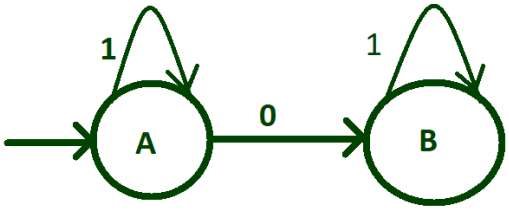
01 – Progettare DFA

1. Costruite un DFA che riconosce il linguaggio

$$L_1 = \{w \in \{0,1\}^* \mid w \text{ contiene un numero di 0 multiplo di 3}\}$$

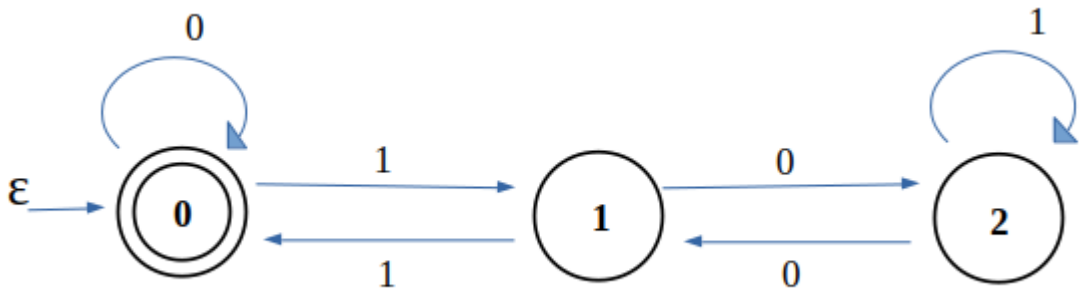
Per esempio, 000, 00110 e 010101010101 appartengono al linguaggio perché contengono rispettivamente 3, 3 e 6 zeri, mentre 00, 001010 e 0101010101, che contengono 2, 4 e 5 zeri, non appartengono al linguaggio.



2. Costruite un DFA che riconosce il linguaggio

$$L_2 = \{w \in \{0,1\}^* \mid w \text{ è la codifica binaria di un mumero multiplo di 3}\}$$

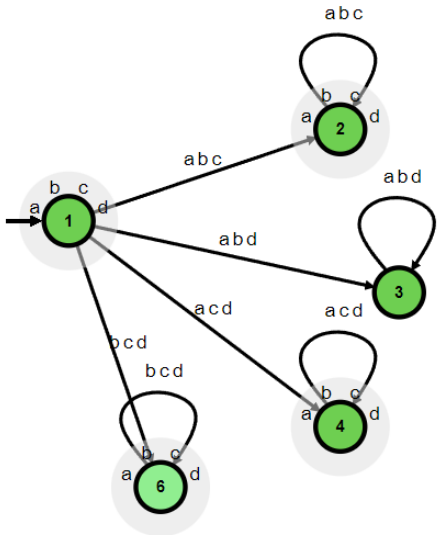
Per esempio, 11, 110 e 1001 appartengono al linguaggio perché sono le codifiche binarie di 3, 6 e 9, mentre 10, 111 e 1011 non appartengono al linguaggio perché sono le codifiche binarie di 2, 7 e 11. La stringa vuota non codifica nessun numero.



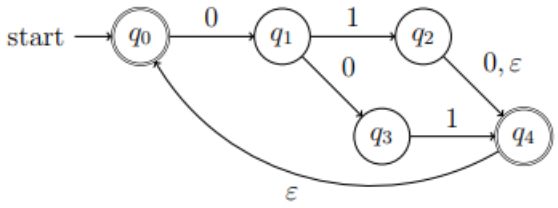
02 – NFA/epsilon-NFA

1. (a) Considera l'alfabeto $\Sigma = \{a, b, c\}$ e costruisci un automa non deterministico che riconosce il linguaggio di tutte le parole tali che uno dei simboli dell'alfabeto *non compare mai*:
   - tutte le parole che non contengono a;
   - + tutte le parole che non contengono b;
   - + tutte le parole che non contengono c.

   (b) Trasforma l'NFA in DFA usando la costruzione per sottoinsiemi.



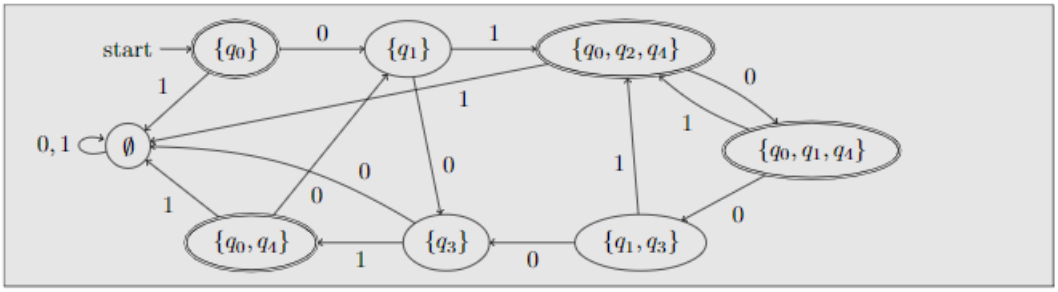
2. Dato il seguente ε-NFA:



(a) costruisci la ε-chiusura di tutti gli stati dell'automa

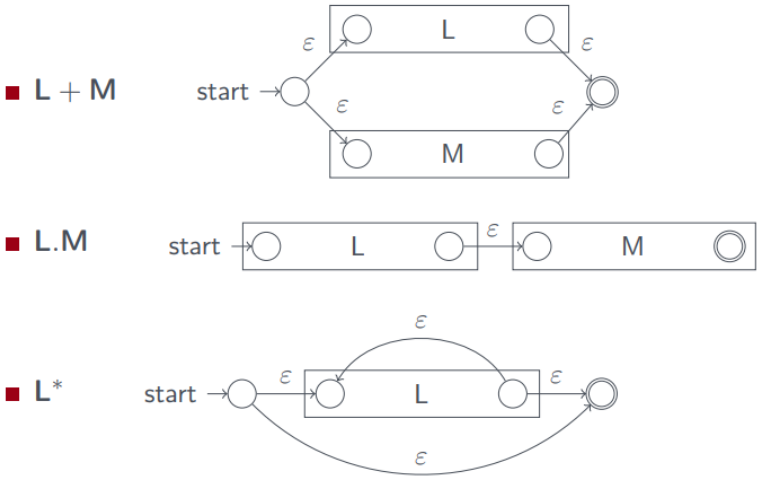
(b) trasforma l'automa in DFA usando la costruzione per sottoinsiemi

La epsilon chiusura individua q2,q4,q0



03 – esercizi

1) Sì a tutte e 3 le domande
2) Intuitivamente (poi seguono gli automi di riferimento):



3) per l'unione basta avere uno stato iniziale comune ed una biforcazione verso due stati
4) per la concatenazione si avrà uno stato iniziale seguito da uno stato finale oppure uno non finale
5) per lo star, basta avere tutte le combinazioni da e verso altri stati

Gioco.pdf

L = {w ∈{a, b}∗ : il numero di a è uguale al numero di b}
Il linguaggio è regolare?

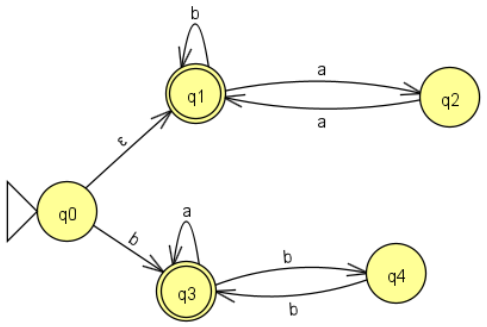
Supponiamo per assurdo sia regolare. Dunque, data k la lunghezza pumping, consideriamo la parola a^kb^k, su cui scegliamo una suddivisione di w tale che:
y ≠ ε e |xy| ≤ k:
Prendiamo ad esempio:
w= aaaa a…..a a..aa..b…bb
x y z
Si sa infatti che $xy^iz \in L(A)$, ma possiamo notare che il numero di a diventa più grande di quello di b per esempio con $xy^2z \in L_{ab}$ e quindi non è più verificata la seconda delle due condizioni dette prima.
Morale della favola: non è regolare.

L = {w ∈{a, b}∗ : il numero di a è maggiore del numero di b}
Il linguaggio è regolare?

Ripetendo lo stesso discorso di prima, valenti le tre condizioni, supponiamo l'espressione lo sia e consideriamo come parola:
$w=a^{(k+1)}b^k$
Prendendo come prima una suddivisione generica della parola w,
tipo xy^0z, avremo che il numero di a è invece inferiore a quello di b.
Morale della favola: non è regolare.

L = {w ∈{a, b}∗: numero di a è pari, il numero di b è dispari}
Il linguaggio è regolare?

Supponiamo per assurdo non sia regolare e consideriamo come parola:
$w= a^{(2k)}b^{2k+1}$
Si nota però che è possibile costruire un automa equivalente del tipo:



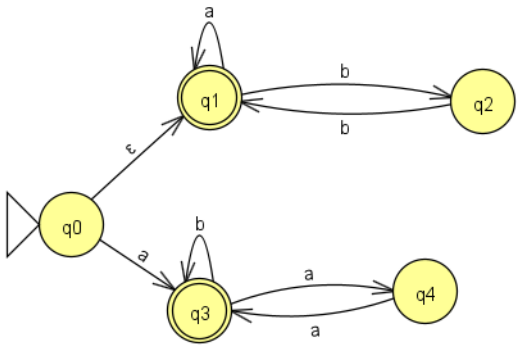
Inoltre, si può rappresentare regolarmente con l'espressione:
(a(a+b)*a(a+b)*b(a+b)*)*
Morale della favola: l'espressione è regolare

L_{2ab} = {w ∈{a, b}∗ : numero di a è due volte il numero di b}
Il linguaggio è regolare?

Supponiamo per assurdo che lo sia e, data una generica *h* lunghezza del PL,
consideriamo la parola *w* = $a^{2k}b^{k}$
Considerando poi un generico split:
w= aa…aa a..a aa…. b
x y z
Si nota che il numero di a è maggiore rispetto a quello di b e le stringhe x ed y sono fatte di sole "a"
Morale della favola: non è regolare

L = {w ∈{a, b}∗: numero di a è dispari, il numero di b è pari}
Il linguaggio è regolare?

Supponiamo per assurdo che lo sia e consideriamo come parola:
w= $a^{(2k+1)}b^{2k}$
Il caso è paritetico a quello precedente, ma con i termini girati e:



L'ER che può rappresentarlo è:
(a(a+b)*b(a+b)*b(a+b)*)*
Morale della favola: l'espressione è regolare

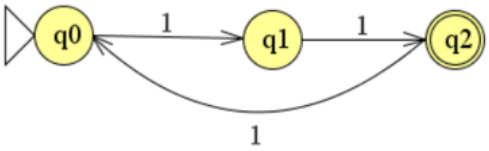
04– gioco2.pdf

Il linguaggio $L = \{1^{3n+2} : n \geq 0\}$ *è regolare?*

- se pensi che il linguaggio **sia regolare**, gira il foglio
- se pensi che il linguaggio **non sia regolare**, riempi lo schema di soluzione qui sotto

Sì, L_{3n+2} è regolare:

- è rappresentato dall'espressione regolare $(111)^*11$
- e riconosciuto dall'automa



Esercizio 6

Il linguaggio $L = \{0^n 1^m 0^n : m + n > 0\}$ *è regolare?*

- se pensi che il linguaggio **sia regolare**, riempi lo schema di soluzione qui sotto
- se pensi che il linguaggio **non sia regolare**, gira il foglio

Sì, il linguaggio è regolare perché è riconosciuto dall'automa a stati finiti

Ripetendo lo stesso discorso di prima, valenti le tre condizioni, supponiamo l'espressione lo sia e consideriamo come parola:
$w=0^{2n}1^{3m}$ (diventa 2n per proprietà alle potenze, somma di n+n)
Prendendo come prima un qualsiasi split w=xyz, y ≠ε, |xy|<=k
Mettendo per esempio i = 3 nella formula xy^iz
avremo che il numero di 0 sarà sempre maggiore di quello di 1 e non è regolare.
Morale della favola: non è regolare

Esercizio 7

Il linguaggio $L = \{0^n 1^m 0^p : m + n + p > 0\}$ *è regolare?*

- se pensi che il linguaggio **sia regolare**, riempi lo schema di soluzione qui sotto
- se pensi che il linguaggio **non sia regolare**, gira il foglio

Supponiamo per assurdo sia regolare e diciamo che, data h la lunghezza del P.L. consideriamo la parola:
$w = 0^{n+p}1^m$
Prendendo come al solito un qualsiasi split w=xyz , y ≠ε, |xy|<=k
Mettendo per esempio i = 3 nella formula xy^iz
Con un qualsiasi i, vedendo che nella xyz ci sarebbe di sicuro una possibile intersezione,
il numero di 0 sarà sempre maggiore di quello degli 1.
Morale della favola: non è regolare

Esercizio 8

Il linguaggio $L = \{w \in \{a,b\}^* : \text{numero di } a \text{ è due volte il numero di } b\}$ *è regolare?*

- se pensi che il linguaggio **sia regolare**, riempi lo schema di soluzione qui sotto
- se pensi che il linguaggio **non sia regolare**, gira il foglio

Supponiamo per assurdo sia regolare e diciamo che, data h la lunghezza del P.L. consideriamo la parola:
$w = a^{2n}b^n$
Prendendo come al solito un qualsiasi split w=xyz , y ≠ε, |xy|<=k
Si può osservare che non si rispetta la condizione 3 (|xy|<=k)
in quanto il numero di a, dati gli esponenti di prima, con un qualsiasi i dentro xy^iz
sarebbe maggiore del numero di "b"
Morale della favola: non è regolare

Grammatiche context-free - 05-esercizi.pdf

Esercizio 1

Definire una grammatica per il seguente linguaggio:

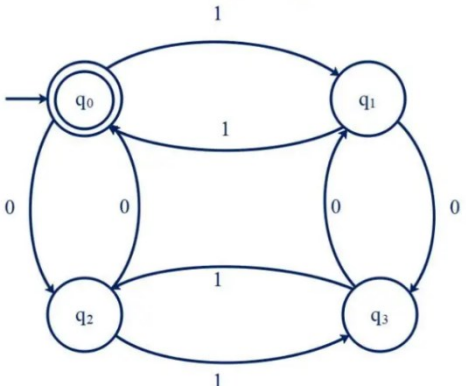
$$L_1 = \{a^i b^j c^k \mid i = j \text{ oppure } j = k\}$$

In pratica si ragiona che, avendo "a" e "b" che si incrementano per lo stesso indice, avendo poi c, di fatto l'idea è di garantire l'alternanza tra "a" e "b", poi unendo l'altro caso dell'oppure, dove si ha lo stesso discorso ma tra "b" e "c".

Per il caso i=j
$S_1 \rightarrow aS_2bS_3 \mid \varepsilon$
$S_2 \rightarrow aS_2b \mid \varepsilon$
$S_3 \rightarrow bS_3c \mid \varepsilon$

Per il caso j=k
$S_1 \rightarrow bS_2cS_3 \mid \varepsilon$
$S_2 \rightarrow cS_2 \mid \varepsilon$
$S_3 \rightarrow bS_3c \mid \varepsilon$

Automa corrispondente:



Oppure:

Answer: $G = (V, \Sigma, R, S)$ with set of variables $V = \{S, W, X, Y, Z\}$, where S is the start variable; set of terminals $\Sigma = \{a, b, c\}$; and rules

$$\begin{aligned} S &\rightarrow XY \mid W \\ X &\rightarrow aXb \mid \varepsilon \\ Y &\rightarrow cY \mid \varepsilon \\ W &\rightarrow aWc \mid Z \\ Z &\rightarrow bZ \mid \varepsilon \end{aligned}$$

Esercizio 2

Definire una grammatica per il seguente linguaggio:

$L_1 = \{w \in \{0,1\}^* \mid w \text{ contiene un numero pari di 0 e un numero dispari di 1}\}$

$S_1 \rightarrow 1S_1 \mid S_1 1 \mid \varepsilon$
$S_2 \rightarrow 0S_2 \mid S_2 0$
$S_3 \rightarrow 1S_3 \mid S_3 1$
$S_4 \rightarrow 0S_4 \mid S_4 0 \mid \varepsilon$

Esercizio 3

Definire una grammatica per il linguaggio di tutte le espressioni regolari sull'alfabeto $\{0, 1\}$, cioè di tutte le espressioni costruite utilizzando

- le costanti di base: $\varepsilon, \emptyset, 0, 1$
- gli operatori: $+, \cdot, *$
- le parentesi

FACTOR → (EXPR) | Ø | ε | 0 | 1
(EXPR) → <FACTOR>
SUM → SUM + SUM | PRODUCT
PRODUCT → PRODUCT * PRODUCT | STAR
STAR → FACTOR* | FACTOR

04 – Pumping Lemma – Esercizi

Supponiamo per assurdo sia regolare. Dunque, data k la lunghezza pumping, consideriamo la parola a^kb^k, su cui scegliamo una suddivisione di w tale che:
y ≠ ε e |xy| ≤ k:
Prendiamo ad esempio:
w= aaaa a…..a a..aa..b…bb
x y z
Si sa infatti che $xy^iz \in L(A)$, ma possiamo notare che il numero di a diventa più grande di quello di b per esempio con $xy^2z \in L_{ab}$ e quindi non è più verificata la seconda delle due condizioni dette prima.
Morale della favola: non è regolare.

06b-esercizi – Grammatiche context-free

Considera l'alfabeto $\Sigma = \{0, 1\}$, e sia L_3 l'insieme di tutte le stringhe che contengono almeno un 1 nella loro seconda metà. Più precisamente, $L_3 = \{uv \mid u \in \Sigma^*, v \in \Sigma^*1\Sigma^* \text{ e } |u| \geq |v|\}$. Definisci una CFG che genera L_3.

$A \to 0A0 \mid 0B1 \mid 1A0 \mid 1B1$
$B \to 0B0 \mid 0B1 \mid 1B0 \mid 1B1 \mid 0 \mid 1 \mid \varepsilon$

Esercizio 4

Definire le grammatiche context-free che generano i seguenti linguaggi. Salvo quando specificato diversamente, l'alfabeto è $\Sigma = \{0, 1\}$.

1. $\{w \mid w$ contiene almeno tre simboli uguali a $1\}$
2. $\{w \mid$ la lunghezza di w è dispari$\}$
3. $\{w \mid w = w^R$, cioè w è palindroma$\}$
4. $\{w \mid w$ contiene un numero maggiore di 0 che di $1\}$
5. Il complemento di $\{0^n1^n \mid n \geq 0\}$
6. Sull'alfabeto $\Sigma = \{0, 1, \#\}$, $\{w\#x \mid w^R$ è una sottostringa di x e $w, x \in \{0, 1\}^*\}$

1) S → X1X1X1X
X → 0X | 1X | ε

2) S ⇒ 0S0 | 1S1| 0S1 | 1S0 |ε

3) S → 0S0
S → 1S1
S → ε
S → 0
S → 1

4) S → TS|0T|0S
T → TT|1T0|0T1|ε

5)
S→1A|A0|0S1
A→0A|1A|ϵ

6)

$$
\begin{array}{lcll}
S & \to & CB & \triangleright \text{ Generates } w\#\Sigma^* w^R \Sigma^*. \\
C & \to & 0C0 \mid 1C1 \mid \#B & \triangleright \text{ Generates } w\#\Sigma^* w^R. \\
B & \to & AB \mid \varepsilon & \triangleright \text{ Generates } A^* = \Sigma^*. \\
A & \to & 0 \mid 1 & \triangleright \text{ Generates } \Sigma.
\end{array}
$$

06 – Forme Normali Esercizi

Esercizio 1

Trasformare la seguente CFG in forma normale di Chomsky, usando l'algoritmo mostrato nelle slide:

$$A \to BAB \mid B \mid \varepsilon$$
$$B \to 00 \mid \varepsilon$$

Aggiungiamo la nuova variabile iniziale:
S_0 → A
A → BAB|B|ε
B → 00|ε

Rimuoviamo le ε-regole, quindi B → ε ed A → ε, partendo proprio da B → ε:
S_0 → A
A → AB|BA|B|BAB|ε
B → 00

e poi A → ε (che più di tanto non ha effetto).
S_0 → A
A → AB|BA|B|BAB|A
B → 00

Ora vogliamo togliere le regole unitarie, quindi A→B, A→A, S_0→A,
partendo da A→A che semplicemente:
S_0 → A
A → BAB|B|AB|BA
B → 00

poi vogliamo togliere A→B (sostituendo B con 00):

S_0 → A
A → BAB|00|AB|BA
B → 00

quindi togliamo S_0→A (sostituendo BAB|00|AB|BA ad A):
S_0 → BAB|00|AB|BA
A → BAB|00|AB|BA
B → 00

Seguendo troviamo le produzioni che hanno più di 2 variabili a destra:
S_0 → BAB A → BAB
introducendo una variabile generica X che sostituisce ad esempio BA:
S_0 → XB|00|AB|BA
A → XB|00|AB|BA
B → 00
X → BA

Notiamo inoltre 00 come simbolo terminale e gli andrà aggiunta una regola:
S_0 → XB|Y|AB|BA
A → XB|Y|AB|BA
B → 00
X → BA
Y → 00

Esercizio 2

Per ogni linguaggio L, sia

$$\text{suffix}(L) = \{v \mid uv \in L \text{ per qualche stringa } u\}.$$

Dimostra che se L è un linguaggio context-free, allora anche suffix(L) è un linguaggio context-free.

Suggerimento: puoi assumere che L sia generato da una grammatica in Forma Normale di Chomski.

Assumendo che il linguaggio sia definito secondo la seguente grammatica:
S → u
u → v | uv

dobbiamo ricondurci ad avere una cosa del tipo:
A → BC
A → a

A questo punto si nota che c'è una forma unitaria del tipo u → v, ma anche S → u. Una eventuale sostituzione del primo caso porterebbe a:
S → v | uv
u → v | uv

e dunque meglio tenersi:
S → u
u → v | uv

Meglio quindi introdurre una nuova regola, essendo "u" stato terminale, tale da sostituirlo:

$S \rightarrow X$
$u \rightarrow v \mid uv$
$X \rightarrow u$

Tale linguaggio rientra nella forma normale definita da Chomsky.
A tale scopo si vede che la stringa è uniformemente ripartita tra u e v, nelle tre parti possibili.
Possiamo ipotizzare seguendo il PL una stringa del tipo $w=u^pv^p$ vedendo come è scritto il linguaggio.
Si nota subito che avendo ad esempio: $x=\varepsilon$, $y=u^k$, $z=v^j$ per qualche j, $k > 0$ si avrà una presenza normale di 0 ed 1 ed equamente distribuita. Inoltre per la stringa di u^k sarà sempre previsto un numero di 0 che rientra nella condizione $|xy| <= k$, quindi del tipo $w=u^pv^{k-p}$, tale che qualsiasi pumping rientri nel linguaggio.
Essendo regolare, il linguaggio è context-free.

16-Riducibilità-Esercizi

1) Supponiamo che ALL_{TM} sia decidibile e che esista una TM M che decida questo linguaggio. A questo applicheremo una funzione di riduzione. Pertanto, decidiamo di strutturare:
   M = Su input w, dove w è una stringa:
   - se la stringa appartiene al linguaggio (quindi a Σ^*) *accetta*
   - altrimenti *rifiuta*, che significa che $L(M) = \emptyset$

   Per poter dimostrare che è indecidibile, abbiamo ora costruito un decisore. Dato che deve essere indecidibile, ora mostriamo una macchina A_{TM} che accetta l'opposto; se ciò avviene, come capita, il linguaggio è indecidibile. Quindi per decidere A_{TM}:
   M' = Su input <M, w>, dove w è una stringa ed M è una TM:
   - esegue R su input <M'>
   - se R accetta, allora *accetta*, altrimenti *rifiuta*

   Dato che A_{TM} è indecidibile, R non può esistere e dunque ALL_{TM} è indecidibile.
2) Supponiamo che X sia decidibile e che esista una TM M che decida questo linguaggio. A questo applicheremo una funzione di riduzione. Pertanto, decidiamo di strutturare:
   M = Su input w, dove w è una stringa:
   - comincio a scorrere il nastro
   - se trovo *w*, lo marco e torno all'inizio del nastro
   - una volta esaurito tutto l'input, *accetta* se *w* è stato marcato
   - altrimenti *rifiuta*

   Abbiamo costruito un decisore; ora andiamo ad applicare la riduzione, usando il decisore appena creato per costruire una TM N che decide A_{TM}.
   Questa macchina N, su input <M, w>, dove M è una TM e w è una stringa:
   - simula <M, w> sul nastro
   - marca la fine destra dell'input con un simbolo
   - lo copia sul nastro dopo il blank
   - si simula M su w'
     - se M accetta, si scrive qualsiasi carattere sulla prima parte del nastro ed *accetta*
     - altrimenti *rifiuta*

   Dando in output <M, w>, avremo che l'ultima macchina accetta l'esatto opposto di quanto voluto; abbiamo dimostrato che esiste un decisore tale da accettare non solo la stringa w ma anche tutte le altre stringhe e non modifica mai l'input; dunque A_{TM} è deciso, ma è una contraddizione sulla base di quanto detto. A_{TM} può quindi dirsi indecidibile.

3) Se $A \leq_m B$ e B è un linguaggio regolare, dobbiamo verificare se A possa essere anch'esso un linguaggio regolare. Però semplicemente si potrebbe avere il caso in cui B comprende già tutte le stringhe di A ed A sia non regolare.
   A tale scopo facciamo l'esempio di avere $A = \{0^n1^n \mid n >=0\}$ e $B = \{1\}$. Dato che sono entrambi decidibili, si ha semplicemente la costruzione per entrambi di una TM che simula sul nastro sulla

stringa w ed accetta qualora riconosca esattamente il linguaggio posto da A e da B. Notiamo inoltre che B riconosce tutte le stringhe di A e B è regolare; A invece non lo è.
Si dimostra quindi che B risolve tutti i problemi di A, ma ciò non implica che A sia necessariamente regolare.

4) Essendo A un linguaggio decidibile, anche A_{TM} lo è e similmente E_{TM} è indecidibile. Dobbiamo quindi dimostrare che non esiste una funzione che passando per il test del vuoto decide il linguaggio di A.
Verificando il funzionamento di A, intesa con una TM M:
M = Su input w, dove w è una stringa:
    - se w appartiene al linguaggio, quindi a Σ*, accetta
    - se w non appartiene al linguaggio, *rifiuta*

Convertiamo quindi ad una funzione di riduzione f che ragiona in questo modo:
M' = Su input <M, w> dove M è una TM e w è una stringa:
    - prendendo in input la macchina precedente, dovremmo accettarne almeno una stringa e quando M accetta, *accetta*
    - altrimenti *rifiuta*

In output riceveremmo una macchina decisore M_1 che decide se esiste almeno una stringa, rispetto alla macchina precedente. Siccome E_{TM} misura il test del vuoto, ciò significa che il linguaggio deve essere vuoto e deve esistere una funzione di riduzione che accetta quando il linguaggio sia vuoto; noi tuttavia abbiamo dimostrato che il linguaggio non può essere vuoto, ma accetterebbe come visto a lezione $\underline{E_{TM}}$. Siccome la decidibilità non è affetta dal complemento, non esiste la riduzione e il linguaggio è indecidibile.

5) Qui dobbiamo mostrare due cose:
- A è Turing-riconoscibile
- $A \leq_m \underline{A}$

Partiamo col mostrare che A è ridotto da $\underline{A}$; questo perché, se ciò accade, $\underline{A}$ è Turing-riconoscibile, allora anche A lo è. Essendo che sia il complementare che il normale linguaggio sono Turing-riconoscibili, allora A è decidibile.
Verificando il funzionamento di A, intesa con una TM M:
M = Su input w, dove w è una stringa:
    - Eseguiamo M su x. Se tale stringa appartiene ad A, *accetta*
    - Se tale stringa non appartiene ad A, *rifiuta*

Abbiamo quindi un decisore per A; vogliamo quindi dimostrare che anche $\underline{A}$ ha un decisore e che quindi accetti le stringhe opposte.
Come tale, costruiamo una TM W su input <M, w>:
    - esegue <M, w> sul nastro
    - se trova un input $\underline{w}$, *accetta*
    - altrimenti *rifiuta*

Ora siccome evidentemente:

$$n \in A \iff f(n) \in A'$$

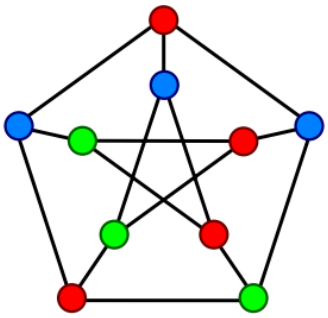
equivalentemente:

$$n \notin A \iff f(n) \notin A'$$

Avendo dimostrato l'esistenza dei decisori per il complemento e anche per le stringhe normali, vediamo che se esiste una stringa appartenente ad A, equivalentemente la stringa non viene accettata da $\underline{A}$ e vale anche il contrario. Pertanto per ogni n, A è decidibile.

19 Classe Np-Esercizi

"Colorare" i vertici di un grafo significa assegnare etichette, tradizionalmente chiamate "colori", ai vertici del grafo in modo tale che nessuna coppia di vertici collegati da un arco condivida lo stesso colore. La figura seguente mostra un esempio di colorazione di un grafo con 10 vertici che usa 3 colori (rosso, verde, blu).



Chiamiamo 3-COLOR il problema di trovare, se esiste, una colorazione di un grafo che usa 3 colori diversi. Dimostrate che 3-Color è un problema in NP nel modo seguente:

1) *definite com'è fatto un certificato per 3-COLOR*
2) *definite un verificatore polinomiale per 3-COLOR*

1) L'idea della dimostrazione è di avere un'informazione che capisca se i nodi vicini siano connessi e siano di un colore diverso rispetto al nodo attuale; tanto basterebbe, dato che letteralmente ad ogni nodo sono connessi almeno tre nodi e ciascuno di questi ha un colore diverso rispetto al nodo attuale (tra loro possono anche avere valore uguale).
   Ciò rappresenta il concetto di certificato per questo particolare problema.
2) Consideriamo poi il verificatore V
   V = Su input <G, c>
   1) Controlla se il nodo attuale appartiene al vettore dei nodi, tale che sia connesso ai vicini
   2) Controlla se G contiene tutti i nodi di C
   3) Se il nodo appartiene a G ed è di diverso colore, allora continua ciclicamente a testare
   4) Se tutti accettano, *accetta*, altrimenti *rifiuta*

Considerate il seguente problema, che chiameremo SubsetSum: dato un insieme di numeri interi S ed un valore obiettivo t, stabilire se esiste un sottoinsieme S' ⊆ S tale che la somma dei numeri in S' è uguale a t. Esempio: se S = {4, 11, 16, 21, 27} e t = 25, allora il sottoinsieme S' = {4, 21} `e una soluzione di SubsetSum perché 4 + 21 = 25.
Dimostrate che SubsetSum è un problema in NP nel modo seguente:

1) *definite com'è fatto un certificato per SubsetSum*
2) *definite un verificatore polinomiale per SubsetSum*

1) Il certificato è il sottoinsieme stesso, dato che abbiamo un insieme di numeri e dobbiamo poi capire se appartengono all'insieme dei numeri completo.
2) L'idea di verificatore in tempo polinomiale è la seguente per V:
   V = Su input <<S, t>, c>:
   a. Controlla se c è una collezione di numeri la cui somma è t
   b. Controlla se la somma S contenga tutti i numeri
   c. Se entrambi passano, *accetta*, altrimenti *rifiuta*

20-esercizi NP Completi

Esercizio 1

Un circuito Hamiltoniano in un grafo G è un ciclo che attraversa ogni vertice di G esattamente una volta. Stabilire se un grafo contiene un circuito Hamiltoniano è un problema NP-completo.

Un *circuito quasi Hamiltoniano* in un grafo G è un ciclo che attraversa esattamente una volta tutti i vertici del grafo *tranne uno*. Il *problema del circuito quasi Hamiltoniano* è il problema di stabilire se un grafo contiene un circuito quasi Hamiltoniano.

- Dimostra che il problema del circuito quasi Hamiltoniano è in NP
- Dimostra che il problema del circuito quasi Hamiltoniano è NP-hard, usando il problema del circuito Hamiltoniano come problema di riferimento.

1) Abbiamo dimostrato in precedenza che il circuito Hamiltoniano è in NP. Dato che in uno quasi-Hamiltoniano è la stessa cosa meno un vertice, riscriviamo estensivamente la dimostrazione articolando:
N = Su input <G, s, t> dove G è un grafo quasi-hamiltoniano, "s" è un insieme di nodi e t è il certificato:

- prendiamo una lista di numeri, che rappresenta i nodi del grafo
- si controlla per le ripetizioni nella lista, se ne viene trovata una si rifiuta
- si considera che s parta dal primo numero p_1 e t vada fino a $p_m - 1$, dato che non considero l'ultimo vertice nell'attraversamento degli archi
- per ciascun arco tra 1 ed m – 2 si controlla se $<p_i, p_{i+1}>$ induttivamente sia un arco di G. Se ciò accade, tutti i test sono passati

Il ciclo che si genera non include nessun nuovo vertice, infatti ognuno avrà almeno grado 1.
Non sappiamo per certo in quanto tempo polinomiale accada; alcuni sono confonti lineari e sappiamo viene verificato in un tempo al più polinomiale; la selezione stessa potenzialmente è un problema non deterministico.
2) Per rappresentare la struttura del circuito quasi Hamiltoniano come problema NP-Hard, dobbiamo provare che tutti i linguaggi in NP sono riducibili in tempo polinomiale con il problema del circuito quasi Hamiltoniano. L'idea quindi del circuito hamiltoniano è di costruire un grafo G sulla base degli n vertici, per una certa costante k; ne aggiungerà almeno k – 1, ciascuno connesso a tutti gli altri. Dato che il circuito hamiltoniano comprende tutti gli archi, rimarrà sempre almeno un vertice di grado 1 ad ogni computazione polinomiale, considerabile per il circuito Hamiltoniano completo come ciclo a sé stante, continuando a computare.
In particolare, data questa ultima considerazione, matematicamente descriviamo:
$G' = (\{v_1, \dots, v_n\} \cup \{v_1', \dots, v_n'\}$ e $E \cup \{(v_i, v_i') \mid 1 \le i \le n\}$
ogni vertice visiterà tutti gli altri con un ciclo $v_i - v_i' - v_i$ (perché almeno un vertice fa ciclo con sé stesso).
Pertanto, il grafo è quasi-hamiltoniano se e solo se G è Hamiltoniano, togliendo il vertice in più.
Se ciò avviene, da in output SI altrimenti manderà in output NO.

oppure

Per completare la prova, dobbiamo dimostrare che è NP-Hard, usando la riduzione di un altro problema NP-completo, in questo caso, sfruttando la riduzione per mezzo del problema del circuito Hamiltoniano.
Dobbiamo, come per il caso precedente, partire da un grafo che chiamiamo G.
Per ogni insieme di vertici, si aggiunge almeno un vertice per ogni iterazione in G
Voglio dimostrare che G ha un circuito hamiltoniano se e solo se H ha un circuito quasi-hamiltoniano.

- Supponiamo che G abbia un circuito hamiltoniano. Aggiungo un vertice senza collegargli nessun arco. Il sottografo certamente contiene un circuito hamiltoniano e raggiungiamo tutti i vertici meno

uno. Questo vertice verrà aggiunto successivamente in un ciclo da H, tale che esso colleghi tutti i vertici e sia considerabile sottoinsieme del precedente (quindi H contiene un ciclo con tutti i vertici considerando anche il vertice tralasciato da G).

- A queste condizioni, G contiene per forza tutti i vertici ed H contiene a sua volta tutta una serie di vertici più quello che non fa parte del ciclo di G. Le due condizioni, come richiesto, vanno di pari passo, affinché G sia hamiltoniano se e solo se H è quasi hamiltoniano.

Dato inoltre che H è praticamente una copia di G, ma solo dei vertici utili (compreso il vertice mancante per G), si ha la correttezza della prova. Inoltre, affermiamo che dato l'insieme di vertici di partenza, la riduzione è corretta, avendo che l'assegnazione di vertici e archi cresce di un fattore costante, nonostante la loro ricerca sia impiegata in tempo lineare. Quindi, anche la riduzione è corretta, dato che il vertice è isolato. Alla luce di tutti questi fatti, anche il problema del circuito quasi-Hamiltoniano è NP-Completo.

Esercizio 2

Considera i seguenti problemi:

$$\begin{aligned}\textsc{SetPartitioning} = &\{\langle S\rangle \mid S \text{ è un insieme di numeri interi che può essere suddiviso}\\ &\text{in due sottoinsiemi disgiunti } S_1, S_2 \text{ tali che la somma}\\ &\text{dei numeri in } S_1 \text{ è uguale alla somma dei numeri in } S_2\}\\ \textsc{SubsetSum} = &\{\langle S, t\rangle \mid S \text{ è un insieme di numeri interi, ed esiste } S' \subseteq S\\ &\text{tale che la somma dei numeri in } S' \text{ è uguale a } t\}\end{aligned}$$

Sappiamo che SetPartitioning è un problema NP-completo.

- Dimostra che SubsetSum è in NP.
- Dimostra che SubsetSum è NP-Hard usando SetPartitioning come problema di riferimento.

1) La dimostrazione di un verificatore V per SUBSET-SUM è articolata in questo modo:
Il certificato è il sottoinsieme stesso, dato che abbiamo un insieme di numeri e dobbiamo poi capire se appartengono all'insieme dei numeri completo.
L'idea di verificatore in tempo polinomiale è la seguente per V:
V = Su input <<S, t>, c>, tale che c è una collezione di numeri (certificato), S è l'insieme di tutti i numeri, c è il certificato:

a. Controlla se c è una collezione di numeri la cui somma è t
b. Controlla se la somma S contenga tutti i numeri
c. Se entrambi passano, *accetta*, altrimenti *rifiuta*

Non sappiamo per certo in quanto tempo polinomiale accada; la selezione stessa sugli insiemi e la determinazione dei due sottoinsiemi verificatori potenzialmente è un problema non deterministico.
Quindi SUBSET-SUM è un problema in NP.

2) Per rappresentare la struttura di SUBSET-SUM/SS come problema NP-Hard, dobbiamo provare che tutti i linguaggi in NP sono riducibili in tempo polinomiale con SETPARTITIONING/SP.
Quindi, abbiamo a che fare:

- per SS un unico insieme tale che la somma sia t
- per SP due insiemi tali che la somma complessiva del primo sia uguale a quella del secondo

Per rappresentare SS prendiamo quindi due insiemi e affermiamo che per entrambi, la somma deve essere uguale a t. Definiamo quindi per l'insieme S_1 un insieme S_{new} a 2t – s, dove s rappresenta la somma degli elementi di S_1. In questo modo, S_1 e S_{new} rappresentano due partizioni che sommano a *t*.
Usando la riduzione, se S_{new} può essere di volta in volta partizionato in una serie di insiemi che contengono 2t – s, significa che avremo sempre che S_{new} è la somma di tutti i numeri interi delle due partizioni che contiene tutti i numeri delle due sottopartizioni.

Quindi, le istanze buone, intese come somme delle sottopartizioni, vengono sommate ad una sola (SS), ma è tale anche che la somma delle due sia spezzabile in due partizioni separate (SS), che è quello che vogliamo realizzare in tempo P.