1

Scrivere l'enunciato del Pumping Lemma per i linguaggi liberi.

2

Sia r = b*a | b*a (epsilon|a|b)* (epsilon|a|b) e sia D il DFA minimo per il riconoscimento di L(r). Dire quanti stati ha D e quanti di questi stati sono finali.

3

Sia L = { $a^i b^j c^k$ | i >= 0 e j + k = i}. Se L è un linguaggio libero rispondere "SI" e fornire una grammatica libera che lo genera. Se invece L non è libero, allora rispondere "NO" e fornire una stringa z da utilizzare con successo nella dimostrazione per contraddizione rispetto al Pumping Lemma dei linguaggi liberi.

4

Sia G la grammatica A -> aAa | bAb | epsilon . Il parser LALR(1) per G ha due conflitti s/r, un conflitto nello stato che si raggiunge dallo stato iniziale con il cammino etichettato a e un conflitto nello stato che si raggiunge dallo stato iniziale con il cammino etichettato b. Si supponga di risolvere entrambi i conflitti a favore dello "shift". Se, sotto questa ipotesi, il parser riconosce tutte le parole di L(G) rispondere "TUTTE", altrimenti indicare una parola in L(G) che non viene riconosciuta dal parser.

5

Sia G la seguente grammatica

S -> aS | B | epsilon

B -> bB | S

Chiamiamo A l'automa caratteristico per il parsing LALR(1) di G, H lo stato iniziale di A, T la tabella di parsing LALR(1) per G. Se T non contiene alcun conflitto nello stato H[babB] rispondere "NO CONFLICT". Altrimenti, per ciascuna X tale che T[H[babB], X] contiene un conflitto, dire specificando a quale X si fa riferimento: a) di che conflitto si tratta; b) quale/i riduzione/i sono coinvolte.

6

Sia G la seguente grammatica

S -> aAaa | abBab

A -> aA | a

B -> aB | b

Chiamiamo A l'automa caratteristico per il parsing LR(1) di G. I lo stato iniziale di A, T la tabella del parsing LR(1) per G. Se non ci sono conflitti nello stato I[abaaB] di T rispondere “NO RIDUZIONI”. Altrimenti per ciascuna X tale che la entry T[I[abaaB], X] contiene qualche riduzione dire, specificando a quale X si fa riferimento, di che riduzione si tratta.

7

Sia S1 il seguente SDD

S -> B {eval(B.v)}

B -> B1 or T {B.v = newNode(Or, B1.v, T.v)}

B -> T {B.v = T.v}

T -> T1 and F {T.v = newNode(And, T1.v, F.v)}

T -> F {T.v = F.v}

F -> not F1 {F.v = newNode(Not, F1.v, null)}

F -> true {F.v = newNode(True, null, null)}

F -> false {F.v = newNode(False, null, null)}

F -> (B) {F.v = B.v}

Dove la funzione newNode(label, e1, e2) crea un nodo con due figli in cui label è l’etichetta del nodo, e1 è il riferimento al figlio sinistro, e2 è il riferimento al figlio destro; null indica la mancanza del nodo figlio. Lo pseudocodice per la funzione eval(N) è la seguente

```
Eval(N){
    if(figlio destro di N =! Null) {eval(figlio destro di N)}

    Print(etichetta di N)

    If(figlio sinistro di N =! Null) {eval(figlio sinitro di N))

}
```

Si immagini di analizzare S1 con un parser LALR(1). Se l’input *not (true and not not false)* non è riconosciuto dal parser scrivete “ERROR”. Altrimenti scrivere il risultato della valutazione dell’input *not (true and not not false)*.

8

Sia S2 il seguente SDD

S -> A {while(T not empty){ num = pop(t); print(num)}}

A -> CB {push(T,1)}

B -> aCB {push(T,2)}

B -> epsilon {push(T,3)}

C -> ED {push(T,4)}

D -> bED {push(T,5)}

D -> epsilon {push(T,6)}

E -> g {push(T,7)}

Dove T è una pila inizialmente vuota. Sulla pila sono definite le comuni funzioni di push(pila, elemento) e pop(pila). Si immagini di analizzare S2 con un parser LALR(1). Se l'input gagbg non è riconosciuto dal parser scrivere "ERROR". Altrimenti scrivere il risultato della valutazione dell'input gagbg.

9

Siano

L1 = L(((a|ab)*b)* (epsilon|(a|ab)*|ab))

L2 = L(((a|ab)*b)*)

L3 = L1\L2

Dove "\" è l'operatore di differenza insiemistica. Dire, motivando la risposta, se L3 è un linguaggio regolare. In caso di risposta affermativa fornire anche il minimo DFA per il riconoscimento di L3.

10

Sia L il linguaggio delle parole sull'alfabeto {a,b} che contengono lo stesso numero di occorrenze di a e di b. Definire una grammatica LALR(1) G tale che L(G) = L. Giustificare la scelta di G.