Automi e Linguaggi Formali – a.a. 2020/2021
Prima prova intermedia – Soluzione

1. Supponiamo che L ed M siano due linguaggi regolari, e mostriamo che anche $\mathit{faro}(L, M)$ è regolare. Poiché L ed M sono regolari, sappiamo che esiste un DFA A_L che riconosce L ed un DFA A_M che riconosce M. Per dimostrare che $\mathit{faro}(L, M)$ è regolare esibiamo un automa a stati finiti A che riconosce $\mathit{faro}(L, M)$.

La funzione ricorsiva $\mathit{faro}(x, z)$ rimescola le parole x e z in questo modo:

- se $|x| = |z|$, allora il risultato è una parola che alterna i simboli di x nelle posizioni dispari coni i simboli di z nelle posizioni pari: $\mathit{faro}(x, z) = x_1 z_1 x_2 z_2 \dots x_n z_n$;
- se x è più corta di z, allora $\mathit{faro}(x, z)$ alterna simboli di x con simboli di z finché possibile, e poi continua con la parte rimanente di z;
- se z è più corta di x, allora $\mathit{faro}(x, z)$ alterna simboli di x con simboli di z finché possibile, e poi continua con la parte rimanente di x.

L'automa che accetta $\mathit{faro}(L, M)$ procede alternando transizioni di A_L con transizioni di A_M per ottenere l'alternanza dei simboli della parola $x \in L$ con quelli della parola $z \in M$. Per poter simulare l'alternanza l'automa deve memorizzare lo stato corrente di A_L, lo stato corrente di A_M e quale automa simulare alla prossima transizione. Gli stati A saranno quindi delle triple (r_L, r_M, t) dove r_L è uno stato di A_L, r_M è uno stato di A_M e $t \in \{L, M\}$ un valore che rappresenta il turno della simulazione. Le transizioni sono definite come segue:

- $(r_L, r_M, L) \xrightarrow{a} (s_L, r_M, M)$ se $\delta_L(r_L, a) = s_L$: quando è il turno di A_L si simula la transizione di A_L, si mantiene inalterato lo stato di A_M e si cambia il turno a M per la prossima transizione;
- $(r_L, r_M, M) \xrightarrow{a} (r_L, s_M, L)$ se $\delta_M(r_M, a) = s_M$: quando è il turno di A_M si simula la transizione di A_M, si mantiene inalterato lo stato di A_L e si cambia il turno a L per la prossima transizione.

Dobbiamo ora stabilire quali sono gli stati finali di A. Quando la simulazione raggiunge uno stato finale di A_L possono succedere due cose: si continua ad alternare transizioni di A_L con transizioni di A_M, oppure "scommettere" che abbiamo terminato di consumare i simboli della parola x, e continuare con la parte rimanente di z fino a raggiungere uno stato finale di A_M. Viceversa, quando la simulazione raggiunge uno stato finale di A_M, sceglie se continuare con l'alternanza oppure con la parte rimanente di x. Useremo il nondeterminismo per rappresentare queste scelte: A sarà un NFA anche se A_L e A_M sono dei DFA. Oltre al nondeterminismo dobbiamo aggiungere altri due tipi di turno: L_{suff} e M_{suff} per rappresentare le computazioni sulla parte rimanente di parola in L o sulla parte rimanente di parola in M. Quindi gli stati di A saranno delle triple (r_L, r_M, t) con $t \in \{L, M, L_{suff}, M_{suff}\}$, e dobbiamo aggiungere le seguenti transizioni all'automa:

- $(r_L, r_M, L) \xrightarrow{a} (s_L, r_M, L_{suff})$ se $\delta_L(r_L, a) = s_L$ e r_M è uno stato finale di A_M;
- $(r_L, r_M, M) \xrightarrow{a} (r_L, s_M, M_{suff})$ se $\delta_M(r_M, a) = s_M$ e r_L è uno stato finale di A_L;
- $(r_L, r_M, L_{suff}) \xrightarrow{a} (s_L, r_M, L_{suff})$ se $\delta_L(r_L, a) = s_L$;
- $(r_L, r_M, M_{suff}) \xrightarrow{a} (r_L, s_M, M_{suff})$ se $\delta_M(r_M, a) = s_M$.

Si può notare che se il turno diventa uguale a L_{suff} allora A prosegue simulando solamente le transizioni di A_L, senza cambiare lo stato r_M. Viceversa, quando il turno diventa uguale a M_{suff} A prosegue simulando solamente le transizioni di A_M, senza cambiare lo stato r_L.

Gli stati finali di A sono tutte le triple (r_L, r_M, t) tali che r_L è uno stato finale di A_L e r_M è uno stato finale di M. Lo stato iniziale è la tripla (q_L^0, q_M^0, L).

2. Consideriamo il linguaggio

$$L_2 = \{w \in \{1, \#\}^* \mid w = x_1\#x_2\#\ldots\#x_k \text{ con } k \geq 0, \text{ ciascun } x_i \in 1^* \text{ e } x_i \neq x_j \text{ per ogni } i \neq j\}.$$

Usiamo il Pumping Lemma per dimostrare che il linguaggio non è regolare.

Supponiamo per assurdo che L_2 sia regolare:

- sia k la lunghezza data dal Pumping Lemma;
- consideriamo la parola $w = 1^k\#1^{k-1}\#\ldots\#1\#\#$, che appartiene ad L_2 ed è di lunghezza maggiore di k;
- sia $w = xyz$ una suddivisione di w tale che $y \neq \varepsilon$ e $|xy| \leq k$;
- poiché $|xy| \leq k$, allora x e y sono entrambe contenute nella prima sequenza di 1. Inoltre, siccome $y \neq \emptyset$, abbiamo che $x = 1^q$ e $y = 1^p$ per qualche $q \geq 0$ e $p > 0$. z contiene la parte rimanente della stringa: $z = 1^{k-q-p}\#1^{k-1}\#\ldots\#1\#\#$. Consideriamo l'esponente $i = 0$: la parola xy^0z ha la forma

  $$xy^0z = xz = 1^q1^{k-q-p}\#1^{k-1}\#\ldots\#1\# = 1^{k-p}\#1^{k-1}\#\ldots\#1\#\#$$

  Siccome la parola z contiene tutte le sequenze di 1 di lunghezza decrescente da $k-1$ a 0, allora una delle sequenze sarà uguale alla sequenza 1^{k-p}, che è di lunghezza strettamente minore di k perché $p > 0$. Di conseguenza, la parola xy^0z non appartiene al linguaggio L_2, in contraddizione con l'enunciato del Pumping Lemma.

Abbiamo trovato un assurdo quindi L_2 non può essere regolare.

3. Per dimostrare che ogni grammatica context-free generalizzata descrive un linguaggio context-free dobbiamo dimostrare che le grammatiche generalizzate sono equivalenti alle normali grammatiche context free.

È facile vedere che le grammatiche context-free sono un caso particolare di grammatiche context-free generalizzate: una regola $A \to u$ dove u è una stringa di variabili e terminali è una regola valida anche per le grammatiche generalizzate.

Dimostreremo che consentire espressioni regolari nelle regole non aumenta il potere espressivo delle grammatiche mostrando come possiamo costruire una grammatica context-free equivalente ad una grammatica generalizzata. La costruzione procede rimpiazzando le regole con espressioni regolari con altre regole equivalenti, finché tutte le regole sono nella forma semplice consentita nelle grammatiche context-free normali:

(a) rimpiazza ogni regola $A \to R + S$ con le due regole $A \to R$ e $A \to S$;

(b) per ogni regola $A \to R.S$, aggiungi due nuove variabili A_R e A_S e rimpiazza la regola con le regole $A \to A_R A_S$, $A \to R$ e $A \to S$;

(c) per ogni regola $A \to S^*$, aggiungi una nuova variabile A_S e rimpiazza la regola con le regole $A \to A_S A$, $A \to \varepsilon$ e $A_S \to S$;

(d) rimpiazza ogni regola $A \to \emptyset$ con la regola $A \to A$;

(e) ripeti da (a) finché non rimangono solamente regole del tipo $A \to u$ dove u è una stringa di variabili e terminali, oppure $A \to \varepsilon$.

Ogni passaggio della costruzione modifica la grammatica in modo da essere sicuri che generi lo stesso linguaggio. Inoltre, la costruzione termina quando tutte le regole sono nella forma "standard" $A \to u$, senza operatori regolari.