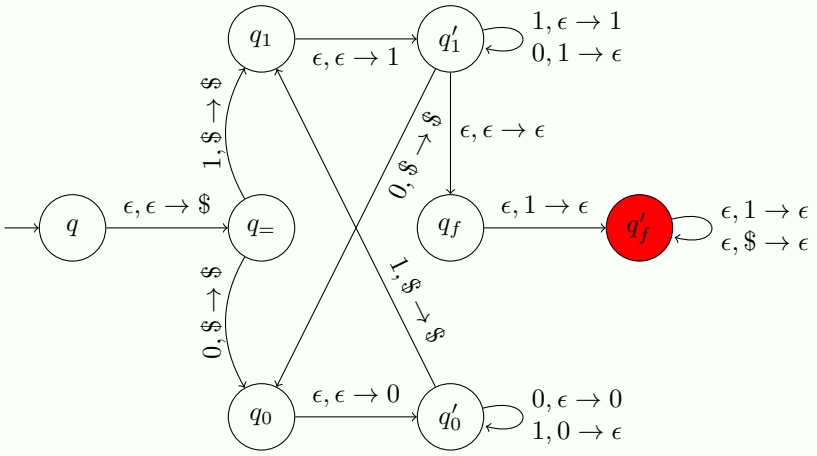
L'automa accetta parole in $(a+b)^+$ di lunghezza dispari e con una a come lettera centrale.

Necessariamente, le parole accettate devono contenere una a, altrimenti la computazione non raggiunge mai lo stato accettante. Inoltre, deve essere vero che il numero di caratteri a destra e a sinistra della a (che può essere una qualsiasi nella parola, a causa del non-determinismo) deve essere uguale, altrimenti non si raggiunge mai lo stato accettante con una pila vuota. Pertanto, l'automa accetta parole in $(a+b)^+$ di lunghezza dispari con una a centrale.

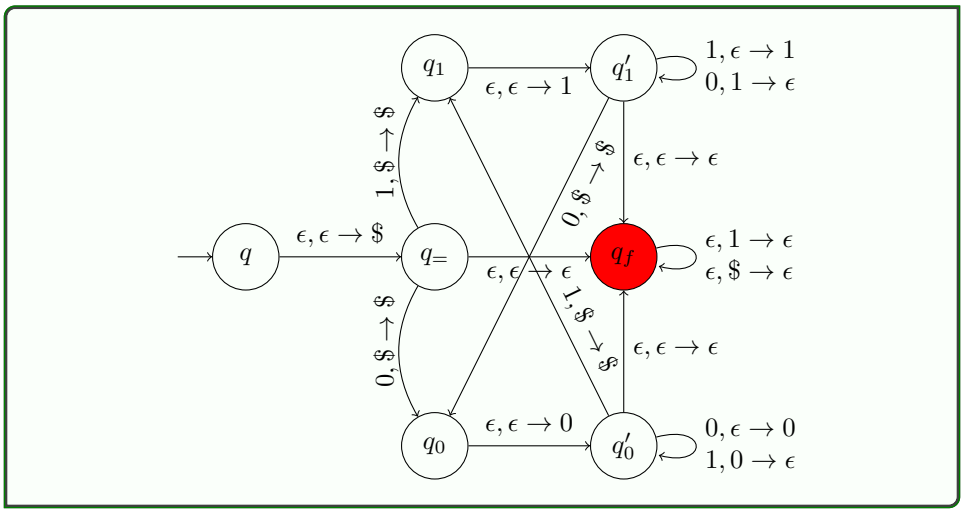
Esercizio 21

Si produca un PDA per ognuno dei seguenti linguaggi:

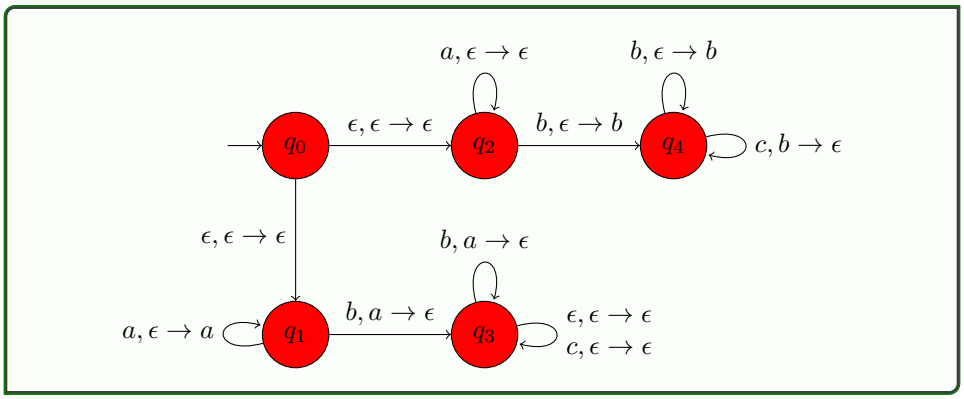
- $\{w \mid w$ contiene più 1 che $0\}$



- $\{w \mid w$ contiene almeno tanti 1 quanti $0\}$

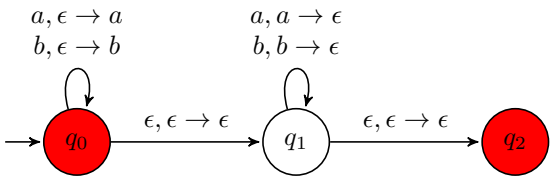


- $\{a^i b^j c^k \mid i = j \text{ o } j = k\}$.

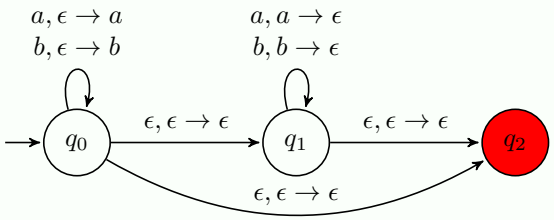


Esercizio 22

Usando il teorema PDA → CFG, si trovi una grammatica per il seguente automa a pila, e si caratterizzi il linguaggio riconosciuto:



Per applicare il teorema visto a lezione, bisogna prima trasformare l'automa in modo che abbia un solo stato finale. Questo si ottiene legando tutti gli stati finali ad uno stato finale unico attraverso ϵ-transizioni. Un automa equivalente a quello risultate è:



Ora, applicando il teorema e rimuovendo alcune regole non necessarie, la grammatica risultante è

$$G = \begin{cases} S & ::= A_{02} \\ A_{01} & ::= aA_{01}a \mid bA_{01}b \\ A_{01} & ::= \epsilon \\ A_{02} & ::= \epsilon \\ A_{12} & ::= \epsilon \\ A_{02} & ::= A_{00}A_{02} \mid A_{01}A_{12} \mid A_{02}A_{22} \\ A_{00} & ::= \epsilon \\ A_{11} & ::= \epsilon \\ A_{22} & ::= \epsilon \end{cases}$$

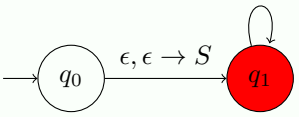
e il linguaggio riconosciuto è $ww^{-1}, w \in \{a, b\}^*$.

Esercizio 23

Usando il teorema CFG $\to$ PDA, per ciascuna delle seguenti grammatiche, si produca un PDA che lo riconosca i seguenti linguaggi, e si mostrino i passi computazionali per riconoscere la stringa w specificata:

- $G = \begin{cases} S & ::= ST \mid C \\ C & ::= 0C0 \mid 1C1 \mid \# \,, \\ T & ::= 0 \mid 1 \end{cases} \qquad w = 0\#01$

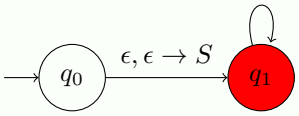
$$\epsilon, S \to ST$$
$$\epsilon, S \to C$$
$$\epsilon, C \to 0C0$$
$$\epsilon, C \to 1C1$$
$$\epsilon, C \to \#$$
$$\epsilon, T \to 0$$
$$\epsilon, T \to 1$$
$$0, 0 \to \epsilon$$
$$1, 1 \to \epsilon$$
$$\#, \# \to \epsilon$$



La porzione di parola riconosciuta, e il contenuto della pila durante la computazione sono le seguenti: $(\epsilon, S) \to (\epsilon, ST) \to (\epsilon, CT) \to (\epsilon, 0C0T) \to (0, C0T) \to (0, \#0T) \to (0\#, 0T) \to (0\#0, T) \to (0\#0, T) \to (0\#0, 1) \to (0\#01, \sqcup)$

- $G = \begin{cases} S & ::= T1T \\ T & ::= 1T0T \mid T0T1 \mid T1T \mid \epsilon \end{cases}, \qquad w = 01101$

$$\epsilon, S \to T1T$$
$$\epsilon, T \to 1T0T$$
$$\epsilon, T \to T0T1$$
$$\epsilon, T \to T1T$$
$$\epsilon, T \to \epsilon$$
$$0, 0 \to \epsilon$$
$$1, 1 \to \epsilon$$



La porzione di parola riconosciuta, e il contenuto della pila durante la computazione sono le seguenti: $(\epsilon, S) \to (\epsilon, T1T) \to (\epsilon, 01T) \to (0, 1T) \to (01, T) \to (01, T1T) \to (01, \epsilon 1T) \to (01, 1T) \to (011, T) \to (011, T0T1) \to (011, \epsilon 0T1) \to (011, 0T1) \to (0110, \epsilon 1) \to (0110, 1) \to (01101, \sqcup)$

Esercizio 24

Mostrare che i seguenti linguaggi, con $\Sigma = \{a, b, c\}$, non sono liberi dal contesto:

- $\{ww \mid w \in \Sigma^*\}$; in questo caso, si faccia un confronto con il caso del linguaggio $\{ww^{-1} \mid w \in \Sigma^*\}$, che è libero dal contesto ma non regolare: qual è la differenza tra le due dimostrazioni?

Si consideri la parola $ww = a^p b^p c^p a^p b^p c^p$, e si applichi il Pumping Lemma per linguaggi liberi dal contesto. Considerando i modi in cui potrebbe essere divisa in $uvxyz$ e pompata a $uv^i xy^i z$, l'unica divisione possibile sarebbe quella nella quale v ed y sono tali che $v = y$, e che entrambe catturano la stessa parte delle due sotto-parole w. Tuttavia, $|w| = 3p$, e dunque non esiste nessuna divisione che rispetti $(|vy| > 0)$ e $(|vxy| \leq p)$. Concludiamo, quindi, che il linguaggio non è libero dal contesto. La differenza fra riconoscere ww e ww^{-1} sta nel fatto che (sebbene solo grazie al non-determinismo) possiamo rendere lo stack simmetrico al nastro, e poi semplicemente fare un match degli elementi.

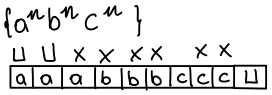
- $\{www \mid w \in \Sigma^*\}$;

Per un ragionamento simile al caso precedente, non può esistere nessuna divisione che rispetti il Pumping Lemma. Consideriamo la parola $www = a^p b^p c^p a^p b^p c^p a^p b^p c^p$. Considerando i modi in cui potrebbe essere divisa in $uvxyz$ e pompata a $uv^i xy^i z$, l'unica divisione possibile sarebbe quella nella quale v ed y sono tali che $v = y$, ma pure se catturassero la stessa parte di due sotto-parole w, la terza sotto-parola w non verrebbe pompata, rimanendo invariata, il che genererebbe parole pompate al di fuori del linguaggio. Concludiamo, quindi, che il linguaggio non è libero dal contesto.

Esercizio 25

Siano L_{CFG} ed L_{REG} rispettivamente un linguaggio regolare ed uno libero dal contesto; si dimostri, tramite la costruzione di un automa apposito, che l'intersezione $L_{\text{CFG}} \cap L_{\text{REG}}$ è libera dal contesto.

Assumiamo per semplicità che L_{CFG} e L_{REG} siano definiti sullo stesso alfabeto Σ. Siano $\mathcal{A}_{\text{CFG}} = (Q_{\text{CFG}}, \Sigma, \Gamma, \delta_{\text{CFG}}, F_{\text{CFG}})$ e $\mathcal{A}_{\text{REG}} = (Q_{\text{REG}}, \Sigma, \delta_{\text{REG}}, q_i \in Q_{\text{REG}}, F_{\text{REG}})$, rispettivamente, un PDA per L_{CFG} e un DFA per L_{REG}. Costruiamo il PDA *prodotto* $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, F)$, tale che $Q = Q_{\text{CFG}} \times Q_{\text{REG}}$, $F = F_{\text{CFG}} \times F_{\text{REG}}$, e con δ è tale che $(((q_{c1}, q_{r1}), \gamma_1, \gamma_2), ((q_{c2}, q_{r2}), \gamma_3)) \in \delta$ per ogni $q_{r1}, q_{r2} \in Q_{\text{REG}}, q_{c1}, q_{c2} \in Q_{\text{CFG}}, \gamma_1 \in \Sigma, \gamma_2, \gamma_3 \in \Gamma$, con $((q_{c1}, \gamma_1, \gamma_2), (q_{c2}, \gamma_3)) \in \delta_{\text{CFG}}$ e $\delta_{\text{REG}}((q_{r1}, \gamma_1), q_{r2})$. Questo permette all'automa PDA di eseguire i due automi contemporaneamente, tenendo traccia, al contempo, *(i)* dei caratteri consumati dalle transizioni del DFA, *(ii)* di quelli consumati dall'automa PDA, *(iii)* dello stato della sua pila.



CAPITOLO 3

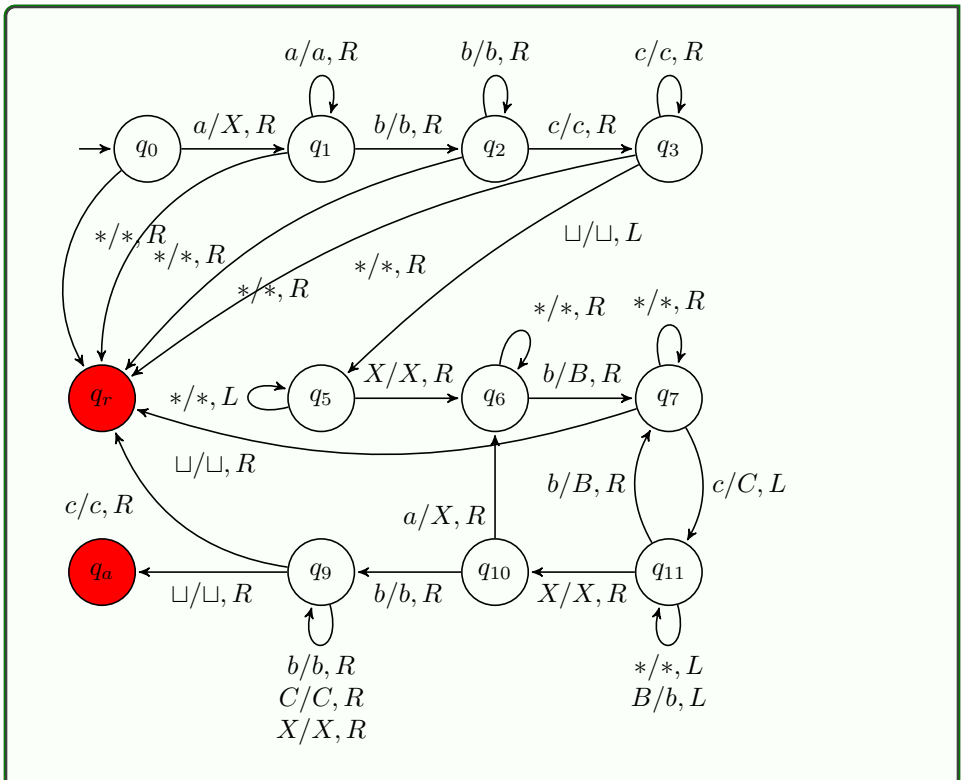
Macchine di Turing

Per snellire i diagrammi che mostrano Macchine di Turing, in questo capitolo indichiamo con $*/*$ un tipo di transizione che *legge qualunque carattere dal nastro lasciandolo invariato*, e che, rispetto alle altre transizioni, ha sempre priorità minore.

Esercizio 26

Si consideri il linguaggio $L = \{a^i b^j c^k \mid i \times j = k, i, j, k \geq 1\}$ e:

• si costruisca una TM per riconoscerlo, rappresentandola con un diagramma di stati;



La TM proposta esegue l'algoritmo:
- metti una X sulla prima a, e scorri il nastro controllando che il resto dell'input sia del tipo b^+c^+, altrimenti RIFIUTA;
- torna alla prima b presente sul nastro;
- finchè esistono ancora b, sostituisci alternativamente una b con B e una c con C (se per una certa b non ci sono più c disponibili, RIFIUTA);
- se non ci sono più a disponibili, allora, se non ci sono neppure più c disponibili, ACCETTA, altrimenti RIFIUTA;
- ripristina tutte le b;
- ripeti dal passo 2.

- si diano due configurazioni C_1, C_2 qualsiasi tali che $C_1 \rightsquigarrow C_2$.

$C_1 = Xaaq_6bccc$ e $C_2 = XaaBq_7ccc$, legate dalla transizione $\delta(q_6, b) = (q_7, B, R)$.

- **Esercizio 27**

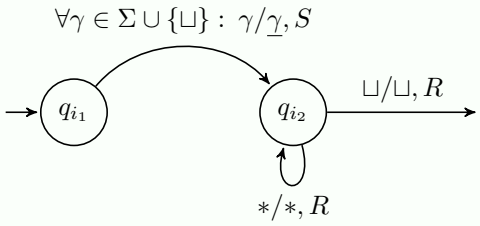
Considerare nuovamente il Teorema k-TM $\rightarrow$ TM del blocco che introduce le macchine di Turing. Riformulare, con sufficiente livello di dettaglio, la sua dimostrazione, ed arricchirla con esempi di trasformazione.

La tesi è che per ogni k-TM $\mathcal{M}$ esiste una TM $\mathcal{M}'$ che riconosce precisamente lo stesso linguaggio. Una macchina di Turing con k nastri differisce da una a nastro singolo, oltre che dal numero di nastri, dal fatto che la funzione di transizione dipende da ogni carattere sotto ogni testina, e può causare la scrittura e lo spostamento sia a destra che a sinistra di ciascuna delle testine in un solo passo di computazione. Intuitivamente, essendo un nastro isomorfo ad $\mathbb{N}$, k nastri sono isomorfi a $\mathbb{N}^k$, e dato che $\forall k \in \mathbb{N} : |\mathbb{N}| = |\mathbb{N}^k|$, è ragionevole ipotizzare che questi due modelli siano in effetti equivalenti, e dunque possiamo provare a costruire una simulazione di una k-TM su una TM. In particolare, per simulare una k-TM, ci risulta comodo usare una Stay-TM, ovvero una TM che ha la possibilità ad ogni passo computazionale di non muoversi, oltre che di muoversi a destra o a sinistra.
Per effettuare la simulazione, facciamo in modo che il nastro di $\mathcal{M}'$ contenga, ad ogni passo computazionale, una codifica del contenuto dei k nastri, arricchita in modo da tenere anche traccia della posizione delle k testine. Sia Σ l'alfabeto di $\mathcal{M}$, l'alfabeto di $\mathcal{M}'$ include tutti i simboli in Σ; in più, per ogni simbolo $\gamma \in \Sigma \cup \{\sqcup\}$, equipaggiamo il nuovo alfabeto con un relativo simbolo marcato $\underline{\gamma}$, che useremo per segnalare la presenza della testina su un certo simbolo sul nastro. Inoltre, aggiungiamo i simboli \$ e $\lhd$ (che supponiamo non esistere nell'alfabeto di partenza); il primo verrà usato come separatore del contenuto di ciascuno dei k nastri; il secondo verrà posto a

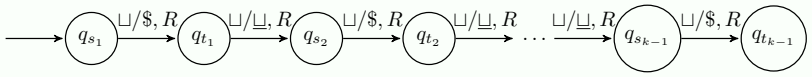
fine nastro per indicare la fine del contenuto del nastro. Si noti che quest'ultimo simbolo è necessario, poiché i simboli $\sqcup$ possono apparire anche all'interno dei k-nastri, e quindi della loro rappresentazione, dunque nessuna sequenza finita di caratteri $\sqcup$ può essere usata come indicatore della fine del nastro reale.
A inizio computazione, troviamo la stringa in input $w = w_1 \dots w_n$ scritta sul nastro, ed iniziamo la computazione generando sul nastro di $\mathcal{M}'$ la situazione:

$$\underline{w}_1 w_2 \dots w_n \$ \underline{\sqcup} \$ \underline{\sqcup} \dots \$ \underline{\sqcup} \triangleleft$$

La sostituzione di w_1 con la sua controparte sottolineata, utile ad indicare la posizione della testina del primo nastro, si può ottenere iniziando la computazione su uno stato che ha sole transizioni di tipo $\gamma/\underline{\gamma}, S$ ad un unico secondo stato, dove $\gamma \in \Sigma \cup \{\sqcup\}$. Da questo stato, poi scorriamo tutto l'input per preparare la configurazione iniziale:



Successivamente, $2 \cdot k$ stati possono essere usati per scrivere $k-1$ volte $\$\underline{\sqcup}$ dopo l'input, uno stato per scrivere $\triangleleft$ a fine nastro. Per ogni $i \in [1, k-1]$, aggiungere stati e transizioni:

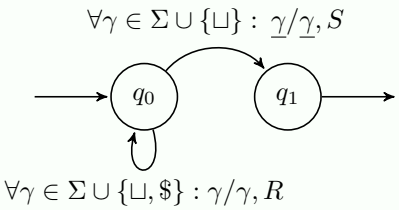


Poniamo infine uno stato per far tornare la testina alla posizione 0 (usando uno stratagemma simile a quello usato per q_{i_2}.
Adesso che abbiamo completato il setup, simuliamo ogni ogni passo computazionale di $\mathcal{M}$. Partiamo dal grafo di transizione della $k-$TM, quindi ereditando ciascuno degli stati originali. Formalmente, ogni transizione in $\mathcal{M}$ è di tipo:

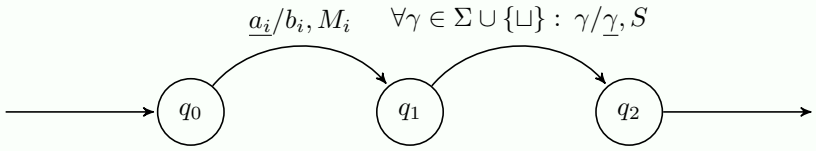
$$(q, a_1, a_2, \dots, a_k) \to (q', b_1, b_2, \dots, b_k, M_1, \dots, M_k)$$

Per simulare ciascuna transizione di questo tipo, avremo dunque bisogno di k stati aggiuntivi $q_{a_1}, q_{a_1 a_2}, \dots, q_{a_1 \dots a_k}$ per leggere cascun carattere $\overline{a}_i$ dalla testina del i-esimo nastro, e di altrettanti stati per scrivere b_i in corrispondenza di ciascuna testina, e muovere la i-esima testina secondo M_i. Un dettaglio importante è che, per muoversi tra uno stato q e lo stato q_{a_1}, così come per muoversi da q_{a_i} a $q_{a_{i+1}}$, necessitiamo prima di spostare la testina reale sul prossimo carattere sottolineato (ipotizzando di iniziare con la

testina reale in prima posizione) e questo si può fare attraverso il seguente gadget:



Una volta giunti allo stato $q_{a_1 \dots a_k}$, facciam tornare la testina alla posizione 0, ed usiamo uno stratagemma simile per simulare la scrittura e il movimento delle testine sui k nastri, ovvero attraverso $O(k)$ stati aggiuntivi: $q_{a_1 \dots a_k | b_1 M_1}, q_{a_1 \dots a_k | b_1 M_1 b_2 M_2}, \dots, q_{a_1 \dots a_k | b_1 M_1 \dots b_k M_k}$. Si noti che anche questi stati, nel grafo di transizione, sono intervallati dall'inserimento del gadget che scorre fino a trovare il prossimo carattere sottolineato. Immaginando di trovarci nel punto della i-esima testina, scrittura sul i-esimo nastro si può simulare semplicemente scrivendo il carattere b_i nella cella corrente, mentre lo spostamento della i-esima testina si può ottenere spostando la testina reale e sostituendo il carattere sotto la testina con la sua versione sottolineata:



A questo proposito si pone un problema: lo spostamento di ciascuna testina può portare la stessa fuori dal contesto del nastro simulato (ovvero, su un simbolo \$). Ogni volta che questo accade, vorremmo estendere la dimensione del nastro corrente in modo che la testina sia ancora al suo interno. Questo si può ottenere spostando tutto ciò che sta a destra della testina avanti di una casella sul nastro, ed introducendo in corrispondenza della testina un $\sqcup$ (che poi viene sostituito con $\underline{\sqcup}$ dalla transizione da q_1 a q_2 nel gadget appena mostrato). Più precisamente, serve un complesso di stati che copino a destra di una posizione tutti i simboli, da quello corrente, fino all'ultimo alla fine del nastro (incluso $\lhd$), riportino indietro la testina fino al simbolo \$ appena inserito (riconoscibile perché è l'unico \$ su tutto il nastro che precede un altro \$), e lo sostituiscano con $\underline{\sqcup}$; la costruzione di tale gadget è lasciata come esercizio. Infine, si riporta la testina alla posizione 0, così che la simulazione delle prossime computazioni può essere eseguita allo stesso modo.

Esercizio 28

Considerare nuovamente il Teorema Left-TM $\rightarrow$ TM del blocco che introduce le macchine di Turing. Riformulare, con sufficiente livello di dettaglio, la sua

dimostrazione.

Sebbene si possa simulare in maniera diretta un nastro infinito a sinistra con due nastri standard, è anche possibile simularlo su un singolo nastro standard, simulando le caselle positive su quelle caselle pari, e le caselle dispari su quelle negative; questo approccio fa leva sul noto mapping che è possibile fare da $\mathbb{Q}$ a $\mathbb{N}$. Supponendo che esista qualche modo per capire se la testina ha raggiunto l'estremo sinistro del nastro, immaginiamo un meccanismo che ricordi se la testina si trova su caselle pari o dispari, ovvero su caselle rispettivamente positive o negative della Left-TM. Dunque, quando la testina si trova su caselle pari, ogni transizione che prevede uno spostamento a destra (o a sinistra) sarà simulata con *due* transizioni che spostano la testina a destra (o a sinistra). Viceversa, quando la testina si trova su caselle dispari, ogni trasizione che prevede spostarsi a destra (o a sinistra) sarà simulata con *due* transizioni che spostano la testina a sinistra (o a destra). È da notare l'inversione delle direzioni, data dal fatto che la parte di nastro a sinistra dell'origine viene simulata in senso inverso.

Esercizio 29

Considerare il verso RAM $\rightarrow$ TM del Teorema RAM = TM del blocco che introduce le macchine di Turing. Riformulare, con sufficiente livello di dettaglio, la sua dimostrazione.

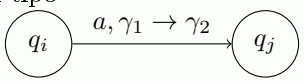
Una TM può essere vista come una RAM dotata di:
- due aree di memoria infinite, separate e indicizzate da numeri naturali, contenenti rispettivamente le istruzioni da eseguire, e i dati da elaborare;
- due registri, contenenti rispettivamente l'indirizzo di una zona di interesse nella memoria di programma ('program counter'), e l'indirizzo di una zona di interesse nella memoria dati ('data pointer'), entrambi inizializzati a '0' (si noti che i due registri codificano rispettivamente le testine sul nastro e sullo stato corrente della TM);
- una sola istruzione con quattro argomenti:
  - due valori (simboli dell'alfabeto);
  - un booleano (sinistra/destra);
  - un indirizzo della memoria di programma (stato),

  che esegue le seguenti operazioni:
  - legge il valore contenuto nella locazione puntata dal data pointer;
  - confronta il valore letto con il primo argomento: se sono diversi, incrementa il program counter e termina, altrimenti prosegue;
  - sovrascrive la locazione puntata del data pointer con il secondo argomento;
  - incrementa il data pointer se il terzo argomento è vero, lo decrementa se è falso;
  - sovrascrive il program counter con il quarto argomento.

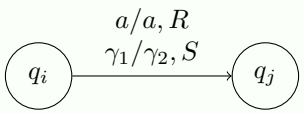
Esercizio 30

Considerare nuovamente il Teorema PDA $\subset$ DEC del blocco che introduce le macchine di Turing. Completare, con lo stesso livello di dettaglio, la sua dimostrazione, ed arricchirla con esempi di trasformazione.

Sappiamo che esiste almeno un linguaggio DEC che non è PDA; ora proviamo che PDA $\subseteq$ DEC, mostrando che (i) per ogni PDA, ne esiste sempre uno che riconosce lo stesso linguaggio e termina sempre, e (ii) per ogni PDA, esiste una macchina di Turing che lo simula. Questi due fatti, insieme, dimostrano che per ogni PDA esiste una macchina di Turing equivalente (e terminante) che decide lo stesso linguaggio.
Un PDA può non terminare la computazione se esiste nel suo grafo di transizione un loop di ϵ-transizioni tra stati non finali, che permette al PDA di ciclare indefinitamente senza consumare caratteri in input. Mostriamo, però, che per ogni automa ne possiamo costruire uno equivalente che non ha ϵ-cicli, e che quindi sicuramente termina. Chiamiamo (q, x) una situazione di *looping* per un PDA, dove q è uno stato del PDA, e x è un simbolo della sua pila, se quando la computazione parte da q con x in cima alla pila, l'automa non legge nulla al di sotto di x e non legge nessun simbolo dall'input. Se la computazione raggiunge uno stato accettante, chiamiamo questa situazione accettante, mentre la chiamiamo rifiutante in caso contrario. Questa situazione è determinabile in tempo finito, e dunque, per ogni situazione di looping accettante aggiungiamo una ϵ-transizione ad un nuovo stato accettante, e per ogni situazione di looping rifiutante aggiungiamo una ϵ-transizione ad un nuovo stato 'cieco' (ovvero che porta l'automa a rifiutare).
Ora consideriamo un PDA qualunque, e costruiamo una macchina di Turing equivalente. In particolare, usiamo una Stay $-$ TM a due nastri, il primo dei quali useremo per scorrere l'input, ed il secondo per simulare la pila. A inizio computazione, l'input si trova sul primo nastro, mentre il secondo nastro è vuoto, il che è compatibile con le condizioni iniziali di un PDA. Partiamo dal grafo di transizione originale, ereditando, quindi, ciascuno degli stati originali e, considerando ciascun tipo di PDA-transizione, la trasformiamo in una transizione Stay-2-TM. Distinguiamo tre tipi di transizioni: quelle che aggiungono un simbolo alla pila, quelle che tolgono un simbolo dalla pila, e quelle che fanno entrambe le cose. Come idea generale, simuliamo una PDA-transizione del tipo

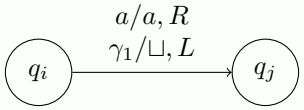


come transizioni Stay-2-TM del tipo

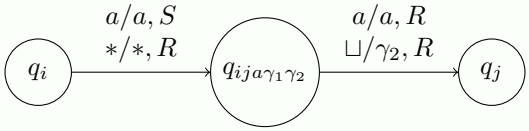


Si noti che questa traduzione presenta alcuni problemi. Innanzitutto, non va bene nel caso di ϵ-transizioni, poiché le TM non permettono tali transizioni. Inoltre, a inizio computazione la testina sulla pila è posizionata su un $\sqcup$,

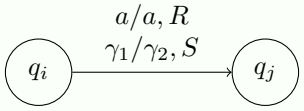
che non appare sicuramente negli alfabeti originali Σ e Γ, e dunque nessuna transizione prodotta in questo modo dal primo stato verrà usata.
Decidiamo allora che la testina della pila va mantenuta, prima e dopo di ogni simulazione di PDA-transizione, sull'ultimo simbolo della pila. Questo ci forza, nel simulare la pila, ad avere sempre almeno un simbolo sul secondo nastro, dunque decidiamo ad inizio computazione di scrivere un simbolo \$ (dove \$ non è presente nell'alfabeto) sul secondo nastro tramite una transizione $\sqcup/\$, S$. A questo punto, ogni transizione può essere simulata tramite transizioni Stay-2-TM che utilizzano al più due passi; in particolare, le transizioni che eseguono un pop ($\gamma_1 = \epsilon, \gamma_2 \neq \epsilon$) possono essere tradotte come:



Le transizioni che eseguono un push ($\gamma_1 \neq \epsilon, \gamma_2 = \epsilon$) possono essere tradotte come:



Le transizioni che eseguono pop e push ($\gamma_1 \neq \epsilon, \gamma_2 \neq \epsilon$) possono essere tradotte come:



Un ultimo dettaglio è dato dal fatto che un PDA accetta solamente a pila vuota, e avendo letto tutto l'input. Pertanto, per ognuno degli stati accettanti della TM prodotta, creiamo due stati, uno al quale arrivano le transizioni in entrata, e uno al quale arrivano le transizioni in uscita. Tra i due stati, poniamo una routine che controlla esattamente se l'input è stato letto completamente, e se il secondo nastro contiene solo \$; in caso contrario, riportiamo le due testine dove stavano prima.

Esercizio 31

Un Automa a Coda è una tupla

$$\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, F)$$

dove Q è l'insieme degli stati, Σ è l'alfabeto (dell'input), Γ è l'alfabeto della coda, δ è una funzione di transizione tale che $\delta : (Q \times \Sigma \times \Gamma^*) \to (Q \times \Gamma^*)$, $q_0 \in Q$ è lo stato iniziale, e $F \subseteq Q$ è l'insieme di stati finali. Dimostrare che questo modello è equivalente al modello della Macchina di Turing.

Per dimostrare l'equivalenza dell'automa a coda con la macchina di Turing, dobbiamo mostrare che è possibile simulare una TM su un Automa a Coda, e viceversa.
L'inverso è facile da mostrare: un inserimento su un Automa a Coda può essere simulato tramite un inserimento all'inizio del nastro seguito da uno shift a destra di tutto il contenuto del nastro, e da una eliminazione del simbolo all'estrema destra del nastro.
Ora mostriamo che un automa a coda $\mathcal{A}$ può sempre simulare tutti i passi computazionali di macchina di Turing. Un Automa a Coda differisce dal noto Automa a Pila per il fatto che le operazioni di push vengono effettuate nell'estremo opposto del buffer, rispetto a quello dove si effettuano i pop. Quello che dà a questo automa potenzialità maggiori rispetto ad un PDA è il fatto di poter leggere tutto l'input attraverso una serie di push/pop che leggono ciascuna lettera dalla cima e la riscrivono sul fondo. Se la dimensione della pila inizialmente è k, dopo un ciclo di k passi di questo tipo lo stato della pila sarà rimasto invariato, ma sarà stato possibile all'automa leggere tutto il contenuto. Un problema che sorge è che il valore di k è ignoto, per cui l'automa non può sapere quanti passi deve compiere per leggere il contenuto della pila una sola volta. Per risolvere questo problema, come nel caso della k-TM introduciamo nell'alfabeto dell'automa, per ogni simbolo $\gamma \in \Gamma$, un simbolo 'marcato' $\underline{\gamma}$, che rappresenta la presenza della testina di lettura su quel simbolo.
Pertanto, se all'inizio della computazione i caratteri di input vengono inseriti nella coda, il primo verrà marcato tramite una sostituzione $\gamma/\underline{\gamma}$. Ora, un generico passo computazionale della macchina di Turing legge un simbolo γ, lo sovrascrive con un nuovo γ' e poi può spostarsi a destra oppure a sinistra. Simuliamo il primo dei due casi tramite la sequenza:
- Pop della versione marcata del simbolo in lettura $\overline{\gamma}$;
- Push del nuovo simbolo (non marcato) γ';
- Pop del simbolo che ora è in testa γ_2;
- Push dello stesso simbolo, però marcato $\overline{\gamma}_2$;
- Scorri fino ad incontrare il simbolo appena inserito. Per scorrimento si intende il pop seguito dal push di uno stesso simbolo. Questo è possibile dato che il simbolo che stiamo cercando è marcato, e ce n'è solo uno.

Il secondo caso è un po' più complesso: per simulare uno spostamento a sinistra, dobbiamo trovare un modo per cui l'ultimo elemento in coda diventa il primo, il primo diventa il secondo e così via, procediamo così:
- Pop (della versione marcata) del simbolo $\overline{\gamma}$;
- Push della versione marcata del nuovo simbolo $\overline{\gamma}'$;
- Prepara una serie di stati per ogni simbolo possibile non marcato (essendo i simboli in numero finito e noti a tempo di costruzione, questo è sempre possibile). Finchè il simbolo che leggi non è marcato (cioè il vecchio primo), usa gli stati appena costruiti per ricordarti cosa hai letto, poi scorri il simbolo;
- Appena trovi il simbolo marcato, fai il push della versione marcata dell'ultimo simbolo letto (che ricordiamo grazie agli stati che abbiamo costruito);

– Fai il pop del simbolo (marcato) in cima, e il push della sua versione non marcata;
– Scorri tutti i simboli finché non trovi quello marcato.

Se il simbolo che la transizione di $\mathcal{M}$ legge è $\sqcup$, anziché eliminare il carattere in cima, lo reinseriamo non marcato, e poi inseriamo la versione marcata del nuovo carattere, ammesso che vogliamo fare un inserimento, seguendo una delle due tecniche sopra descritte.

Esercizio 32

Definire formalmente il modello di computazione Automa con 2 Pile, e mostrare che si tratta di un modello equivalente alla Macchina di Turing.

Un Automa a due Pile è una tupla $(Q, \Sigma, \Gamma, \delta, q_0, F)$, dove Q è l'insieme degli stati; Σ è l'alfabeto dell'input; Γ è l'alfabeto delle pile (volendo, potremmo anche assumere due alfabeti diversi per ognuna delle due pile); δ è una funzione di transizione del tipo $\delta : (Q \times \Sigma \times \Gamma^2 \to Q \times \Gamma^2)$; q_0 è lo stato iniziale e $F \subseteq Q$ è l'insieme degli stati finali (accettanti). L'input è una sequenza di simboli ordinati su di un 'nastro di input' (astratto) in sola lettura, e abbiamo due pile di tipo LIFO per salvare i simboli di Γ. Questo automa accetta *se e solo se* sia le pile che il nastro sono vuoti, e ci troviamo in un qualche stato $q \in F$. Per mostrare che è equivalente alla macchina di Turing, proviamo a fare una costruzione simile a quella nell'esercizio precedente: dato un qualsiasi automa a due pile $\mathcal{A}$, mostrare che esiste una macchina di Turing $\mathcal{M}$ che lo simula; e data una qualsiasi macchina di Turing $\mathcal{M}$, costruiamo un automa a due pile $\mathcal{A}$ che la simula. La prima direzione è quella più semplice, specialmente se immaginiamo una macchina di Turing a 3 nastri: allora, ogni operazione può essere simulata in modo del tutto simile a quanto fatto nell'Esercizio **??**, e quindi abbiamo la prima direzione. Per la direzione inversa, partiamo dall'inizializzazione dell'automa. Come era per l'automa a coda, il nastro di lettura è del tutto inutile, quindi procediamo immediatamente a trasferirne l'input su una pila (che quindi sarà al contrario), per poi rimetterlo in ordine sulla seconda pila (intuitivamente, essere in grado di 'riciclare' l'input è un ottimo indizio della possibilità di un automa di essere Turing-completo). In questo caso non avremo bisogno di un alfabeto marcato, dato che la posizione della testina è rappresentata dal fatto di trovarsi in cima ad una delle due pile. Simuliamo ora una transizione di $\mathcal{M}$ che prevede un movimento a destra:
– Pop del simbolo in cima alla seconda pila;
– Push del nuovo simbolo sulla prima pila.

Se $\mathcal{M}$ arriva al simbolo $\sqcup$, significa che la seconda pila è vuota, perciò basta non eseguire il primo passo. Simuliamo ora un movimento a sinistra:
– Pop del simbolo in cima alla seconda pila;
– Push del nuovo simbolo sulla seconda pila;
– Sposta il simbolo in cima alla prima pila sulla seconda pila.

Anche in questo caso, se $\mathcal{M}$ arriva al simbolo $\sqcup$, significa che la prima pila

è vuota, e perciò ignoriamo l'ultimo passaggio. Si può vedere come questa costruzione è molto più semplice di quella per l'automa a coda: in effetti, il nastro di una macchina di Turing può essere visto una coppia di stack posti 'uno contro l'altro', in cui la cima del primo è l'elemento subito a sinistra della testina, e la cima del secondo è l'elemento sotto la testina.

Esercizio 33

Data la seguente tabella che riassume le proprietà di chiusura di ciascuna classe di linguaggi, completare la riga riferita a DEC, dimostrando ciascun caso:

	unione	composizione	star di Kleene	intersezione	complemento	differenza insiemistica
REG	✓	✓	✓	✓	✓	✓
CFG	✓	✓	✓	✗	✗	✗
DEC	✓				✓	

- **Intersezione:** ✓ Poiché i linguaggi sono insiemi, valgono le proprietà degli insiemi, in particolare: $L_1 \cap L_2 = (\overline{L_1} \cup \overline{L_2})$. Essendo DEC chiuso sia per unione che per complemento, la chiusura per intersezione segue;
- **Differenza insiemistica:** ✓ Come per l'intersezione, ragionando sulle proprietà degli insiemi: $L_1 \setminus L_2 = L_1 \cup \overline{L_2}$; essendo DEC chiuso per unione e complemento, la chiusura per differenza segue;
- **Composizione:** ✓ Dati due linguaggi L_1, L_2 e le loro TM associate $\mathcal{M}_1$, $\mathcal{M}_2$, costruiamo la 2-TM $\mathcal{M}$ che opera sul primo nastro come $\mathcal{M}_1$ finché non arriva ad uno stato di accettazione (altrimenti rifiuta), e invece di accettare passa allo stato iniziale di $\mathcal{M}_2$, che opera sul secondo nastro. Poiché le 2-TM sono equivalenti alle TM 'normali', la chiusura è dimostrata;
- **Star di Kleene:** ✓ Dato un linguaggio L e la sua TM associata $\mathcal{M}$, per riconoscere L^*, possiamo costruire una seconda TM che ripete l'esecuzione di $\mathcal{M}$ su più sotto-parole dell'input. Per farlo, usiamo una 2-TM che inizialmente, esegue passo non-deterministico per dividere l'input in n parti; dopodiché, per ciascuna parte, essa viene copiata nel secondo nastro, dove viene simulata $\mathcal{M}$. La 2-TM accetta solo e tutte le simulazioni accettano, e rifiuta altrimenti;

Esercizio 34

Considerare la gerarchia di Chomsky, e dare un esempio completo di grammatica per ogni tipo, evidenziando le differenze con la grammatica di tipo immediatamente inferiore attraverso opportuni esempi.

Esempi di grammatiche in base al tipo:

- **Tipo 3 (regolare)** Ammette regole di produzione del tipo $A ::= aB$ oppure $A ::= a$, dove a è un terminale, e A, B sono non terminali. Ad esempio, dato l'alfabeto $\{a, b, c\}$, la seguente grammatica regolare corrisponde al linguaggio a^*bc^+:
$$G = \begin{cases} S ::= aS \mid bT \\ T ::= cT \mid c \end{cases}$$
- **Tipo 2 (libera dal contesto)** Ammette regole di produzione del tipo $A ::= \alpha$, dove A è un non terminale e α una qualsiasi sequenza di terminali e non terminali (quindi estende il Tipo 3). Si può limitare la scelta di α ad una coppia di non terminali oppure ad un solo non terminale, ottenendo comunque una grammatica equivalente (forma normale di Chomsky). Ad esempio, dato l'alfabeto $\{0, 1\}$, la seguente grammatica libera dal contesto corrisponde al linguaggio 0^n1^n, $\forall n \in \mathbb{N}$:
$$G = \begin{cases} S ::= 0S1 \mid \epsilon \end{cases}$$
- **Tipo 1 (dipendente dal contesto)** Ammette regole di produzione del tipo $\beta A\gamma ::= \beta\alpha\gamma$, dove A è un non terminale; α, β, γ sono sequenze arbitrarie di terminali e non terminali, e sia β che γ possono essere omesse (se entrambe sono omesse, otteniamo una regola libera dal contesto). Ad esempio, dato l'alfabeto $\{0, 1, 2\}$, la seguente grammatica legata al contesto corrisponde al linguaggio $0^n1^n2^n$, $\forall n \in \mathbb{N}$:
$$G = \begin{cases} S ::= 0BC \mid 0SBC \\ CB ::= BC \\ 0B ::= 01 \\ 1B ::= 11 \\ 1C ::= 12 \\ 2C ::= 22 \end{cases}$$
- **Tipo 0 (non ristretta)** Ammette regole di produzione del tipo $\alpha ::= \beta$, dove sia α che β sono sequenze qualsiasi di simboli terminali o non terminali. È possibile limitare la scelta di α a soli simboli non terminali, e comunque ottenere una grammatica equivalente. Ad esempio, dato l'alfabeto $\{a\}$, la seguente grammatica ricorsivamente enumerabile corrisponde

al linguaggio a^{2^n}:

$$G = \begin{cases} S ::= ACaB \\ Ca ::= aaC \\ CB ::= DB \mid E \\ aD ::= Da \\ AD ::= AC \\ aE ::= Ea \\ AE ::= \epsilon \end{cases}$$

Esercizio 35

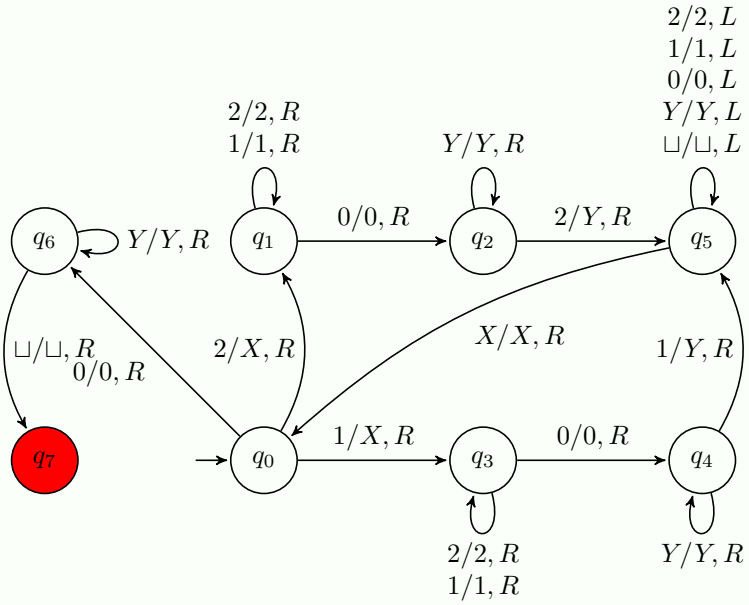
Si consideri la TM $\mathcal{M} = (\{q_i \mid i = 0, \ldots, 7\}, \Sigma = \{0, 1, 2\}, \Gamma = \Sigma \cup \{X, Y\}, \delta, q_0, q_7, q_r)$, dove le seguenti tuple costituiscono δ:

- ➔ $((q_0, 2), (q_1, X, R))$; $((q_0, 1), (q_3, X, R))$;
- ➔ $((q_1, 2), (q_1, 2, R))$; $((q_3, 2), (q_3, 2, R))$;
- ➔ $((q_1, 1), (q_1, 1, R))$; $((q_3, 1), (q_3, 1, R))$;
- ➔ $((q_1, 0), (q_2, 0, R))$; $((q_3, 0), (q_4, 0, R))$;
- ➔ $((q_2, Y), (q_2, Y, R))$; $((q_4, Y), (q_4, Y, R))$;
- ➔ $((q_2, 2), (q_5, Y, R))$; $((q_4, 1), (q_5, Y, R))$;
- ➔ $((q_5, 2), (q_5, 2, L))$; $((q_5, 1), (q_5, 1, L))$, $((q_5, 0), (q_5, 0, L))$; $((q_5, Y), (q_5, Y, L))$, $((q_5, \sqcup), (q_5, \sqcup, L))$;
- ➔ $((q_5, X), (q_0, X, R))$;
- ➔ $((q_0, 0), (q_6, 0, R))$;
- ➔ $((q_6, Y), (q_6, Y, R))$;
- ➔ $((q_6, \sqcup), (q_7, \sqcup, R))$.

Adesso:

- si disegni l'automa;

L'automa è simile al seguente; si immagini un aggiuntivo stato di rifiuto q_r al quale portano tutte le transizioni non esplicitate in figura:



- si dica se si tratta di una TM deterministica;

Sì.

Non esistono ϵ-transizioni, nè transizioni che partono dallo stesso stato e che leggono lo stesso carattere.

- si dica quali delle seguenti parole sono riconosciute:
  - 102

No.

  - 202

Sì.

  - 2012

No.

  - 12012

Sì.

- si dica in linguaggio naturale quali parole sono accettate.

La TM accetta parole del tipo $w0w,\ w \in \{1,2\}^*$.

CAPITOLO 4

Calcolabilità

Esercizio 36

Dimostrare che sono decidibili i seguenti linguaggi:

• EQ_{DFA}, il linguaggio di tutte le coppie di descrizioni di DFA che riconoscono lo stesso linguaggio;

Usiamo due risultati noti: che REG è chiuso per differenza insiemistica, e che $E_{\text{DFA}} = \{\langle \mathcal{A} \rangle \mid \mathcal{A} \text{ è un DFA tale che } L(\mathcal{A}) = \emptyset\} \in \text{DEC}$. Sia $\mathcal{M}$ una TM che decide E_{DFA}, diamo una procedura Turing-computabile per decidere EQ_{DFA}:
- Data una stringa w, controlla se è nel formato $\langle \mathcal{A}_1, \mathcal{A}_2 \rangle$, dove A_1, A_2 sono due DFA;
- Costruisci due nuovi DFA A_{21} e A_{12} che riconoscono rispettivamente le differenze $L(A_2) \setminus L(A_1)$ e $L(A_1) \setminus L(A_2)$;
- Simula $\mathcal{M}(\langle A_{21} \rangle)$ e $\mathcal{M}(\langle A_{12} \rangle)$, accettando solo se accettano entrambe le simulazioni, e rifiutando altrimenti.

Chiaramente, se $\mathcal{A}_1$ e $\mathcal{A}_2$ riconoscono lo stesso linguaggio, allora A_{21} e A_{12} riconoscono esattamente $\emptyset$, dunque entrambe le simulazioni accettano, e quindi anche la nostra TM. In caso contrario, almeno una delle due simulazioni rifiuterà, portando TM a rifiutare. Si noti che l'esistenza di A_{21} e A_{12} è assicurata dal fatto che REG è chiuso per differenza insiemistica.

• E_{CFG}, il linguaggio di tutte descrizioni di grammatiche libere il cui linguaggio è vuoto.

Data una grammatica $G = (V, \Sigma, \mathcal{R}, S)$, definiamo il suo *albero di produzione* come l'albero che ha S per radice, e tale che ciascun nodo $\gamma_1 \dots \gamma_n$, con $\gamma_i \in V \cup \Sigma$, ha un figlio per ogni applicazione a $\gamma_1 \dots \gamma_n$ di una regola di produzione, e nessun figlio se nessuna regola di produzione è applicabile (ovvero, se il nodo è una stringa di simboli terminali). Si noti che l'albero ha nelle foglie le stringhe accettate da G ma, se esistono cicli tra le produzioni di G, l'albero risulta infinito, per cui una visita dell'albero (in ampiezza o profondità) non basta per decidere se $L(G) = \emptyset$. Tuttavia, per la natura dei linguaggi liberi dal contesto, per decidere se esiste almeno una foglia, è possibile visitare l'albero tralasciando i percorsi root-to-leaves che usano una stessa regola di produzione più di una volta. Pertanto, la seguente TM, con input w, decide E_{CFG}:

– Controlla se w è nel formato corretto di una grammatica;
– Esegui una visita dell'albero di produzione della grammatica, ignorando ad ogni nodo le regole già utilizzate lungo il percorso corrente, accettando se si raggiunge una foglia con soli simboli terminali, e rifiutando altrimenti.

Esercizio 37 33:54 c.a. lezione 12

Data la seguente tabella che riassume le proprietà di chiusura di ciascuna classe di linguaggi, completare la riga riferita a R.E., dimostrando ciascun caso:

	unione	composizione	star di Kleene	intersezione	complemento	differenza insiemistica
REG	✓	✓	✓	✓	✓	✓
CFG	✓	✓	✓	✗	✗	✗
DEC	✓	✓	✓	✓	✓	✓
R.E.	●				●	●

• **Unione:** ✓ Dati due linguaggi ricorsivamente enumerabili L_1, L_2, esistono due macchine di Turing $\mathcal{M}_1$, $\mathcal{M}_2$ che li semi-decidono. Attraverso una 2-TM $\mathcal{M}$ che simula $\mathcal{M}_1$ su un nastro e $\mathcal{M}_2$ sull'altro, e che accetta se una delle due simulazioni accetta, è evidente che $\mathcal{M}$ è un enumeratore per $L = L_1 \cup L_2$. Si noti che non è un decisore, dato che accetta tutte le volte che la parola appartiene ad uno dei linguaggi, ma non dà garanzie di terminazione nel caso la parola non appartenga a nessuno dei due;
• **Intersezione:** ✓ Come nel caso precedente, una 2-TM può simulare le due TM, accettando solo se entrambe le simulazioni accettano, ottenendo così un enumeratore per $L = L_1 \cap L_2$;
• **Composizione:** ✓ In questo caso è sufficiente una TM a nastro singolo non deterministica, che sceglie non-deterministicamente una divisione della parola in due parti, simula $\mathcal{M}_1$ sulla prima parte della parola, $\mathcal{M}_2$ sulla seconda, accettando se entrambe le simulazioni hanno accettato;
• **Star di Kleene:** ✓ Similmente al caso precedente, è sufficiente una TM a nastro singolo non deterministica, che sceglie non-deterministicamente una divisione della parola in n parti, e simula $\mathcal{M}$ su ciascuna parte, accettando se tutte le simulazioni hanno accettato;
• **Complemento:** ✗ Dato un linguaggio $L \in$ R.E., se anche $\overline{L}$ fosse R.E., allora $L \in$ DEC; per cui se R.E. fosse chiuso per complemento, si darebbe il caso che R.E. = DEC, che sappiamo essere falso;

- **Differenza insiemistica: ✗** Se R.E. fosse chiuso differenza insiemistica, allora sarebbe chiuso anche per complemento, poiché $\overline{L} = \Sigma^* \setminus L$ e $\Sigma^* \in$ R.E..

Esercizio 38

Dimostrare che se L_1, L_2 sono due linguaggi tali che $L_1 \leq_m L_2$ e che $L_2 \in$ R.E., allora $L_1 \in$ R.E.

Se $L_2 \in$ R.E. allora esiste una TM $\mathcal{M}_2$ che lo semi-decide. Inoltre, siccome $L_1 \leq_m L_2$ e la funzione che testimonia la riducibilità è ricorsivamente enumerabile, esiste una macchina di Turing $\mathcal{M}_f$ che la computa. Allora, possiamo dare una macchina che semi-decide L_1 che, con input w, calcola $f(w)$ usando $\mathcal{M}_f$, ed esegue $\mathcal{M}_2(f(w))$, accettando se accetta, e rifiutando altrimenti. Al contrario del caso decidibile visto a lezione, la simulazione potrebbe non terminare; per questo L_1 potrebbe non essere in DEC. Tuttavia, per parole $w \in L_1$, $f(w) \in L_2$, dunque la computazione terminerà, e quindi $L_1 \in$ R.E.

Esercizio 39 48:00 12

Si verifichi se valgono o no le seguenti proprietà per $\leq_m$:

- Riflessività;

Sì.

Ogni problema si può ridurre a se stesso usando come riduzione la funzione identità.

- Simmetria;

No.

In classe abbiamo visto che $\overline{A}_{\text{TM}} \leq_m EQ_{\text{TM}}$. Se $\leq_m$ fosse simmetrica, allora $EQ_{\text{TM}} \leq_m \overline{A}_{\text{TM}}$ e, essendo $\overline{A}_{\text{TM}} \in$ CoR.E, allora anche $EQ_{\text{TM}} \in$ CoR.E. Tuttavia, questo è in disaccordo con il risultato $EQ_{\text{TM}} \notin$ CoR.E, dimostrato a lezione.

- Transitività;

Sì.

Se $X \leq_m Y \leq_m Z$, e f, g sono le funzioni di riduzione $f : X \to Y,\ g : Y \to Z$, allora $f \circ g : X \to Z$ è una riduzione che dimostra $X \leq_m Z$.

- Antisimmetria.

No.

Se $\leq_m$ fosse antisimmetrica, allora dati due problemi che si possono $\leq_m$-ridurre reciprocamente, essi sarebbero lo stesso problema. Tuttavia, i due problemi A_{TM} e H_{TM} si possono ridurre reciprocamente, pur trattandosi di due problemi diversi. Nelle trasparenze abbiamo visto la riduzione $A_{\text{TM}} \leq_m H_{\text{TM}}$, diamo ora una riduzione che dimostri $H_{\text{TM}} \leq_m A_{\text{TM}}$. Data una descrizione del tipo $\langle \mathcal{M}, w \rangle$, la riduzione deve computare una descrizione $\langle \mathcal{M}', w' \rangle$ tale che $\mathcal{M}$ termina su w se e solo se $\mathcal{M}'$ accetta w'. A questo scopo, calcoliamo $w' = w$ e costruiamo la macchina $\mathcal{M}'$ come segue:
– Esegue $\mathcal{M}(w)$;
– Accetta.
Questa riduzione è computabile e rispetta le condizioni: se $\mathcal{M}(w)$ termina, allora $\mathcal{M}'$ accetta tutte le stringhe, e quindi anche w'; altrimenti, $\mathcal{M}'$ non terminerà mai e dunque non accetterà nessuna stringa, inclusa w'.

Esercizio 40 1:11 13

Dimostrare i seguenti fatti riguardo all'insieme E_{TM}:

- $E_{\text{TM}} \notin$ DEC;

Supponiamo, per assurdo, che E_{TM} sia decidibile, cioè esiste una macchina di Turing $\mathcal{M}'$ che, data la descrizione di una TM $\langle \mathcal{A}' \rangle$ termina sempre e decide $\mathcal{A}'$ ha linguaggio vuoto. Allora, potremmo costruire una nuova macchina $\mathcal{M}$ che decide A_{TM}. Con input x, $\mathcal{M}$:
– Verifica che x sia nel formato $\langle \mathcal{A}, w \rangle$;
– Simula $\mathcal{A}(w)$, accettando se la simulazione accetta, e rifiutando altrimenti;
– Codifica $\mathcal{A}'$ nel seguente modo:
  – Se la stringa di input è diversa da w, rifiuta;
  – Altrimenti, simula $\mathcal{A}(w)$.
– Esegui $\mathcal{M}'$ su input $\mathcal{A}'$, accettando se $\mathcal{M}'$ rifiuta, e rifiutando se $\mathcal{M}'$ accetta.
Poiché, per costruzione, l'unica stringa che $\mathcal{A}'$ potrebbe accettare è w, allora chiaramente, se $\mathcal{M}'$ ci dice che $\mathcal{A}'$ non accetta niente, significa proprio che non accetta w, cioè abbiamo deciso A_{TM}. Tuttavia, non essendo A_{TM} decidibile, otteniamo che $\mathcal{M}'$ non può esistere, perciò anche E_{TM} non è decidibile.

- $A_{\text{TM}} \leq_m E_{\text{TM}}$ è falso;

Se $A_{\text{TM}} \leq_m E_{\text{TM}}$, allora la stessa riduzione dimostra $\overline{A}_{\text{TM}} \leq_m \overline{E}_{\text{TM}}$ (poiché le riduzioni $\leq_m$ richiedono una doppia implicazione). Tuttavia, abbiamo visto a lezione che $\overline{E}_{\text{TM}} \in$ R.E, perciò anche $\overline{A}_{\text{TM}} \in$ R.E. Ma dato che è

anche il caso che $\overline{A}_{\text{TM}} \in \text{CoR.E}$, allora $\overline{A}_{\text{TM}} \in \text{DEC}$ e $A_{\text{TM}} \in \text{DEC}$, che è falso.

- $A_{\text{TM}} \leq_T E_{\text{TM}}$.

La riduzione $A_{\text{TM}} \leq_T E_{\text{TM}}$ è stata data nel primo punto.

Esercizio 41

Dimostrare che i seguenti linguaggi non sono decidibili:

- il linguaggio di tutte le descrizioni di macchine di Turing il cui linguaggio è libero dal contesto;

Sia L il linguaggio di tutte le descrizioni di macchine di Turing il cui linguaggio è libero dal contesto, diamo una riduzione $A_{\text{TM}} \leq_m L$. La riduzione f mappa ogni descrizione $\langle \mathcal{M}, w \rangle$ in una stringa $\langle \mathcal{M}' \rangle$ tale che $\mathcal{M}$ accetta w se e solo se $L(\mathcal{M}')$ è libero dal contesto. Decidiamo di costruire $\mathcal{M}'$ in modo tale che accetti Σ^* (che è libero dal contesto) se $\mathcal{M}$ accetta w, e che in caso contrario riconosca $0^n1^n2^n$, che non è un linguaggio libero dal contesto. $\mathcal{M}'$ con input x: controlla se l'input è di tipo $0^n1^n2^n$, accettando in caso positivo; in caso contrario, simula $\mathcal{M}(w)$, accettando se la simulazione accetta, e rifiutando altrimenti.

- il linguaggio di tutte le descrizioni di macchine di Turing il cui linguaggio è infinito;

Sia L il linguaggio di tutte le descrizioni di macchine di Turing il cui linguaggio è finito, diamo una riduzione $A_{\text{TM}} \leq_m L$. La riduzione f mappa ogni descrizione $\langle \mathcal{M}, w \rangle$ in una stringa $\langle \mathcal{M}' \rangle$ tale che $\mathcal{M}$ accetta w se e solo se $L(\mathcal{M}')$ è finito. Decidiamo di costruire $\mathcal{M}'$ in modo tale che accetti solo il linguaggio Σ^* se $\mathcal{M}$ accetta w, e che in caso contrario riconosca solo la stringa 0 (che costituisce un linguaggio finito). $\mathcal{M}'$ con input x: controlla se l'input è 0, accettando in caso positivo; in caso contrario, simula $\mathcal{M}(w)$, accettando se la simulazione accetta, e rifiutando altrimenti.

- il linguaggio di tutte le descrizioni di macchine di Turing il cui linguaggio è finito;

Si noti che questo linguaggio non è esattamente il complemento di quello precedente, per cui per dimostrare il risultato direttamente non possiamo usare la chiusura di $\overline{\text{DEC}}$ per complemento. Inoltre, il linguaggio è il complemento di quello precedente rispetto al linguaggio delle descrizioni di macchine di Turing, che è chiaramente un linguaggio decidibile (si tratta di un qualche

tipo di linguaggio a grammatica), e dunque per il nostro scopo non è nemmeno utilizzabile la chiusura di $\overline{\text{DEC}}$ per differenza. Invece, la dimostrazione si dà per riduzione di Turing dal linguaggio precedente. Se, infatti, il linguaggio FIN in questione fosse decidibile, allora, sia $\mathcal{M}_{FIN}$ una macchina di Turing che lo decide, si potrebbe decidere il linguaggio INF tramite una macchina di Turing che, con input x:
- Verifica che x sia una descrizione di macchina di Turing, rifiutando in caso contrario;
- Esegue $\mathcal{M}_{FIN}(x)$, rifiutando se la simulazione accetta e accettando in caso contrario.

• il linguaggio di tutte le descrizioni di macchine di Turing il cui linguaggio contiene le stringhe 010^*;

Sia L il linguaggio di tutte le descrizioni di macchine di Turing il cui linguaggio contiene le stringhe 010^*, diamo una riduzione $A_{\text{TM}} \leq_m L$. La riduzione f mappa ogni descrizione $\langle \mathcal{M}, w \rangle$ in una stringa $\langle \mathcal{M}' \rangle$ tale che $\mathcal{M}$ accetta w se e solo se $L(\mathcal{M}')$ è contiene le stringhe 010^*. Decidiamo di costruire $\mathcal{M}'$ in modo tale che accetti Σ^* (che contiene stringhe di tipo 010^*) se $\mathcal{M}$ accetta w, e che in caso contrario riconosca solo la stringa 0 (che non è di tipo 010^*). $\mathcal{M}'$ con input x: controlla se l'input è 0, accettando in caso positivo; in caso contrario, simula $\mathcal{M}(w)$, rifiutando se la simulazione rifiuta; se invece la simulazione accetta, decidi se w è di tipo 010^*.

• il linguaggio di tutte le descrizioni di macchine di Turing tali che, se accettano w, allora accettano anche w^{-1}.

Sia L il linguaggio di tutte le descrizioni di macchine di Turing tali che se accettano w allora accettano anche w^{-1}, diamo una riduzione $A_{\text{TM}} \leq_m L$. La riduzione f mappa ogni descrizione $\langle \mathcal{M}, w \rangle$ in una stringa $\langle \mathcal{M}' \rangle$ tale che $\mathcal{M}$ accetta w se e solo se $L(\mathcal{M}') = L^{-1}(\mathcal{M}') = \{w \mid w^{-1} \in L(\mathcal{M}')\}$. Decidiamo di costruire $\mathcal{M}'$ in modo tale che accetti solo la stringa 10 se $\mathcal{M}$ accetta w, e che in caso contrario riconosca solo la stringa 01. $\mathcal{M}'$ con input x: controlla se l'input è 01, accettando in caso positivo; in caso contrario, simula $\mathcal{M}(w)$, rifiutando se la simulazione rifiuta; se invece la simulazione accetta, decidi se w è 10. Charamente, se $w \in L(\mathcal{M})$, allora $\mathcal{M}'$ accetta 10 e 01, e ha la caratteristica richiesta; altrimenti, accetta solo 01, e non ha la caratteristica richiesta.

Esercizio 42 24:46 13

Si caratterizzi la decidibilità dei seguenti linguaggi:

• $\{\langle \mathcal{M}, w \rangle \mid \mathcal{M}$, al computare w, scrive il simbolo a sul nastro$\}$;

Indecidibile.

Sia L il linguaggio in questione, diamo una riduzione $A_{\text{TM}} \leq_m L$. La riduzione f mappa ogni descrizione $\langle \mathcal{M}, w \rangle$ in una stringa $\langle \mathcal{M}', w' \rangle$ tale che $\mathcal{M}$ accetta w se e solo se $\mathcal{M}'$ al computare w' scrive a sul nastro. Decidiamo di costruire $\mathcal{M}'$ in modo tale che simuli $\mathcal{M}(w)$, e nel caso in cui la simulazione accetti, scriva una a sul nastro (e poi termini accettando o rifiutando). Per assicurarci che nel caso la simulazione non accetti, $\mathcal{M}'$ non abbia mai scritto a sul nastro, dobbiamo impedire alla simulazione di scrivere a sul nastro in qualunque momento. Per farlo, aggiungiamo una lettera $\overline{a}$ al suo alfabeto, e appena prima della simulazione, sostituiamo ogni a in w con $\overline{a}$, e modifichiamo le transizioni in $\mathcal{M}$ in modo tale che anzichè leggere o scrivere a, leggano o scrivano $\overline{a}$.

- $\{\langle \mathcal{M}, \mathcal{M}', S \rangle \mid S \subseteq \Sigma^*, \mathcal{M}$ e $\mathcal{M}'$ si comportano nello stesso modo su tutte le stringhe di $S\}$;

Indecidibile.

Sia $L_=$ il linguaggio in questione, diamo una riduzione $EQ_{\text{TM}} \leq_m L_=$. La riduzione f mappa ogni descrizione $\langle \mathcal{M}_1, \mathcal{M}_1' \rangle$ in una stringa $\langle \mathcal{M}_2, \mathcal{M}_2', S \rangle$ tale che $L_=(\mathcal{M}_1) = L_=(\mathcal{M}_1')$ se e solo se per ogni $(\mathcal{M}_2, \mathcal{M}_2') \in A(S)$. Banalmente, $M_2 = M_1, M_2' = M_1, S = \Sigma^*$ soddisfa questa condizione, poiché in genere $L_=(\mathcal{M}) = L_=(\mathcal{M}') \iff (\mathcal{M}, \mathcal{M}') \in A(\Sigma^*)$.

- $\{\langle \mathcal{M}, \mathcal{M}', S \rangle \mid S \subseteq \Sigma^*, \mathcal{M}$ e $\mathcal{M}'$ si comportano in modo diverso su tutte le stringhe di $S\}$;

Indecidibile.

Sia $L_{\neq}$ il linguaggio in questione; esso si può ridurre al linguaggio $L_=$ che, come visto nel punto precedente, è indecidibile. La riduzione f mappa ogni descrizione $\langle \mathcal{M}_1, \mathcal{M}_1', S_1 \rangle$ in una stringa $\langle \mathcal{M}_2, \mathcal{M}_2', S_2 \rangle$ tale che $(\mathcal{M}_1, \mathcal{M}_1') \in A(S)$ se e solo se $(\mathcal{M}_2, \mathcal{M}_2') \in D(S)$. Questo si può ottenere fissando lo stesso inseme ($S_2 = S_1$), la stessa prima TM ($\mathcal{M}_2 = \mathcal{M}_1$), e usando la *macchina complemento* della seconda TM come nuova seconda TM: $\mathcal{M}_2' = \text{complement}(\mathcal{M}_1')$. Una macchina complemento si ottiene semplicemente scambiando gli stati accettante e di rifiuto, e la riduzione rispetta la condizione perchè, siccome due macchine complemento danno risposte sempre diverse, si ha la relazione generale $(\mathcal{M}, \mathcal{M}') \in A(S) \iff (\mathcal{M}, \text{complement}(\mathcal{M}')) \in D(S)$. Essendo $EQ_{\text{TM}} \notin$ R.E., $EQ_{\text{TM}} \notin$ CoR.E, e così pure il suo complemento $\overline{EQ}_{\text{TM}}$, deriviamo che pure $L_=, L_{\neq} \notin$ R.E e $L_=, L_{\neq} \notin$ CoR.E.

- $\{\langle \mathcal{M}, q \rangle \mid \mathcal{M}$ ha uno stato q che non viene mai usato al computare di qualunque input $w \in \Sigma^*\}$.

Indecidibile.

Sia L il linguaggio in questione, diamo una riduzione $E_{\mathrm{TM}} \leq_m L$, che sfrutta il fatto che qualsiasi stato accettante non è mai visitato se $L = \emptyset$. Così, supponiamo di avere una macchina $\mathcal{M}$ che decide se $\langle \mathcal{M}, q \rangle$ è una descrizione di TM e estato dove q è uno stato inutile di M; mostriamo che possiamo costruire una macchina $\mathcal{M}'$ che riconosce E_{TM}. Con input x, $\mathcal{M}'$:

- Controlla che x sia una descrizione di TM valida;
- Trova lo stato accettante di q (se ce n'è più di uno compie un passo per renderlo unico);
- Simula $\mathcal{M}(x, q)$, accettando o rifiutando appropriatamente.

Se la descrizione di macchina in input x è quella di una macchina con linguaggio vuoto, allora q è uno stato inutile e la simulazione accetta; altrimenti, se il linguaggio non è vuoto, la simulazione rifiuterà.

CAPITOLO 5

Linguaggi Logici

Esercizio 43

Classificare le seguenti formule tra tautologie, contraddizioni, e contingenze; tra queste ultime, dire quali sono soddisfatte dall'interpretazione $I_1 = \{p = 1, q = 0, r = 0\}$, e quali dall'interpretazione $I_2 = \{p = 1, q = 1, r = 0\}$:

- $(p \vee \neg p)$;

Tautologia.

$(p \vee \neg p) \equiv \top$

- $(p \wedge q) \rightarrow r$;

Contingenza.

Soddisfatta da I_1.

- $((p \wedge q) \rightarrow \neg(r \rightarrow q)) \vee \neg q$;

Contingenza.

Usando le sostituzioni: $(p \rightarrow q) \rightarrow (\neg p \vee q)$, le regole di De Morgan, e rimuovendo letterali ridondanti, si ottiene la formula equivalente: $(\neg p \vee \neg q \vee (r \wedge \neg q))$, che è soddisfatta da I_1.

- $(((\neg p) \rightarrow q)) \vee (p \rightarrow r)$.

Tautologia.

Operando alcune trasformazioni, si ottiene la formula equivalente $(p \vee q \vee \neg p \vee r)$, che è sempre vera poiché contiene solo disgiunzioni, e $(p \vee \neg p)$ è una tautologia.

- $((p \wedge (p \rightarrow q)) \rightarrow q)$;

Tautologia.

Operando alcune trasformazioni, si ottiene la formula equivalente $(\neg p \vee \neg q \vee q)$ che, per gli stessi motivi del punto precedente, è una tautologia.

- $((p \wedge (p \to q)) \wedge \neg q)$;

Contraddizione.

Applicando passaggi simili a quelli del punto precedente, si ottiene la formula equivalente $(p \wedge q \wedge \neg q)$, a sua volta equivalente a $(\bot)$.

- $(p \vee q) \wedge (\neg q \wedge \neg p)$;

Contraddizione.

Applicando De Morgan, si trova che $(p \vee q)$ e $(\neg q \wedge \neg p)$ sono precisamente l'una la negata dell'altra.

- $(p \vee q) \wedge (p \to r \wedge q) \wedge (q \to \neg r \wedge p)$.

Contraddizione.

Se p oppure q sono vere (prima clausola), allora almeno uno dei conseguenti delle due clausole $(p \to r \wedge q)$ e $(q \to \neg r \wedge p)$ devono essere veri. Di queste due clausole, ognuno dei conseguenti implica però anche l'antecedente dell'altra, quindi entrambi i conseguenti devono essere veri. Ma essendo i conseguenti in contraddizione sul valore di verità di r, non può allora esistere nessuna interpretazione che renda la formula vera.

Esercizio 44

Dati gli operatori booleani $\neg$, $\bot$, $\top$, $\vee$, $\wedge$ e $\to$, un sottoinsieme di operatori si dice *completo* se, usando solo operatori del sottoinsieme, è possibile esprimere tutti gli operatori rimanenti. Dimostrare che sono completi gli insiemi di operatori:

- $\{\bot, \to\}$;

$$\begin{aligned} \neg p &\equiv (p \to \bot) \\ \top &\equiv \bot \to \bot \\ (p \vee q) &\equiv (\neg\neg p \vee q) \equiv ((p \to \bot) \to q) \\ (p \wedge q) &\equiv (\neg(\neg p \vee \neg q)) \equiv ((p \to \bot) \to (q \to \bot) \to \bot) \end{aligned}$$

- $\{\vee, \neg\}$;

$$\begin{aligned} \top &\equiv p \vee \neg p \\ \bot &\equiv \neg(p \vee \neg p) \\ (p \wedge q) &\equiv \neg(\neg p \vee \neg q) \\ (p \to q) &\equiv (\neg p \vee q) \end{aligned}$$

- $\{\uparrow\}$ (cioè *nand*);

$$\begin{aligned} \neg p &\equiv (p \uparrow p) \\ \top &\equiv p \vee \neg p \\ \bot &\equiv \neg(p \vee \neg p) \\ (p \wedge q) &\equiv \neg(p \uparrow q) \equiv ((p \uparrow q) \uparrow (p \uparrow q)) \\ (p \vee q) &\equiv (\neg p \uparrow \neg q) \equiv ((p \uparrow p) \uparrow (q \uparrow q)) \\ (p \to q) &\equiv (\neg p \vee q) \equiv (p \uparrow (q \uparrow q)) \end{aligned}$$

- $\{\downarrow\}$ (cioè *nor*).

$$\begin{aligned} \neg p &\equiv (p \downarrow p) \\ \top &\equiv p \vee \neg p \\ \bot &\equiv \neg(p \vee \neg p) \\ (p \vee q) &\equiv \neg(p \downarrow q) \equiv ((p \downarrow q) \downarrow (p \downarrow q)) \\ (p \wedge q) &\equiv (\neg p \downarrow \neg q) \equiv ((p \downarrow p) \downarrow (q \downarrow q)) \\ (p \to q) &\equiv (\neg p \vee q) \equiv ((p \downarrow p) \downarrow q) \downarrow ((p \downarrow p) \downarrow q) \end{aligned}$$

Esercizio 45

Dimostrare i seguenti fatti usando l'algoritmo del tableau:

- $(\neg p \vee q) \wedge (\neg q \vee r) \wedge (p \vee \neg r) \wedge (p \vee q \vee r) \to (p \wedge q \wedge r)$;

1. L'intera proposizione deve essere vera;
$$(\neg p \vee q \wedge \neg q \vee r \wedge p \vee \neg r \wedge p \vee q \vee r).V$$
2. I congiunti devono essere veri;
$$(\neg p \vee q).V \qquad (\neg q \vee r).V \qquad (p \vee \neg r).V \qquad (p \vee q \vee r).V$$
3. Per convenienza, si riscrivono le prime tre disgiunzioni come implicazioni;
$$(p \to q).V \qquad (q \to r).V \qquad (r \to p).V \qquad (p \vee q \vee r).V$$
4. Si espandono la prima delle disgiunzioni, derivando una contraddizione in uno dei due rami.
$$(p).F \qquad | \qquad (q).V$$

- $(\neg q) \wedge (p \to q) \wedge (\neg p \to \neg r) \to \neg r$;

1. L'intera proposizione deve essere vera;
$$(\neg q \wedge (p \to q) \wedge (\neg p \to \neg r)).V$$
2. I congiunti devono essere veri;
$$(\neg q).V \qquad ((p \to q) \wedge \neg p \to \neg r).V$$
$$(\neg q).V \qquad (p \to q).V \qquad (\neg p \to \neg r).V$$
3. La negazione deve essere falsa e, se è vera l'implicazione, o è falsa la premessa, o è vera la conseguenza;
$$(q).F \qquad (p).F \vee (q).V \qquad (\neg p).F \vee (\neg r).V$$
4. La conseguenza della prima implicazione è necessariamente falsa;
$$(\neg r).V$$
5. La premessa della seconda implicazione è necessariamente falsa;
$$(p).F \qquad (\neg r).V$$
6. La negazione deve essere vera.
$$(r).F$$

Si trova, dunque, che l'unico modo di soddisfare la formula è: $\{p = \bot,\ q = \bot,\ r = \bot\}$, derivando nel mentre che $\neg r = \top$.

- $(\neg p \vee \neg q \vee r) \wedge (\neg r \vee s) \wedge (\neg r \vee t) \wedge (\neg s \vee \neg t \vee u) \wedge p \wedge q \to u$;

1. L'intera proposizione deve essere vera;
$$(\neg p \vee \neg q \vee r \wedge \neg r \vee s \wedge \neg r \vee t \wedge \neg s \vee \neg t \vee u \wedge p \wedge q).V$$
2. I congiunti devono essere veri;
$$(\neg p \vee \neg q \vee r).V \quad (\neg r \vee s).V \quad (\neg r \vee t).V \quad (\neg s \vee \neg t \vee u).V \quad (p).V \quad (q).V$$
3. Considerando la prima disgiunzione:
$$(\neg p).V \quad | \quad (\neg q \vee r).V \qquad (p).F \quad | \quad (\neg q \vee r).V$$
4. Il ramo sinistro genera una contraddizione proposizionale, dunque si procede col ramo destro:
$$(\neg q \vee r).V$$
$$(\neg q).V \quad | \quad (r).V \qquad (q).F \quad | \quad (r).V$$
5. Similmente, il ramo sinistro genera una contraddizione proposizionale, dunque si procede col ramo destro:
$$(r).V$$
6. Considerando la seconda disgiunzione:
$$(\neg r).V \quad | \quad (s).V$$
$$(r).F \quad | \quad (s).V$$

7. Ancora una volta, il ramo sinistro genera una contraddizione proposizionale, dunque si procede col ramo destro. Allo stesso modo, si considera la terza disgiunzione:

$$\begin{array}{ccc} (\neg r).V & | & (t).V \\ (r).F & | & (t).V \end{array}$$

8. Ancora una volta, il ramo sinistro genera una contraddizione proposizionale, dunque si procede col ramo destro. Si considera la quarta disgiunzione, che si risolve in maniera simile alla prima, derivando u sull'unico ramo che non genera contraddizioni:

$$\begin{array}{ccccc} (\neg s).V & | & (\neg t).V & | & (u).V \\ (s).F & | & (t).F & | & (u).V \end{array}$$

Esercizio 46

Si trasformino le seguenti formule in forma clausale:

- $p \vee \neg(p \wedge \neg r)$;

$$\begin{aligned} p \vee \neg(p \wedge \neg r) &\equiv p \vee (\neg p \vee r) \\ &\equiv p \vee \neg p \vee r \end{aligned}$$

- $\neg(((q \to p) \to \neg r) \wedge ((q \to p) \to q))$;

$$\begin{aligned} \neg(((q \to p) \to \neg r) \wedge ((q \to p) \to q)) &\equiv \neg((q \to p) \to (\neg r \wedge q)) \\ &\equiv \neg(\neg(q \to p) \vee (\neg r \wedge q)) \\ &\equiv \neg(\neg(\neg q \vee p) \vee (\neg r \wedge q)) \\ &\equiv \neg((q \wedge \neg p) \vee (\neg r \wedge q)) \\ &\equiv \neg(q \wedge \neg p) \wedge \neg(\neg r \wedge q) \\ &\equiv (\neg q \vee p) \wedge (r \vee \neg q) \end{aligned}$$

- $(p \wedge (p \to q)) \wedge \neg q$;

$$\begin{aligned} (p \wedge (p \to q)) \wedge \neg q &\equiv (p \wedge (\neg p \vee q)) \wedge \neg q \\ &\equiv p \wedge (\neg p \vee q) \wedge \neg q \end{aligned}$$

- $(p \vee \neg(p \wedge q)) \vee \neg q$.

$$\begin{aligned} (p \wedge (p \to q)) \vee \neg q &\equiv (p \wedge (\neg p \vee q)) \vee \neg q \\ &\equiv ((p \vee \neg q) \wedge ((\neg p \vee q) \vee \neg q)) \\ &\equiv (p \vee \neg q) \wedge (\neg p \vee q \vee \neg q) \end{aligned}$$

Esercizio 47

Si dica se le seguenti formule si possono scrivere come formule di Horn, e, in caso affermativo, si scriva una forma di Horn equivalente:

- $p \to (\neg q \vee r)$;

$$\begin{aligned} p \to (\neg q \vee r) &\equiv \neg p \vee (\neg q \vee r) \\ &\equiv \neg p \vee \neg q \vee r \\ &\equiv (p \wedge q) \to r \end{aligned}$$

- $(\neg p \vee q) \to r$;

No.

- $(p \wedge q) \to r$;

$$\begin{aligned} (p \wedge q) \to r &\equiv \neg(p \wedge q) \vee r \\ &\equiv (\neg p \vee \neg q) \vee r \\ &\equiv \neg p \vee \neg q \vee r \\ &\equiv (p \wedge q) \to r \end{aligned}$$

- $(p \wedge (p \to q)) \to q$;

$$\begin{aligned} (p \wedge (p \to q)) \to q &\equiv \neg(p \wedge (\neg p \vee q)) \vee q \\ &\equiv (\neg p \vee \neg(\neg p \vee q)) \vee q \\ &\equiv (\neg p \vee (p \wedge \neg q)) \vee q \\ &\equiv ((\neg p \vee p) \wedge (\neg p \vee \neg q)) \vee q \\ &\equiv (\neg p \vee \neg q) \vee q \\ &\equiv \neg p \vee \neg q \vee q \\ &\equiv (p \wedge q) \to q \end{aligned}$$

- $(p \wedge (p \to q)) \wedge \neg q$.

$$\begin{aligned} (p \wedge (p \to q)) \wedge \neg q &\equiv (p \wedge (\neg p \vee q)) \wedge \neg q \\ &\equiv p \wedge (\neg p \vee q) \wedge \neg q \\ &\equiv (p \wedge \neg p \wedge \neg q) \vee (p \wedge q \wedge \neg q) \\ &\equiv \bot \end{aligned}$$

Esercizio 48

Dimostrare i seguenti fatti usando l'algoritmo del forward checking:

- da $\neg q$, $p \to q$, $\neg p \to \neg r$ si deduce $\neg r$;

Riscrivendo in forma implicativa, la base di conoscenza diventa:

$$\begin{aligned} q &\to \bot \\ p &\to q \\ r &\to p \end{aligned}$$

Per dimostrare $\neg r$, si ipotizza r per assurdo, derivando poi $\bot$. Dunque inizialmente $A = \{r\}$. Usando $r \to p$, si ottiene $A = \{r, p\}$, ed usando $p \to q$ si ottiene $A = \{r, p, q\}$. Infine, usando $q \to \bot$ si ottiene $A = \{r, p, q, \bot\}$. Essendo che $\bot \in A$, si ha il risultato.

- da $\neg p \vee \neg q \vee r$, $\neg r \vee s$, $\neg r \vee t$, $\neg s \vee \neg t \vee u$, p, q si deduce u;

Riscrivendo in forma implicativa, la base di conoscenza diventa:

$$\begin{aligned} (p \wedge q) &\to r \\ r &\to s \\ r &\to t \\ (s \wedge t) &\to u \\ p & \\ q & \end{aligned}$$

Inizialmente, $A = \{p, q\}$. Considerando $(p \wedge q) \to r$, si ha $A = \{p, q, r\}$. Considerando $r \to s$, si ottiene $A = \{p, q, r, s\}$, e considerando $r \to t$, si ottiene $A = \{p, q, r, s, t\}$. Infine, considerando $(s \wedge t) \to u$ si ottiene $A = \{p, q, r, s, t, u\}$. Terminato l'algoritmo, $\bot \notin A$ e $u \in A$, quindi si ha il risultato.

Esercizio 49

Si dica se le seguenti formule proposizionali quantificate sono vere oppure no:

- $\forall p \exists q (p \to q)$;

Vero.

---

p implica almeno se stesso, e dunque, per qualsiasi valore di verità di p, una interpretazione I per cui $I \models q \iff I \models p$ rende vero $p \to q$.

- $\forall p \forall q (p \to q)$;

Falso.

Per qualsiasi p esiste un valore di verità da dare a q tale che $p \to q$ è falso: $\neg p$. Pertanto, qualunque interpretazione I per cui $I \models q \iff I \models \neg p$ confuta la verità della formula.

- $\exists p \forall q (p \to q)$;

Vero.

Se p è falsa, qualsiasi valore di verità di q rende vero $p \to q$.

- $\forall p \forall q \exists r((p \wedge q) \to r)$;

Vero.

$(p \wedge q)$ implica almeno se stesso, indipendentemente da come p e q sono scelte, dunque una qualunque interpretazione I per cui $I \models r \iff I \models p \wedge q$ rende vera $(p \wedge q) \to r$.

- $\forall p \exists r \forall q((p \wedge q) \to r)$;

Vero.

Per qualsiasi valori di verità per p e q, si può scegliere r vera, rendendo vera anche l'implicazione $(p \wedge q) \to r$. Si noti che questo argomento vale anche per la formula precedente.

- $\forall p \exists r \forall q(r \to (p \wedge q))$.

Vero.

Per qualsiasi valori di verità per p e q, si può scegliere r falsa, rendendo vera l'implicazione $r \to (p \wedge q)$.

Esercizio 50

Si dica se le seguenti formule proposizionali quantificate sono vere oppure no, usando l'algoritmo del truth checking:

- $\forall p \exists q(p \to q)$;

Vero.

$TC(\forall p \exists q(p \to q), \emptyset)$
$= TC(\exists q(p \to q), \{p = 1\})$ and $TC(\exists q(p \to q), \{p = 0\})$
$= (TC(p \to q, \{p = 1, q = 1\})$ or $TC(p \to q, \{p = 1, q = 0\}))$
and $(TC(p \to q, \{p = 0, q = 1\})$ or $TC(p \to q, \{p = 0, q = 0\}))$
= ('YES' or 'NO') and ('YES' or 'YES')
= 'YES' and 'YES'
= 'YES'

- $\forall p \forall q (p \to q)$.

Falso.

$TC(\forall p \forall q(p \to q), \emptyset)$
$= TC(\forall q(p \to q), \{p = 1\})$ and $TC(\forall q(p \to q), \{p = 0\})$
$= (TC(p \to q, \{p = 1, q = 1\})$ or $TC(p \to q, \{p = 1, q = 0\}))$
and $(TC(p \to q, \{p = 0, q = 1\})$ or $TC(p \to q, \{p = 0, q = 0\}))$
= ('YES' and 'NO') and ('YES' and 'YES')
= 'NO' and 'YES'
= 'NO'

- $\forall p \exists r \forall q((p \wedge q) \to r)$;

Vero.

$TC(\forall p \exists r \forall q((p \wedge q) \to r), \emptyset)$
$= TC(\exists r \forall q((p \wedge q) \to r), \{p = 1\})$ and $TC(\exists r \forall q((p \wedge q) \to r), \{p = 0\})$
$= (TC(\forall q((p \wedge q) \to r), \{p = 1, r = 1\})$
or $TC(\forall q((p \wedge q) \to r), \{p = 1, r = 0\}))$
and $(TC(\forall q((p \wedge q) \to r), \{p = 0, r = 1\})$
or $TC(\forall q((p \wedge q) \to r), \{p = 0, r = 0\}))$
$= ((TC((p \wedge q) \to r, \{p = 1, r = 1, q = 1\})$
and $TC((p \wedge q) \to r, \{p = 1, r = 1, q = 0\}))$
or $(TC((p \wedge q) \to r, \{p = 1, r = 0, q = 1\})$
and $TC((p \wedge q) \to r, \{p = 1, r = 0, q = 0\})))$
and $((TC((p \wedge q) \to r, \{p = 0, r = 1, q = 1\})$
and $TC((p \wedge q) \to r, \{p = 0, r = 1, q = 0\}))$
or $(TC((p \wedge q) \to r, \{p = 0, r = 0, q = 1\})$
and $TC((p \wedge q) \to r, \{p = 0, r = 0, q = 0\})))$
$= (('YES'$ and $'YES')$
or $('NO'$ and $'YES'))$
and $(('YES'$ and $'YES')$ or $('YES'$ and $'YES'))$
$= ('YES'$ or $'NO')$ and $('YES'$ or $'YES')$
$= 'YES'$ and $'YES'$
$= 'YES'$

CAPITOLO 6

Complessità in Tempo

Esercizio 51

Data la seguente tabella che riassume le proprietà di chiusura di ciascuna classe di linguaggi, completare la riga riferita a P, dimostrando ciascun caso:

	unione	composizione	star di Kleene	intersezione	complemento	differenza insiemistica
REG	✓	✓	✓	✓	✓	✓
CFG	✓	✓	✓	✗	✗	✗
DEC	✓	✓	✓	✓	✓	✓
R.E.	✓	✓	✓	✓	✗	✗
P						

• **Complemento:** ✓ Se $\mathcal{M}$ è una TM che termina in $O(n^k)$ tempo per qualche k e decide un linguaggio L, allora la macchina $\overline{\mathcal{M}}$ che esegue $\mathcal{M}$ e inverte la risposta termina in $O(n^k)$ e decide $\overline{L}$;
• **Unione:** ✓ Siano L_1, L_2 due linguaggi decisi da due macchine di Turing $\mathcal{M}_1$, $\mathcal{M}_2$ rispettivamente in tempo $O(n_1^k)$ e $O(n_2^k)$ per qualche k_1, k_2. Una TM che esegue prima $\mathcal{M}_1$ e poi $\mathcal{M}_2$, accettando nel caso accettino entrambe, e rifiutando altrimenti, è una TM per l'unione $L_1 \cup L_2$ che termina in tempo $O(n^{k_1+k_2})$, che è ancora polinomico.
• **Composizione:** ✓ Supponiamo che L_1 ed L_2 siano linguaggi in P decisi da $\mathcal{M}_1$ (in tempo $O(n^{k_1})$)e $\mathcal{M}_2$ (in tempo $O(n^{k_2})$), e vogliamo mostrare che il linguaggio $L_1 \circ L_2 \in$ P. Per fare questo usiamo una macchina $\mathcal{M}$ che, con input w:
– Per tutti gli indici i tali che $w_1 w_2 \dots w_i \circ w_{i+1} w_{i+1} \dots w_{|w|} = w$, esegui $\mathcal{M}_1(w_1 w_2 \dots w_i)$ e $\mathcal{M}_2(w_{i+1} w_{i+1} \dots w_{|w|})$, e se entrambe hanno accettato, ACCETTA;
– Se il ciclo è terminato, RIFIUTA.
Questa macchina termina in $O((n+1) \cdot n^{max\{k_1,k_2\}})$, ed è quindi polinomica

in tempo;
• **Star di Kleene:** ✓ Sia $L \in P$, e $\mathcal{M}$ una TM per L che termina in $O(n^k)$ tempo per qualche k. Diamo un algoritmo di programmazione dinamica per L^*, mostrando che anche la sua complessità in tempo è polinomicio. Chiamiamo $w_{i,j}$ la sotto-parola $w_i \dots w_j$ di w, con $w = w_1 \dots w_n$. L'algoritmo costruisce una matrice Booleana $R[n, n]$, dove $R[i, j]$ è vero $\iff w_{i,j} \in L^*$, riempiendo i valori di tale matrice per diagonali (ovvero prima per sotto-parole di lunghezza 1, poi per parole di lunghezza 2, e così via). Si noti che $w \in L^*$ se e solo se è vera una delle tre condizioni *(i)* $w = \epsilon$, *(ii)* $w \in L$, oppure *(iii)* $\exists u, v, w = uv$ e $u, v \in L^*$; l'osservazione chiave è che, al momento della valutazione su una parola w, tutte e tre le condizioni sono tutte calcolabili in tempo polinomiale, considerato che la verità di $u, v \in L^*$ è già scritta in matrice, avendo u e v lunghezza inferiore di w. L'algoritmo completo procede così:
– se w è ϵ, ACCETTA;
– inizializza la matrice $R[n, n]$, assegnando falso ad ogni cella;
– per ogni $l = 1, \dots, n$ e per ogni $i = 1, \dots, n - (l - 1)$, ripeti
  – $j := i + (l - 1)$;
  – se $M(w_{i,j})$ accetta, allora assegna vero a $R[i, j]$;
  – se $M(w_{i,j})$ rifiuta, allora per ogni $k = i, \dots, j - 1$, ripeti
    – se $R[i, k]$ e $R[k + 1, j]$, allora assegna vero a $R[i, j]$
– se $R[1, n]$, ACCETTA, altrimenti RIFIUTA.
L'algoritmo presenta tre cicli annidati, ciascuno dei quali esegue al massimo $O(n)$ iterazioni; in più, il secondo ciclo più interno esegue una chiamata a M con una sotto-parola di w, che costa al massimo $O(n)$ tempo, mentre il ciclo più interno esegue una singola operazione di assegnazione (costo costante). Perciò, l'algoritmo ha complessità in tempo $O(n) \cdot O(n) \cdot O(n^k + n) = O(n^2 + max(k, 1))$, che è polinomiale in n.
• **Intersezione:** ✓ Siano L_1, L_2 due linguaggi decisi da due macchine di Turing $\mathcal{M}_1$, $\mathcal{M}_2$ rispettivamente in tempo $O(n_1^k)$ e $O(n_2^k)$ per qualche k_1, k_2. Una TM che esegue prima $\mathcal{M}_1$ e poi $\mathcal{M}_2$, accettando nel caso accettino entrambe, e rifiutando altrimenti, è una TM per l'intersezione $L_1 \cap L_2$ che termina in tempo $O(n^{k_1+k_2})$, che è ancora polinomico.
• **Differenza insiemistica:** ✓ Ragionando sulle proprietà degli insiemi: $L_2 \setminus L_2 = L_2 \cup \overline{L_2}$; essendo P chiuso per unione e complemento, la chiusura per differenza segue.

Esercizio 52

Mostrare che il seguente linguaggio è P:

$$RP = \{\langle n, m\rangle \mid n, m \in \mathbb{N} \text{ e } n, m \text{ sono relat. primi}\}.$$

Due numeri sono relativamente primi (o *coprimi*) se non hanno comuni divisori diversi da 1. Dunque, il problema di riconoscere se due numeri sono relativamente primi si può risolvere con un algoritmo che trovi il massimo comun divisore (MCD), e che infine verifichi se questo è 1. L'algoritmo di Euclide risolve il problema dell'MCD in tempo polinomiale, e pertanto $RP \in$ P.

Innanzitutto, diamo una TM che risolve il problema usando l'algoritmo di Euclide; con input x, la TM:

- Verifica che x sia la descrizione di due numeri interi n ed m in un qualche codifica (ad esempio la codifica binaria classica), rifiutando in caso negativo;
- Calcola l'algoritmo di Euclide per trovare l'mcd, accettando se questo è 1 e rifiutando altrimenti.

Dobbiam mostrare che l'algoritmo di Euclide, calcolato su due numeri a partire dalla loro descrizione binaria, è polinomiale. L'algoritmo, dati due numeri n e m:

function $gcd(n, m)$
$i \leftarrow n$
$j \leftarrow m$
if $i < j$ **then**
  $tmp \leftarrow j$
  $j \leftarrow i$
  $i \leftarrow tmp$
end if
while $m \neq 0$ **do**
  $t \leftarrow m$
  $m \leftarrow n \bmod m$
  $n \leftarrow t$
end while
return n

Esercizio 53

Data la seguente tabella che riassume le proprietà di chiusura di ciascuna classe di linguaggi, completare la riga riferita a NP, dimostrando ciascun caso:

	unione	composizione	star di Kleene	intersezione	complemento	differenza insiemistica
REG	✓	✓	✓	✓	✓	✓
CFG	✓	✓	✓	✗	✗	✗
DEC	✓	✓	✓	✓	✓	✓
R.E.	✓	✓	✓	✓	✗	✗
P	✓	✓	✓	✓	✓	✓
NP						

Si noti che la chiusura per complemento di NP non è mai stata provata né smentita; che cosa possiamo dire della chiusura per differenza?

- **Unione:** ✓ Si consideri una 2-TM non-deterministica, nella quale sul primo nastro viene simulata la TM per risolvere il primo problema, e sul secondo nastro la TM per risolvere il secondo: la 2-TM accetta se almeno una delle risposte dei due oracoli soddisfa le condizioni del problema;
- **Composizione:** ✓ Si consideri una 2-TM non-deterministica: sul primo nastro si pone la prima parte della composizione, e sul secondo la seconda. Per prima cosa, viene simulata la TM che riconosce la prima parte: se la simulazione termina con rifiuto, la 2-TM rifiuta. Altrimenti, prosegue simulando la seconda TM sul secondo nastro, accettando se la simulazione accetta, e rifiutando altrimenti;
- **Star di Kleene:** ✓ Si consideri una 2-TM non-deterministica che, per prima cosa, sceglie in maniera non-deterministica il modo in cui dividere la parola in input, ad esempio separando le diverse parti con un simbolo speciale (e accettando subito se l'input è la stringa vuota). Poi, per ciascuna parte, la copia sul secondo nastro, dove viene simulata la TM corrispondente: se la simulazione rifiuta, la macchina rifiuta, altrimenti continua finché non raggiunge la fine del primo nastro, accettando. Nel caso peggiore, esistono tanti pezzi quanti sono i caratteri della stringa in input; dunque, se la TM per riconoscere ciascuna parte è polinomiale, allora anche la TM risultante è polinomiale;
- **Intersezione:** ✓ Simile al caso dell'unione, ma accettando solo se entrambe le simulazioni accettano;
- **Differenza insiemistica:** ? Se sapessimo che NP è chiuso differenza insiemistica, allora sarebbe chiuso anche per complemento, poiché $\overline{L} = \Sigma^* \setminus L$

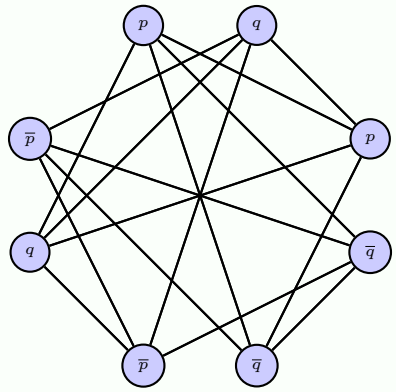
e $\Sigma^* \in$ NP.

Esercizio 54

Si consideri la formula proposizionale in forma $3CNF$

$$(p \vee q) \wedge (\neg p \vee q) \wedge (p \vee \neg q) \wedge (\neg p \vee \neg q).$$

Facendo riferimento al Teorema $3SAT \leq_p CLIQUE$, si ricavi il grafo corrispondente, e si discuta sulle conseguenze sul grafo della non soddisfacibilità della formula.



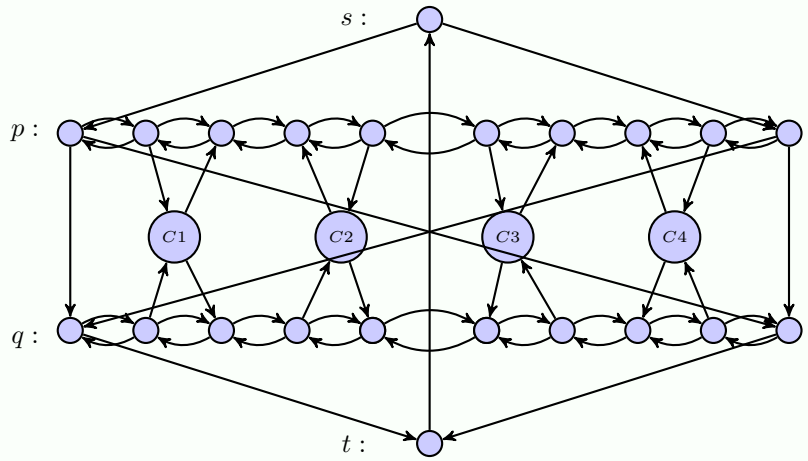
Il grafo è altamente simmetrico, per cui è facile vedere che, nonostante esistano diverse 3-clique, a nessuna di queste è possibile aggiungere un quarto nodo. Infatti, poiché la formula non è soddisfacibile, non esiste una 4-clique nel grafo (e viceversa).

Esercizio 55

Si consideri la formula proposizionale in forma $3CNF$

$$(p \vee q) \wedge (\neg p \vee q) \wedge (p \vee \neg q) \wedge (\neg p \vee \neg q).$$

Facendo riferimento al Teorema $3SAT \leq_p HAM$, si ricavi il grafo corrispondente, e si discuta sulle conseguenze sul grafo della non soddisfacibilità della formula.



Nel grafo non esiste nessun un ciclo Hamiltoniano, in quanto non è possibile percorrere le due *traccie* corrispondenti a p e q in nessun verso (da sinistra a destra, oppure da destra a sinistra), tale che il percorso possa coprire contemporaneamente tutti e quattro i nodi $C1$, $C2$, $C3$, e $C4$. Infatti, poiché la formula non è soddisfacibile, non esiste un ciclo Hamiltoniano nel grafo (e viceversa).

Esercizio 56

Facendo riferimento al Teorema $HAM \leq_p UHAM$, si dimostri che, se il grafo di partenza non ha un ciclo Hamiltoniano, allora non lo ha neppure il grafo di arrivo.

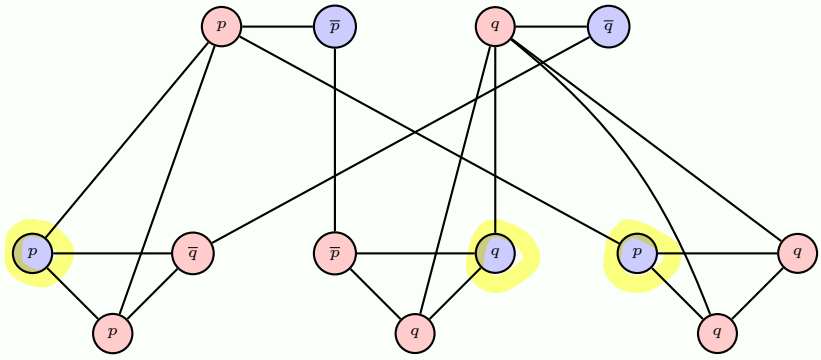
Usando una notazione più conveniente rispetto a quella presente sulle trasparenze, ricordiamo che la trasformazione richiede di creare, per ciascun nodo v_i del grafo di partenza, tre nodi $v_{i,s}, v_i, v_{i,e}$ nel grafo di arrivo, legati a modo di catena da due archi $(v_{i,s}, v_i), (v_i, v_{i,e})$, e tali che tutti gli archi che erano originariamente entranti (uscenti) in v siano ora entranti in $v_{i,s}$ ($v_{i,e}$). Dimostriamo il risultato dimostrando che se il grafo di arrivo ha un ciclo Hamiltoniano allora anche il grafo di partenza ne ha uno. Ipotizziamo allora che il grafo di arrivo abbia un ciclo Hamiltoniano π_u. A causa della struttura di questo grafo, il ciclo avrà la seguente forma: $\pi_u = v_{p_0,s} \rightsquigarrow v_{p_0} \rightsquigarrow v_{p_0,e} \rightsquigarrow \ldots \rightsquigarrow v_{p_n,s} \rightsquigarrow v_{p_n} \rightsquigarrow v_{p_n,e} \rightsquigarrow v_{p_0,s}$. Allora, il grafo originale avrà dunque almeno un ciclo Hamiltoniano diretto π_d, costituito da $v_{p_0} \rightsquigarrow \ldots \rightsquigarrow v_{p_n} \rightsquigarrow v_{p_0}$. Questo vale perché: *(i)* per ogni sequenza di tre nodi $v_{p_j,s} \rightsquigarrow v_{p_j} \rightsquigarrow v_{p_j,e}$ in π_u, il grafo originale ha un vertice v_{p_j}, e se π_u contiene tutti i vertici del grafo di arrivo, allora π_d contiene tutti i vertici del grafo di partenza, e *(ii)* essendo π_u è un ciclo Hamiltoniano, allora ogni coppia $v_{p_j,e} \rightsquigarrow v_{p_{j+1},s}$ (e così per $v_{p_n,e} \rightsquigarrow v_{p_0,s}$) in π_u mostra che nel grafo di partenza esisteva un arco $(v_{p_j}, v_{p_{j+1}})$ $((v_{p_n}, v_{p_0}))$.

Esercizio 57

Si consideri la seguente formula proposizionale in forma $3CNF$

$$(p \vee p \vee \neg q) \wedge (\neg p \vee q \vee q) \wedge (p \vee q \vee q).$$

Facendo riferimento alla dimostrazione $3SAT \leq_p COVER$, si applichi la costruzione vista nel teorema, ricavando il grafo e il valore k corrispondenti. È questa formula soddisfacibile? Se sì, si indichi almeno una k-copertura, e si trovi l'interpretazione corrispondente che rende la formula vera.



La formula ha 3 clausole e 2 proposizioni, perciò $k = 2 \cdot 3 + 2 = 8$. La formula è soddisfacibile, e il grafo presenta diverse 8-coperture; nella figura ne è stata mostrata una, la cui interpretazione corrispondente è $\{p = 1, q = 1\}$.

Esercizio 58

Si identifichi e spieghi l'errore nella seguente dimostrazione. Sia $L = \{0^n 1^n \mid n \geq 0\}$, mostriamo $L \leq_p COVER$ tramite la seguente riduzione:

$$f(\langle G, k \rangle) = \begin{cases} 01 & \text{se } G \text{ ha una } k\text{-copertura} \\ 0 & \text{altrimenti} \end{cases}$$

Dato che $COVER \in \text{NPC}$, $L \leq_p COVER$ e $L \in P$, otteniamo che $P = NP$.

L'errore sta nel fatto che la riduzione f non è necessariamente calcolabile in tempo polinomiale; in particolare, lo è se $COVER \in P$, fatto la cui verità non è mai stata provata (ed è, anzi, improbabile).

Esercizio 59

Data la seguente tabella che riassume le proprietà di chiusura di ciascuna classe di linguaggi, completare la riga riferita a NPC, dimostrando ciascun caso:

	unione	composizione	star di Kleene	intersezione	complemento	differenza insiemistica
REG	✓	✓	✓	✓	✓	✓
CFG	✓	✓	✓	✗	✗	✗
DEC	✓	✓	✓	✓	✓	✓
R.E.	✓	✓	✓	✓	✗	✗
P	✓	✓	✓	✓	✓	✓
NP	✓	✓	✓	✓		
NPC			●			

Per queste dimostrazioni assumeremo $\Sigma = \{0, 1\}$, e useremo il fatto che non sono NPC né il linguaggio vuoto $L_\emptyset = \emptyset$, né il linguaggio Σ^*.

- **Intersezione: ✗** Dato un linguaggio $L \in$ NPC, si considerino $L_0 = \{0w \mid w \in L\}, L_1 = \{1w \mid w \in L\}$. Chiaramente, L_0 e L_1 sono ancora NPC, ma la loro intersezione è il linguaggio vuoto, che non è NPC;
- **Unione: ✗** Considerati ancora i due linguaggi $L_0, L_1 \in$ NPC, del caso precedente, si consideri il linguaggio $L'_0 = L_0 \cup \epsilon$. Chiaramente esso è ancora NPC, ma l'unione $L'_0 \cup L_1$ è Σ^*, che non è NPC;
- **Composizione: ✗** Considerati gli stessi linguaggi del caso precedente, è facile vedere come è possibile ottenere qualsiasi parola tramite la loro composizione; dunque la loro composizione è Σ^*, che non è NPC;
- **Star di Kleene: ✗** Dato un linguaggio $L \in$ NPC, il linguaggio $\{0, 1\} \cup L$ è ancora NPC, ma la sua chiusura è Σ^*;
- **Differenza insiemistica: ✗** Dato un linguaggio $L \in$ NPC, $L \setminus L = L_\emptyset$, che non è NPC.

CAPITOLO 7

Complessità in Spazio

Esercizio 60

Data la seguente tabella che riassume le proprietà di chiusura di ciascuna classe di linguaggi, completare la riga riferita a PSPACE, dimostrando ciascun caso:

	unione	composizione	star di Kleene	intersezione	complemento	differenza insiemistica
REG	✓	✓	✓	✓	✓	✓
CFG	✓	✓	✓	✗	✗	✗
DEC	✓	✓	✓	✓	✓	✓
R.E.	✓	✓	✓	✓	✗	✗
P	✓	✓	✓	✓	✓	✓
NP	✓	✓	✓	✓		
NPC	✗	✗	✗	✗		✗
PSPACE				●	●	

- **Unione:** ✓ Siano L_1 e L_2 due linguaggi decisi in spazio polinomiale da due macchine di Turing (anche non-deterministiche) $\mathcal{M}_1$ e $\mathcal{M}_2$. Una NTM $\mathcal{M}$ che esegue prima $\mathcal{M}_1$ e poi $\mathcal{M}_2$, accettando non appena una delle due componenti termina accettando, e rifiutando se entrambe hanno terminato rifiutando, riconosce l'unione di L_1 e L_2 in spazio polinomiale. Infatti, lo spazio necessario è pari a quello massimo necessario per ciascuna delle due TM originali;
- **Composizione:** ✓ Similmente al caso precedente, una NTM può operare un passo non-deterministico nel quale l'input viene diviso in due parti, e vengono eseguite prima $\mathcal{M}_1$ sulla prima parte e poi $\mathcal{M}_2$ sulla seconda parte. Lo spazio occupato rimane polinomiale;
- **Complemento:** ✓ Sia L un linguaggio deciso in spazio polinomiale da una macchina di Turing $\mathcal{M}$; allora, la macchina $\overline{\mathcal{M}}$ che esegue $\mathcal{M}$ e inverte la risposta decide in spazio polinomiale $\overline{L}$;

• **Star di Kleene:** ✓ Sia L un linguaggio deciso da una macchina di Turing $\mathcal{M}$ in spazio $O(n^k)$ tempo per qualche k. Una NTM $\mathcal{M}'$ può decidere L^* operando prima un passo non-deterministico di divisione della parola in n sotto-parole, eseguendo $\mathcal{M}$ su ciascuna sotto-parola, infine accettando solo se tutte le esecuzioni hanno accettato. Le esecuzioni possono essere eseguite sequenzialmente, per cui lo spazio necessario rimane $O(n^k)$;
• **Intersezione:** ✓ Ragionando sulle proprietà degli insiemi: $L_1 \cap L_2 = \overline{(\overline{L_1} \cup \overline{L_2})}$; essendo PSPACE chiuso per unione e complemento, la chiusura per intersezione segue;
• **Differenza insiemistica:** ✓ Ragionando sulle proprietà degli insiemi: $L_1 \setminus L_2 = L_1 \cup \overline{L_2}$; essendo PSPACE chiuso per unione e complemento, la chiusura per differenza segue.

Esercizio 61

Si consideri una TM deterministica e terminante con complessità in spazio $O(f(n))$. Possiamo dare un limite superiore alla sua complessità in tempo?

Sì. Con una complessità in spazio è limitata in $O(f(n))$, il numero di possibili posizioni per la testina è anch'esso limitato a $O(f(n))$. Pertanto, se q è il numero di stati della TM e s è il numero di simboli del nastro, il numero di diverse configurazioni è limitato da $O(qn^3s^{f(n)}) = O(2^{f(n)})$. Una macchina deterministica e terminante non può mai entrare nella stessa configurazione più volte, e dunque questa espressione limita anche la complessità in tempo.

Esercizio 62

Dimostrare che $GG \in$ PSPACE.

Si può dare dimostrazione simile a quella presente sulle trasparenze per $QSAT \in$ PSPACE: in effetti, lo schema dell'algoritmo del truth checking può essere modificato per risolvere anche GG. Con un argomento booleano is_S_turn inizializzato a vero, e un vertice di riferimento q inizializzato al vertice di partenza v, l'algoritmo esegue:
– Se q non ha vicini, rispondi 'Sì' se is_S_turn è vero, e 'No' se is_S_turn è falso;
– Se is_S_turn, richiamati con is_S_turn falso su ciascun vicino di q; rispondi 'Sì' se almeno una chiamata ha risposto 'Sì', e 'No' in caso contrario;
– Se is_S_turn è falso, richiamati con is_S_turn vero su ciascun vicino di q; rispondi 'Sì' se tutte le chiamate hanno risposto 'Sì', e 'No' in caso contrario.
Poiché ogni possibile sequenza di mosse non è più lunga del numero di vertici, una macchina deterministica che visita l'albero di ricorsione in profondità

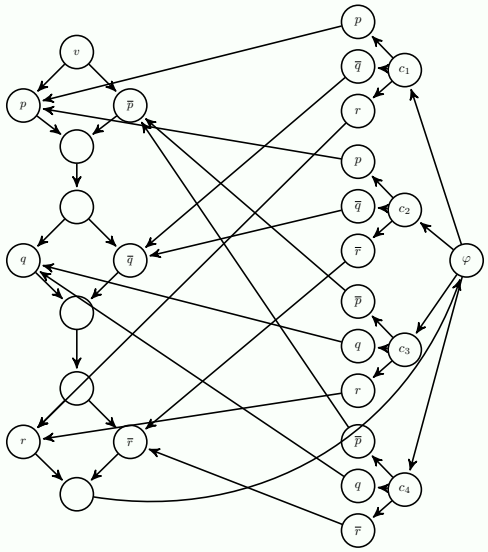
utilizzerà $O(|V|)$ spazio, necessario per memorizzare la lista di vertici del nodo di ricorsione corrente. Pertanto $GG \in$ PSPACE.

Esercizio 63

Si consideri la seguente formula proposizionale quantificata

$$\exists p \forall q \exists r((p \vee \neg q \vee r) \wedge (p \vee \neg q \vee \neg r) \wedge (\neg p \vee q \vee r) \wedge (\neg p \vee q \vee \neg r)).$$

Facendo riferimento alla dimostrazione $QSAT \leq_p GG$, si applichi la costruzione vista nel teorema, ricavando il grafo G corrispondente, e indicando il vertice di inizio v del gioco. È questa formula vera? Si mostrino le conseguenze sul gioco (G, v).



La formula non è vera; infatti, scelto un qualsiasi valore di verità per p, scelta che rende vere le prime due clausole oppure le ultime due clausole, assegnando il valore opposto a q si possono rendere le due clausole rimanenti semanticamente incompatibili, e quindi la formula insoddisfacibile. Questo schema è visibile anche nel grafo: focalizzandosi sulla parte destra della figura, per qualsiasi scelta tra p e $\overline{p}$, l'avversario può scegliere rispettivamente $\overline{q}$ o q, causando la copertura per due terzi di entrambe le clausole $c1$ e $c2$, oppure di entrambe le clausole $c3$ e $c4$. A questo punto, qualsiasi scelta tra r e $\overline{r}$ coprirà totalmente una delle due clausole; pertanto, andando avanti nel gioco, e quindi spostandosi sulla parte sinistra del grafo, l'avversiario potrà scegliere il nodo associato a tale clausola, e il giocatore S si troverà "tappato", perdendo la partita.

Esercizio 64

Si dimostri che PSPACE-hard $\subseteq$ NP-hard.

Un linguaggio PSPACE-hard è tale che ogni linguaggio in PSPACE si possa ridurre ad esso; pertanto, anche tutti i problemi in NP si possono ridurre ad esso, e quindi è anche NP-hard.

Esercizio 65

Data la seguente tabella che riassume le proprietà di chiusura di ciascuna classe di linguaggi, completare la riga riferita a LOG, dimostrando ciascun caso:

	unione	composizione	star di Kleene	intersezione	complemento	differenza insiemistica
REG	✓	✓	✓	✓	✓	✓
CFG	✓	✓	✓	✗	✗	✗
DEC	✓	✓	✓	✓	✓	✓
R.E.	✓	✓	✓	✓	✗	✗
P	✓	✓	✓	✓	✓	✓
NP	✓	✓	✓	✓		
NPC	✗	✗	✗	✗		✗
PSPACE	✓	✓	✓	✓	✓	✓
LOG						

Si noti che il risultato di chiusura sotto star di Kleene non è noto, ma è possibile dimostrare che esso è condizione necessaria e sufficiente per LOG = NLOG e, pertanto, si crede che questa chiusura non sussista.

- **Unione:** ✓ Siano L_1, L_2 due linguaggi decisi da due macchine di Turing $\mathcal{M}_1$, $\mathcal{M}_2$ rispettivamente in tempo $O(k_1 lg(n))$ e $O(k_2 lg(n))$ per qualche k_1, k_2. Una TM che esegue prima $\mathcal{M}_1$ e poi $\mathcal{M}_2$, accettando nel caso accettino entrambe, e rifiutando altrimenti, è una TM per l'unione $L_1 \cup L_2$ che termina in tempo $O((k_1 + k_2) lg(n))$, che è ancora logaritmico;
- **Composizione:** ✓ Siano L_1, L_2 due linguaggi LOG decisi in spazio logaritmico da due macchine di TM deterministiche $\mathcal{M}_1$ e $\mathcal{M}_2$. Una macchina $\mathcal{M}$ può riconoscere $L_1 \circ L_2$ testando, per tutte le possibili divisioni in due sottostringhe della parola in input, se le due sottostringhe sono riconosciute rispettivamente da $\mathcal{M}_1$ e $\mathcal{M}_2$. Essendo il numero di le possibili divisioni sono

più che logaritmico ($n-1=\Theta(n)$), per restare in spazio logaritmico $\mathcal{M}$ non scriverà in maniera esplicita le due sottostringhe, ma mantenere in memoria un contatore che indicizza la fine della prima delle due sottostringhe (e che, quindi, determina anche l'inizio della seconda). In definitiva, $\mathcal{M}$ esegue un ciclo dove questo contatore i viene inizializzato a 0, e incrementato di 1 fino a $n-1$, e ad ogni iterazione, entra in una subroutine che consiste in $\mathcal{M}_1$ eseguita sul nastro di input, ma limitata alla posizione i, e similmente entra in una subroutine che consiste in $\mathcal{M}_2$ eseguita sul resto del nastro di input;
• **Intersezione:** ✓ Siano L_1, L_2 due linguaggi decisi da due macchine di Turing $\mathcal{M}_1$, $\mathcal{M}_2$ rispettivamente in tempo $O(k_1 lg(n))$ e $O(k_2 lg(n))$ per qualche k_1, k_2. Una TM che esegue prima $\mathcal{M}_1$ e poi $\mathcal{M}_2$, accettando nel caso accettino entrambe, e rifiutando altrimenti, è una TM per l'intersezione $L_1 \cap L_2$ che termina in tempo $O((k_1+k_2)lg(n))$, che è ancora polinomico;
• **Complemento:** ✓ Sia $\mathcal{M}$ è una TM che termina in $O(klg(n))$ tempo per qualche k e decide un linguaggio L; la macchina $\overline{\mathcal{M}}$ che esegue $\mathcal{M}$ e inverte la risposta termina in $O(klg(n))$ e decide $\overline{L}$;
• **Differenza insiemistica:** ✓ Ragionando sulle proprietà degli insiemi: $L_1 \setminus L_2 = L_1 \cup \overline{L_2}$; essendo LOG chiuso per unione e complemento, la chiusura per differenza segue.

Esercizio 66

Dimostrare che il linguaggio A_{DFA} è LOG.

È possibile simulare un DFA utilizzando solamente due contatori, che indichino rispettivamente lo stato del DFA corrente e la posizione della testina di lettura. Il secondo contatore viene iterativamente incrementato di una unità, mentre quello dello stato viene sovrascritto a seconda del grafo di transizione. La macchina infine accetta se, quando il secondo contatore è pari a $|w|$ (dove w è la parola in input all'automa), lo stato puntato dal secondo contatore è accettante, e rifiuta altrimenti. Si noti che prima della simulazione, la macchina deve eseguire un controllo sul formato della stringa in input, che però non richiede spazio aggiuntivo; pertanto, l'intero processo costa spazio logaritmico.

Esercizio 67

Dimostrare che il seguente linguaggio è LOG:

$$BIN = \{\langle a, b, c\rangle \mid a, b, c \text{ sono interi in binario e } a \cdot b = c\}$$

L'algoritmo della moltiplicazione tipico richiede spazio lineare nella lunghezza degli input (ovvero, richiede al massimo il doppio dello spazio utile per memorizzare il più grande dei due numeri); pertanto, non è utile in questo

contesto. È possibile, però, sfruttare il fatto che il problema non richiede di calcolare il risultato della moltiplicazione, ma solo di controllare che quello fornito nella stringa di input sia corretto. Quindi, tenendo conto che si vogliono usare solo contatori, sul nastro di lavoro istanziamo: *(i)* un indice per tenere traccia della cifra di c da controllare (inizializzato all'indice dell'ultima cifra di c), *(ii)* due indici relativi alle cifre dei due fattori a e b, *(iii)* un accumulatore che indica il risultato corretto della cifra attualmente in considerazione, e *(iv)* un accumulatore che indica il riporto. Poiché il risultato delle cifre di peso minore non è influenzato da quelle di peso maggiore, si possono riciclare gli accumulatori del risultato e del riporto ogni volta che è terminato il confronto della cifra precedente. Di nuovo, si noti che il controllo sul formato della stringa in input costa solo tempo, e non spazio; pertanto questo problema è LOG.

Esercizio 68

Data la seguente tabella che riassume le proprietà di chiusura di ciascuna classe di linguaggi, completare la riga riferita a NLOG, dimostrando ciascun caso:

	unione	composizione	star di Kleene	intersezione	complemento	differenza insiemistica
REG	✓	✓	✓	✓	✓	✓
CFG	✓	✓	✓	✗	✗	✗
DEC	✓	✓	✓	✓	✓	✓
R.E.	✓	✓	✓	✓	✗	✗
P	✓	✓	✓	✓	✓	✓
NP	✓	✓	✓	✓		
NPC	✗	✗	✗	✗		✗
PSPACE	✓	✓	✓	✓	✓	✓
LOG	✓	✓		✓	✓	✓
NLOG					✓	

Si noti che in questo contesto ci limitiamo a dare come risultato noto la chiusura per complemento (Teorema di Immerman-Szelepcsényi).

- **Unione:** ✓ Siano L_1, L_2 due linguaggi decisi in spazio logaritmico da due NTM $\mathcal{M}_1$, $\mathcal{M}_2$. Una NTM che opera un passo non-deterministico scegliendo se eseguire $\mathcal{M}_1$ o $\mathcal{M}_2$ riconosce l'unione $L_1 \cup L_2$ in spazio logaritmico;

• **Composizione:** ✓ Similmente al caso precedente, una NTM può operare un passo non-deterministico nel quale l'input viene diviso in due parti, e vengono eseguite prima $\mathcal{M}_1$ sulla prima parte e poi $\mathcal{M}_2$ sulla seconda parte. La divisione in due parti è codificabile tramite un indice sulla parola in input, pertanto lo spazio occupato rimane logaritmico;
• **Stella di Kleene:** ✓ Sia L un linguaggio deciso da una NTM $\mathcal{M}$ in spazio $O(klg(n))$ tempo per qualche k. Una NTM $\mathcal{M}'$ può decidere L^* operando prima un passo non-deterministico di divisione della parola in n sotto-parole, eseguendo $\mathcal{M}$ su ciascuna sotto-parola, infine accettando solo se tutte le esecuzioni hanno accettato. Tuttavia, è necessario dettagliare meglio il modo viene effettuata la divisione non-deterministica, poiché per memorizzare una divisione in n sotto-parole sono necessari un numero più che logaritmico di indici ($n-1 = \Theta(n)$). Il non-determinismo viene in aiuto, rendendo possibile la divisione utilizzando due soli indici x e y. L'intero algoritmo, con input w, procede come segue:
– Inizializza y alla posizione 0 della parola in input;
– Ripeti:
  – Accetta se $y = |w|$;
  – Assegna $x := y$, e sposta avanti y di qualche posizione in maniera non-deterministica;
  – Esegui $\mathcal{M}$ sulla porzione di w tra le posizioni x e y;
  – Se l'esecuzione rifiuta, rifiuta.

Le esecuzioni vengono eseguite sequenzialmente, per cui lo spazio necessario rimane $O(klg(n))$;
• **Intersezione:** ✓ Ragionando sulle proprietà degli insiemi: $L_1 \cap L_2 = \overline{(\overline{L_1} \cup \overline{L_2})}$; essendo NLOG chiuso per unione e complemento, la chiusura per intersezione segue;
• **Differenza insiemistica:** ✓ Ragionando sulle proprietà degli insiemi: $L_1 \setminus L_2 = L_1 \cup \overline{L_2}$; essendo NLOG chiuso per unione e complemento, la chiusura per differenza segue.

Esercizio 69

Diciamo che un grafo indiretto è *bipartito* se e solo se è possibile separare i vertici in due gruppi in maniera che tutti gli archi vanno da un gruppo all'altro. Dimostrare che il seguente linguaggio è NLOG:

$$BIP = \{\langle G \rangle \mid G \text{ è un grafo indiretto bipartito}\}$$

Iniziamo mostrando che un grafo è bipartito se e solo se non ha un ciclo di un numero dispari di nodi. Dopodiché, mostriamo che il problema di determinare se un grafo ha un ciclo di un numero dispari di nodi, equivalente a $\overline{BIPARTITE}$, è NLOG, e il risultato segue dalla chiusura di NLOG per complemento.

Dunque, sia $G = (V, E)$ un grafo bipartito, e $A, B \subset V$ gli insiemi internamente disconnessi che lo compongono. Supponiamo per assurdo che G abbia un ciclo dispari, che chiamiamo $n_1, n_2, \ldots, n_{2k}, n_{2k+1}$. Per la struttura di un grafo bipartito, è chiaro che se n_1 appartiene a A, n_2 appartiene a B, e così via. Questo implica che i nodi n_1 e n_{2k+1} appartengono allo stesso insieme, il che contraddice l'ipotesi di bipartizione, dato che sono tra loro connessi.
Supponiamo invece che un grafo non abbia un ciclo dispari, e mostriamo che è necessariamente bipartito. Eseguiamo un 2-coloring sul grafo secondo la seguente procedura: scegliamo un nodo v, e lo coloriamo di rosso; coloriamo di nero i suoi vicini; di nuovo, coloriamo di rosso i loro vicini, e di nero i loro vicini. Ripetendo questa procedura, a fine colorazione avremmo partizionato i nodi in due insiemi A e B, che sono una valida bipartizione. Infatti, non esisteranno mai nodi vicini colorati dello stesso colore; per mostrarlo, ipotizziamo per assurdo di trovare due nodi u e t dello stesso colore: questo comporta che esistono due cammino pari da v a u e da v a t, ma l'unione di questi due cammino, che porta da u a t passando per v, è un cammino dispari, e costituisce anche un ciclo dispari considerando che u e t sono connessi.
Finalmente, mostriamo che il problema $\overline{BIPARTITE}$ è in NLOG tramite il seguente algoritmo per riconoscere se un grafo ha un ciclo dispari. Con input G, l'algoritmo:
- Controlla che G sia nel formato corretto;
- Inizializza un contatore i a 0;
- Scegli non-deterministicamente un nodo iniziale u;
- Scegli non-deterministicamente un suo successore v;
- Finché $i \leq |V|$:
  - Se $v = u$ e i è un numero dispari, accetta;
  - Assegna a v un suo successore scelto non-deterministicamente;
  - Incrementa i di 1.

L'algoritmo usa solo tre contatori con valore massimo $|V|$, ed è dunque logaritmico in spazio.

Esercizio 70

Diciamo che un grafo diretto è *fortemente connesso* se e solo se per ogni coppia di vertici v e w esiste un cammino da v a w. Dimostrare che il seguente linguaggio è NLOG-completo:

$$CONN = \{\langle G \rangle \mid G \text{ è un grafo diretto fortemente connesso}\}$$

Il risultato richiede di dimostrare $CONN \in$ NLOG e $CONN \in$ NLOG-hard. Dimostriamo $CONN \in$ NLOG dando una TM per $\overline{CONN}$, e considerando che $NLOG = CoNLOG$, e che quindi NLOG è chiuso per complemento. La macchina, con input x:

– Controlla che l'input sia in formato $\langle G \rangle$;
– Scegli non-deterministicamente due nodi v e w;
– Esegui $PATH(v, w)$, rifiutando se l'esecuzione accetta, e accettando altrimenti.

Per dimostrare $CONN \in$ NLOG-hard diamo una riduzione per $PATH \leq_l CONN$, che con input $\langle G, s, t \rangle$. La riduzione, dato un input $\langle G, s, t \rangle$, produce $\langle G' \rangle$, dove G' consiste nel grafo G esteso con archi aggiuntivi; in particolare, vengono aggiunti un arco (v, s) e un arco (t, v) per ogni vertice v presente in G'. Se esisteva un cammino da s a t in G, allora G' è fortemente connesso, poiché tra due nodi qualunque esiste sempre il cammino che passa per s, i nodi sul cammino da s a t, e t. Viceversa, se non esisteva un cammino da s a t in G, allora G' non è fortemente connesso, poiché gli unici archi aggiunti sono uscenti da t ed entranti in s.

Esercizio 71

Si consideri la classe di complessità REG: come si rapporta rispetto a PSPACE, NLOG e LOG?

Per rispondere alla domanda, si consideri il costo in termini di spazio di eseguire le computazioni tipiche degli automi regolari su una macchina di Turing. La caratteristica degli automi regolari è che non hanno memoria, infatti, il grafo di transizione, che costituisce il cuore dell'algoritmo, è direttamente traducibile ad un grafo di transizione di una TM che non scrive nulla sul nastro. Questo comporta che REG $\subset$ NSPACE(1), e quindi che REG $\subset$ PSPACE, REG $\subset$ NLOG e REG $\subset$ LOG.