Automi e Linguaggi (M. Cesati)

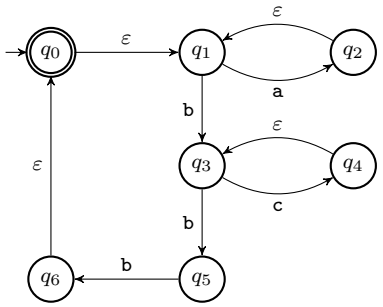
Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 1 settembre 2020

(La prova è stata svolta "a libri aperti".)

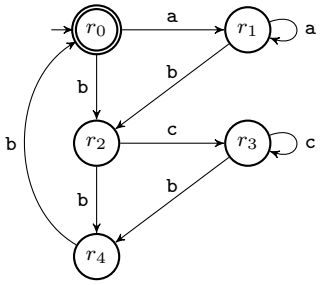
Esercizio 1. Determinare un automa deterministico che riconosca il linguaggio generato dalla espressione regolare $((\texttt{a}^*\texttt{bc}^*)\texttt{bb})^*$.

Soluzione: L'esercizio si può risolvere in modo totalmente meccanico derivando innanzi tutto un NFA dalla espressione regolare, e successivamente trasformando lo NFA in un DFA. Applicando qualche semplificazione allo NFA derivato dalla espressione regolare si ottiene:



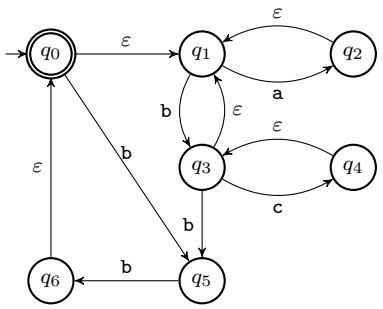
Un automa deterministico equivalente è il seguente:

$r_0 = \{q_0, q_1\},\ \{q_1, q_0, q_6\}$
$r_1 = \{q_1, q_2\}$
$r_2 = \{q_3\}$
$r_3 = \{q_3, q_4\}$
$r_4 = \{q_5\}$



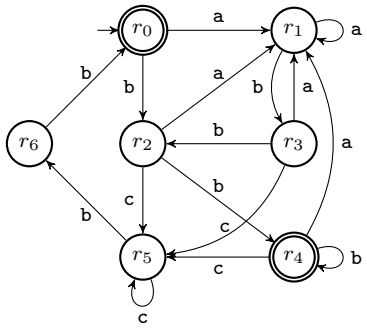
Esercizio 2. Determinare un automa deterministico che riconosca il linguaggio generato dalla espressione regolare $((\texttt{a}^*\texttt{bc}^*)^*\texttt{bb})^*$.

Soluzione: L'esercizio si può risolvere in modo totalmente meccanico derivando innanzi tutto un NFA dalla espressione regolare, e successivamente trasformando lo NFA in un DFA. Applicando qualche semplificazione allo NFA derivato dalla espressione regolare si ottiene:



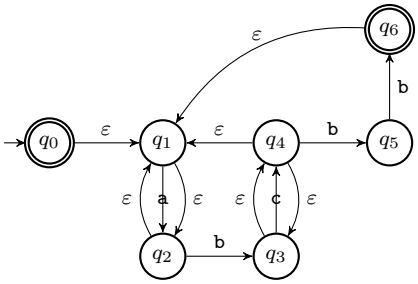
Un automa deterministico equivalente è il seguente:

$r_0 = \{q_0, q_1\}$
$r_1 = \{q_1, q_2\}$
$r_2 = \{q_1, q_3, q_5\}$
$r_3 = \{q_1, q_3\}$
$r_4 = \{q_0, q_1, q_3, q_5, q_6\}$
$r_5 = \{q_3, q_4\}$
$r_6 = \{q_5\}$



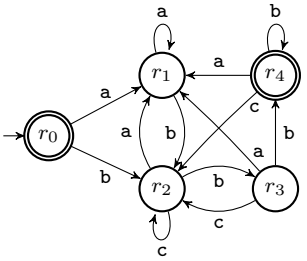
Esercizio 3. Determinare un automa deterministico che riconosca il linguaggio generato dalla espressione regolare $((\mathtt{a}^*\mathtt{bc}^*)^+\mathtt{bb})^*$.

Soluzione: [Rev. 17.06.2022] La notazione x^+ è una abbreviazione per $x\,x^*$. L'esercizio si può risolvere in modo totalmente meccanico derivando innanzi tutto un NFA dalla espressione regolare, e successivamente trasformando lo NFA in un DFA. Applicando qualche semplificazione allo NFA derivato dalla espressione regolare si ottiene:



Un automa deterministico equivalente è il seguente:

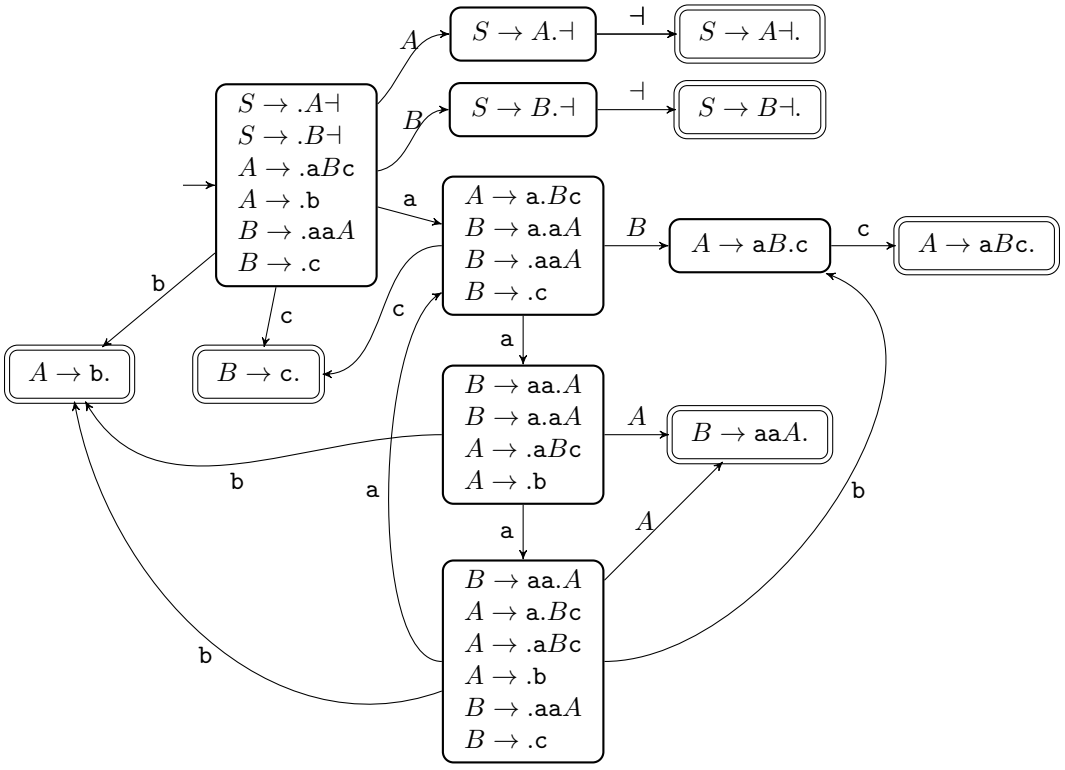
$r_0 = \{q_0, q_1, q_2\}$
$r_1 = \{q_1, q_2\}$
$r_2 = \{q_1, q_2, q_3, q_4\}$
$r_3 = \{q_1, q_2, q_3, q_4, q_5\}$
$r_4 = \{q_1, q_2, q_3, q_4, q_5, q_6\}$



Esercizio 4. Determinare se la seguente grammatica CFG con variabile iniziale S è deterministica:

$$\begin{array}{lcl} S \to & A\dashv & \mid \quad B\dashv \\ A \to & \mathtt{a}B\mathtt{c} & \mid \quad \mathtt{b} \\ B \to & \mathtt{aa}A & \mid \quad \mathtt{c} \end{array}$$

Soluzione: Per determinare se la grammatica è deterministica eseguiamo il DK-test, ottenendo così il seguente diagramma:

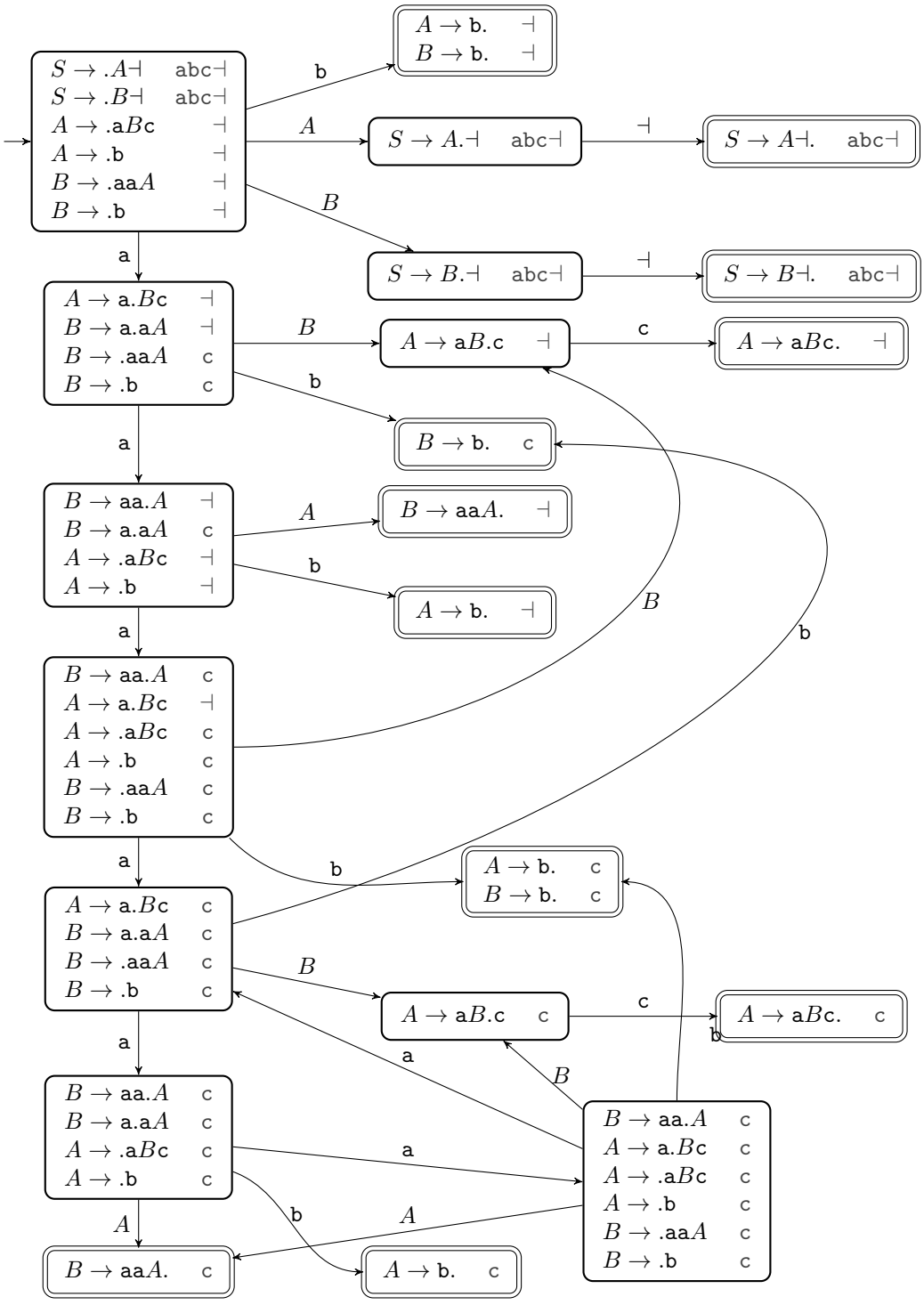


Poiché ogni stato finale ha una sola regola, quella completata, il DK-test risulta soddisfatto, pertanto la grammatica analizzata è deterministica.

Esercizio 5. Determinare se la seguente grammatica CFG con variabile iniziale S è LR(1):

$$\begin{aligned} S &\to A\dashv \mid B\dashv \\ A &\to \mathtt{a}B\mathtt{c} \mid \mathtt{b} \\ B &\to \mathtt{aa}A \mid \mathtt{b} \end{aligned}$$

Soluzione: Per determinare se la grammatica è deterministica eseguiamo il DK_1-test, ottenendo così il seguente diagramma:

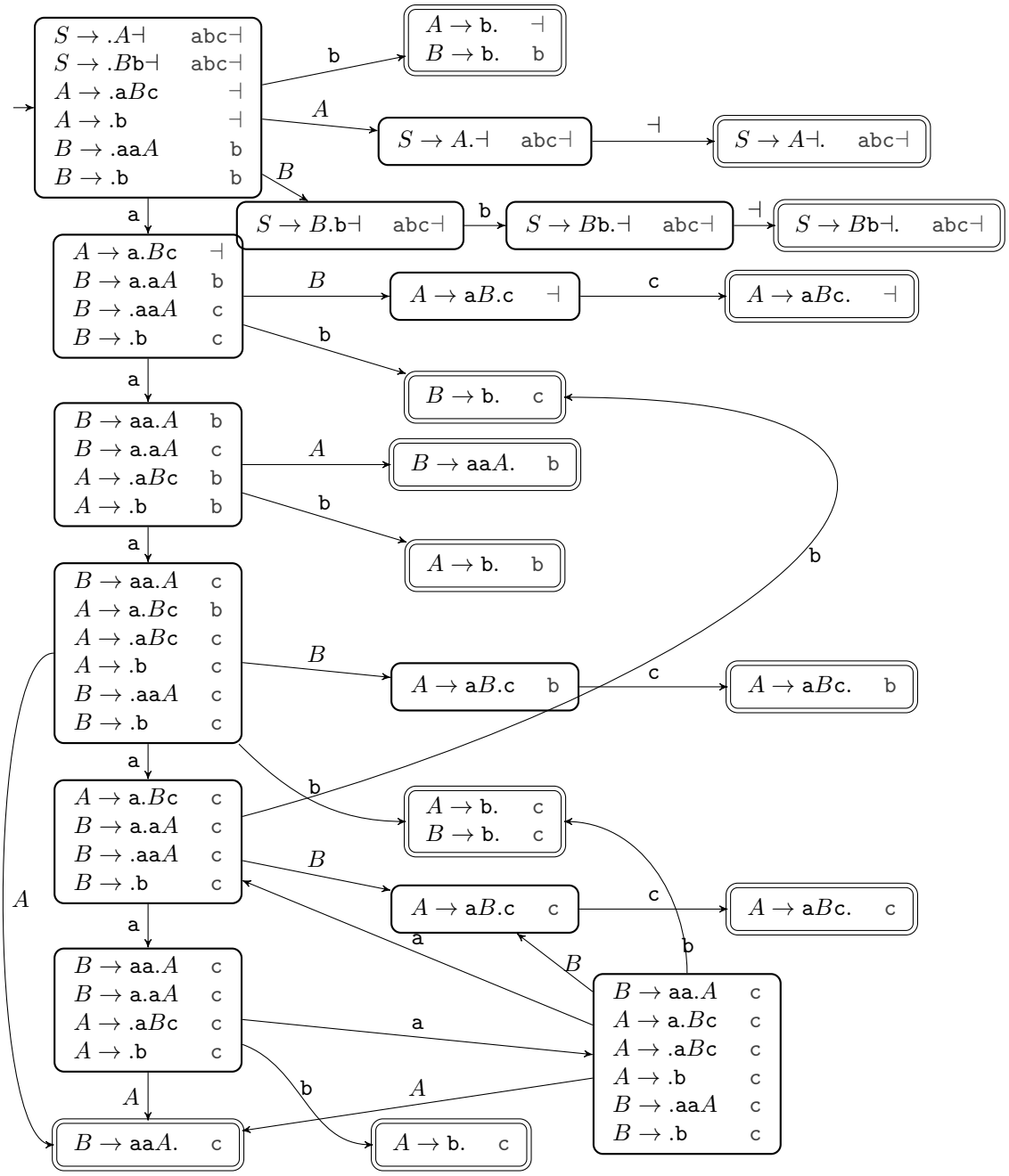


Poiché lo stato finale più in alto ha due regole completate consistenti, il DK_1-test non risulta soddisfatto, pertanto la grammatica analizzata non è LR(1).

Esercizio 6. Determinare se la seguente grammatica CFG con variabile iniziale S è LR(1):

$$\begin{array}{lcl} S \to & A\dashv & \mid \quad B\,\mathtt{b}\dashv \\ A \to & \mathtt{a}B\mathtt{c} & \mid \quad \mathtt{b} \\ B \to & \mathtt{aa}A & \mid \quad \mathtt{b} \end{array}$$

Soluzione: Per determinare se la grammatica è deterministica eseguiamo il DK_1-test:



Poiché lo stato finale al centro della figura ha due regole completate consistenti, il DK_1-test non risulta soddisfatto, pertanto la grammatica analizzata non è LR(1).

Esercizio 7. Si consideri il seguente linguaggio contenente codifiche di macchine di Turing con alfabeto di ingresso Σ:

> $L = \{\langle M \rangle \mid M$ è una DTM ed esiste $x \in \Sigma^*$ tale che $M(x)$ scrive un "blank" in una cella contenente un simbolo di $\Sigma\}$.

L è decidibile? Giustificare la risposta con una dimostrazione.

Soluzione: Osserviamo preliminarmente che non è possibile applicare il teorema di Rice: infatti la proprietà che caratterizza il linguaggio L, pur essendo non banale, non è realmente del linguaggio riconosciuto dalle macchine di Turing ma delle macchine di Turing stesse. Infatti, si consideri $M \in L$, dunque esista un input $x \in \Sigma^*$ tale che $M(x)$ scrive il simbolo "blank" in una cella contenente un simbolo di Σ. Consideriamo ora una DTM M' identica ad M tranne che M' include un nuovo simbolo di nastro "#". Inoltre ogni transizione di M in cui viene scritto il simbolo "blank" viene sostituita in M' da una analoga transizione in cui viene scritto il simbolo "#". Infine, per ogni transizione in cui viene letto il simbolo "blank" viene aggiunta una analoga transizione in cui viene letto il simbolo "#". È immediato ora verificare che $M' \notin L$ e che $L(M) = L(M')$.

L non è decidibile. Per dimostrarlo, supponiamo per assurdo che lo sia, e sia D una DTM che decide il linguaggio L. Consideriamo il problema indecidibile $\mathcal{A}_{\text{TM}}$: data una istanza $\langle M, w \rangle$, $\langle M, w \rangle \in \mathcal{A}_{\text{TM}}$ se e solo se $M(w)$ accetta. Si consideri ora la seguente DTM:

> P= " On input $\langle M, w \rangle$, where M is a TM and w is a string:
> 1. From $\langle M, w \rangle$, build the encoding of the following DTM R, which never writes a blank symbol over an input symbol except when explicitly stated:
>
> > R=" On any input x:
> > a. If $x \neq w$, then halt
> > b. Run $M(w)$
> > d. If $M(w)$ rejects, then halt
> > e. If $M(w)$ accepts, then:
> > f. Write an input symbol on tape
> > g. Write a blank symbol over the input symbol
> > h. Halt"

2. Run D on input $\langle R \rangle$
3. If $D(\langle R \rangle)$ accepts, then accept, else reject"

La DTM R utilizza un simbolo di "blank" differente da quello utilizzato per codificare il "blank" di M. Se $D(\langle R \rangle)$ accetta, allora $\langle R \rangle \in L$, e dunque esiste un input x tale che $R(x)$ scrive un "blank": necessariamente $x = w$ e $M(w)$ accetta. Se invece $D(\langle R \rangle)$ rifiuta, allora $\langle R \rangle \notin L$, dunque non esiste alcun input x per il quale $R(x)$ scrive un blank. In particolare, se $x = w$ allora $M(w)$ o rifiuta o non termina, quindi non accetta. La DTM P dunque decide $\mathcal{A}_{\text{TM}}$, ma questo è assurdo perché tale problema è indecidibile. Pertanto, resta dimostrato che L è indecidibile.

Esercizio 8. Si consideri il seguente linguaggio contenente codifiche di macchine di Turing:

$$L = \{\langle M \rangle \mid M \text{ accetta almeno 15 stringhe di ingresso}\}.$$

Dimostrare che L è ricorsivamente enumerabile (ossia Turing-riconoscibile) ma non decidibile.

Soluzione: Per dimostrare che L è ricorsivamente enumerabile è sufficiente esibire un riconoscitore per tale linguaggio:

R= " On input $\langle M \rangle$, where M is a TM with input alphabet Σ:
1. Make room for two counters i, n on the tape
2. `for` $i = 1$ `to` $+\infty$:
    3. Let $n \leftarrow 0$
    4. For each string s among the first i strings in Σ^*:
        5. Run M on s for at most i steps
        6. If in the previous step $M(s)$ accepted, then let $n \leftarrow n + 1$
        7. If $n \geq 15$, then accept

R utilizza la classica tecnica per evitare di entrare in un ciclo infinito durante la simulazione di una computazione $M(s)$ non terminante: per ciascun valore di i, R esegue i passi di M sulle prime i stringhe in string order di Σ^*. Se esistono almeno 15 stringhe accettate da M, consideriamo la quindicesima di esse in ordine lessicografico, e sia il suo indice I. Sia inoltre N il numero massimo di passi che M impiega per accettare ciascuna delle prime 15 stringhe. Allora nella iterazione con $i = \max\{I, N\}$ R simula l'esecuzione accettante di M su ciascuna delle 15 stringhe, e dunque il valore del contatore n raggiunge il valore 15. Pertanto $R(\langle M \rangle)$ accetta. Se invece M accetta meno di 15 stringhe, il valore di n non potrà mai raggiungere il valore 15, e quindi R non accetterà mai.

Per dimostrare che L non è decidibile, è sufficiente evidenziare che esso ricade sotto le ipotesi del teorema di Rice. Infatti la proprietà che caratterizza L può essere sintetizzata come $|L(M)| \geq 15$, dunque è una proprietà dei linguaggi riconosciuti dalle TM piuttosto che delle TM stesse; è inoltre evidente che la proprietà non è banale (ad esempio, non vale per $\emptyset$, ma vale per Σ^*). Dunque si applica il teorema di Rice, che stabilisce che L non è decidibile.

Esercizio 9. Si consideri il seguente linguaggio contenente codifiche di macchine di Turing:

$$L = \{\langle M \rangle \mid M \text{ accetta meno di 15 stringhe di ingresso}\}.$$

Dimostrare che L non è ricorsivamente enumerabile (ossia non è Turing-riconoscibile).

Soluzione: Per dimostrare che L non è ricorsivamente enumerabile è conveniente ragionare sulla decidibilità del suo linguaggio complemento: $L^c = A \cup B$, ove

$$\begin{aligned} A = &\quad \{\langle M \rangle \mid M \text{ accetta almeno 15 stringhe di ingresso}\} \\ B = &\quad \{X \mid \text{la stringa } X \text{ non codifica una TM}\}. \end{aligned}$$

Possiamo assumere che il linguaggio B sia decidibile, in quanto il controllo sulla correttezza formale della codifica di una TM è una operazione di routine in tutte le TM che presentiamo.

Possiamo dimostrare che il linguaggio A è ricorsivamente enumerabile ma non decidibile. Per questa dimostrazione si veda l'esercizio precedente.

Poiché A è ricorsivamente enumerabile, allora anche L^c lo è, in quanto l'unione di un linguaggio ricorsivamente enumerabile e di un linguaggio decidibile è ricorsivamente enumerabile. Supponiamo per assurdo che L sia ricorsivamente enumerabile; di conseguenza, L^c sarebbe decidibile. Poiché $A = L^c \setminus B$ (in quanto $A \cap B = \emptyset$), allora anche A sarebbe decidibile. La contraddizione prova che L non è ricorsivamente enumerabile.

Esercizio 10. Si consideri il seguente problema: data una lista di n insiemi S_1, S_2, $\ldots S_n$ ed un numero $k \in \mathbb{N}$, decidere se esiste una collezione di k insiemi dalla lista tali che nessun elemento è contenuto in due di questi k insiemi. Ad esempio, gli insiemi $\{1,3,5\}$, $\{1,2,3\}$, $\{2,4\}$, $\{2,5,7\}$, $\{6\}$ e $k = 3$ sono una "istanza-sì" in quanto gli insiemi $\{1,3,5\}$, $\{2,4\}$ e $\{6\}$ non hanno elementi in comune. Dimostrare che il linguaggio corrispondente a questo problema decisionale è NP-completo.

Soluzione: Il problema in questione, conosciuto come Set Packing, è verificabile in tempo polinomiale. Infatti, data una istanza con n insiemi S_i e da un intero $k > 0$, un certificato per

l'istanza è costituito da k insiemi $S_{i_1}, \ldots S_{i_k}$ tali che $S_{i_j} \cap S_{i_h} = \emptyset$ per ogni $1 \leq j < h \leq k$. Poiché il certificato include un sottoinsieme degli elementi della istanza, la sua dimensione non è superiore a quella della istanza. Inoltre una TM che riceve come input sul nastro una istanza del problema ed un certificato può facilmente controllare in tempo polinomiale che tutti gli insiemi del certificato siano insiemi della istanza e che nessuna coppia di insiemi del certificato abbia un elemento in comune. Pertanto SET PACKING appartiene alla classe NP.

Per dimostrare che SET PACKING è NP-hard, consideriamo una riduzione polinomiale dal problema NP-completo INDEPENDENT SET. Sia dunque (G, k) una istanza di INDEPENDENT SET, ove $G = (V, E)$ è un grafo con insieme di vertici V ed un insieme di archi E, e $k > 0$ è un intero. Fissiamo un ordinamento qualunque degli archi di G: $E = \{e_1, e_2, \ldots e_m\}$. Per ciascun nodo $v \in V$, includiamo nella collezione $\mathcal{S}$ della istanza di SET PACKING l'insieme di numeri $S_v = \{x \in \mathbb{N} \,|\, \text{l'arco } e_x \text{ incide su } v\}$. In altri termini, $\mathcal{S}$ contiene tanti insiemi quanti sono i nodi di G, e ciascuno di essi è costituito dagli indici degli archi incidenti sul nodo stesso. L'istanza ridotta di SET PACKING corrispondente a (G, k) è $(\mathcal{S}, k)$. È immediato verificare che è possibile costruire una istanza ridotta $(\mathcal{S}, k)$ in tempo polinomiale nella dimensione di (G, k).

Supponiamo che (G, k) sia una istanza-sì di INDEPENDENT SET: pertanto esiste un sottoinsieme di nodi $U \subseteq V$ di dimensione almeno k tale che non esiste alcun arco in G tra i nodi in U. Consideriamo quindi gli insiemi S_v in $\mathcal{S}$ tali che $v \in U$: questi sottoinsiemi rappresentano un certificato valido per SET PACKING. Se infatti vi fosse un elemento x in comune tra due sottoinsiemi S_u e S_v nel certificato, allora tra i nodi u e v di G esisterebbe un arco e_x, ma questo è impossibile perché $u, v \in U$, che è una soluzione di INDEPENDENT SET. Viceversa, dato un certificato che testimonia sull'esistenza di una soluzione per l'istanza ridotta di SET PACKING, l'insieme di nodi di G corrispondenti agli insiemi nel certificato costituiscono necessariamente un insieme indipendente di dimensione k per G. Concludiamo pertanto che SET PACKING è NP-hard, e quindi NP-completo.

Esercizio 11. Si consideri il seguente problema: data una lista (multinsieme) di numeri positivi S, determinare se esiste una partizione di S in due multinsiemi A e B ($A \cup B = S$, $A \cap B = \emptyset$) tali che la somma degli elementi di A coincide con la somma degli elementi di B. Dimostrare che il linguaggio corrispondente è NP-completo.

Soluzione: Questo problema è noto, nella sua forma più generale in cui i numeri sono interi sia positivi che negativi, con il nome di PARTITION.

È facile dimostrare che PARTITION è verificabile in tempo polinomiale. Infatti data una istanza S consistente in una collezione di numeri interi (indifferentemente positivi o negativi), un

certificato consiste in un singolo multinsieme A. Il verificatore in tempo polinomiale controlla che il multinsieme A è contenuto in S (ossia che tutti gli elementi in A sono anche elementi di S), e che la somma degli elementi di A coincide con la somma degli elementi di S che non sono in A. Poiché la dimensione del certificato non è superiore alla dimensione dell'istanza ed il verificatore esegue semplici somme di numeri interi, si conclude che PARTITION (e quindi anche la sua restrizione agli interi positivi) è in NP.

Per dimostrare che PARTITION è NP-hard mostriamo una riduzione polinomiale dal problema NP-completo SUBSET SUM. Sia data dunque una istanza (S, t) di SUBSET SUM, ove S è un multinsieme di interi e t è un intero. Sia $\mu = \sum_{x \in S} x$ la somma di tutti gli elementi in S. L'istanza ridotta di PARTITION consiste nello stesso multinsieme S con un elemento y in più, precisamente il valore intero $y = 2\,t - \mu$: $P = S \cup \{y\}$. La somma di tutti gli elementi di P è $\mu + (2\,t - \mu) = 2\,t$. È evidente che la funzione che trasforma una istanza di SUBSET SUM in una istanza di PARTITION è calcolabile in tempo polinomiale.

Supponiamo che (S, t) sia una istanza-sì di SUBSET SUM, e quindi che esiste un sottoinsieme T degli elementi di S la cui somma è esattamente t: $\sum_{x \in T} x = t$. Si consideri allora la partizione di P costituita da T da una parte, e da $S \setminus T \cup \{y\}$ dall'altra. La somma degli elementi del secondo multinsieme è $\mu - t + (2\,t - \mu) = t$. Quindi P è una istanza-sì di PARTITION.

Supponiamo ora che P sia una istanza-sì di PARTITION. Pertanto esistono due sotto-multinsiemi A e B tali che $A \cup B = P$ e $A \cap B = \emptyset$, e tali che $\sum_{x \in A} x = \sum_{x \in B} x = t$. L'elemento y deve appartenere ad A oppure a B; senza perdita di generalità supponiamo sia in A. Pertanto B è costituito da elementi di S la cui somma è esattamente t. Quindi (S, t) è una istanza-sì di SUBSET SUM.

La trasformazione mostrata è dunque una riduzione polinomiale, quindi PARTITION è NP-hard, e quindi NP-completo.

Esercizio 12. Si consideri il seguente problema decisionale: dato un insieme $X \subset \mathbb{N}^2$ di coppie (v, w) e due numeri $C, W \in \mathbb{N}$, esiste un sottoinsieme $S \subseteq X = \{(v_1, w_1), \ldots (v_m, w_m)\}$ tale che $\sum_{i=1}^{m} v_i \geq C$ e $\sum_{i=1}^{m} w_i \leq W$? Dimostrare che il linguaggio corrispondente è NP-completo.

Soluzione: Questo problema è noto con il nome KNAPSACK ("zaino" o "bisaccia"): infatti può essere considerato come l'equivalente del problema di riempire un contenitore avente una capacità finita W in modo che il valore degli oggetti inseriti sia il maggiore possibile (nella versione decisionale, sia maggiore di una soglia C).

Il problema KNAPSACK è verificabile in tempo polinomiale. Infatti un certificato per una istanza (X, C, W) è costituita da un insieme $S \subset \mathbb{N}^2$ con non più di $|X|$ coppie. Il verificatore controlla che tutte le coppie nel certificato siano incluse in X, che siano tutte distinte, che la somma dei primi elementi delle coppie sia non inferiore a C, e che la somma dei secondi elementi delle coppie sia non superiore a W. Considerata che la dimensione del certificato è inferiore alla dimensione dell'istanza e che il verificatore effetta semplici somme e confronti di interi, KNAPSACK è in NP.

Consideriamo una riduzione dal problema SUBSET SUM: sia (S, t) una istanza costituita da un multinsieme di numeri interi $S = (s_1, s_2, \ldots, s_n)$ ed un numero "target" t. La corrispondente istanza ridotta di KNAPSACK è semplicemente (K, t, t), ove $K = ((s_1, s_1), (s_2, s_2), \ldots, (s_n, s_n))$ (quindi ogni intero S è trasformato in una coppia avente lo stesso intero come primo e come secondo elemento).

Sia (S, t) una istanza-sì di SUBSET SUM: dunque esiste un sotto-multinsieme T di S tale che $\sum_{x \in T} x = t$. Perciò il corrispondente sotto-multinsieme di coppie $L = ((x, x) \,|\, x \in T)$ è tale che la somma dei primi elementi è t $(= C)$ e la somma dei secondi elementi è t $(= W)$. L'esistenza di L implica che (K, t, t) è una istanza-sì di KNAPSACK.

Sia invece (K, t, t) una istanza-sì di KNAPSACK; dunque esiste un sotto-multinsieme L di K tale che $\sum_{(x,x) \in L} x \geq t$ e $\sum_{(x,x) \in L} x \leq t$. Perciò $\sum_{(x,x) \in L} x = t$. Se dunque consideriamo $T = (x \,|\, (x, x) \in L)$, la somma degli elementi del sotto-multinsieme T di S ha somma esattamente t. Dunque (S, t) è una istanza-sì di SUBSET SUM.

L'esistenza della riduzione polinomiale da SUBSET SUM a KNAPSACK dimostra che quest'ultimo è NP-hard, e di conseguenza NP-completo.