Immagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamenteImmagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente

Per trasformare un DFA (Automa a Stati Finiti Deterministico) in una grammatica context-free equivalente, si può seguire la seguente procedura:

1. Creare un non-terminale per ogni stato del DFA.
2. Per ogni transizione δ(q, a) = p nel DFA, dove q e p sono stati e a è un simbolo dell'alfabeto, aggiungere una regola della forma: q → ap.
3. Per ogni stato finale f del DFA, aggiungere una regola della forma: f → ε.

La grammatica risultante genererà esattamente lo stesso linguaggio accettato dal DFA di partenza.

Vediamo un esempio pratico:

Consideriamo il seguente DFA che riconosce il linguaggio delle stringhe binarie che terminano con 01:

Dove q0 è lo stato iniziale e q1 è l'unico stato finale.

Applicando la procedura:

1. Creiamo un non-terminale per ogni stato: Q0 e Q1.
2. Per la transizione δ(q0, 0) = q0, aggiungiamo la regola: Q0 → 0Q0.
3. Per la transizione δ(q0, 1) = q1, aggiungiamo la regola: Q0 → 1Q1.
4. Per lo stato finale q1, aggiungiamo la regola: Q1 → ε.

La grammatica risultante è: Q0 → 0Q0 Q0 → 1Q1 Q1 → ε

Che genera esattamente lo stesso linguaggio del DFA di partenza.

Ecco degli esempi per illustrare le quattro tecniche per progettare grammatiche context-free presentate dal professore:

1. Unione di linguaggi più semplici: Supponiamo di voler generare il linguaggio L = L1 ∪ L2, dove L1 = {0^n 1^n | n ≥ 1} e L2 = {0^n 1^m | n ≠ m, n ≥ 1, m ≥ 1}. Possiamo definire due grammatiche separate G1 e G2 per generare rispettivamente L1 e L2, e poi combinarle in una nuova grammatica G che genera L1 ∪ L2.

G1: S → 0A | B A → 0A1 | 1 B → 0B1 | ε

G2: S → 0A | 0B A → 0A1 | 1 B → 0B1 | ε

G: S → S1 | S2 S1 → 0A | B (regole di G1) A → 0A1 | 1 B → 0B1 | ε S2 → 0A | 0B (regole di G2) A → 0A1 | 1 B → 0B1 | ε

1. Trasformare un DFA in CFG: Considera il seguente DFA che riconosce il linguaggio delle stringhe binarie che contengono almeno una sotto-stringa "01":

G: S → S1 | S2 S1 → 0A | B (regole di G1) A → 0A1 | 1 B → 0B1 | ε S2 → 0A | 0B (regole di G2) A → 0A1 | 1 B → 0B1 | ε

1. Trasformare un DFA in CFG: Considera il seguente DFA che riconosce il linguaggio delle stringhe binarie che contengono almeno una sotto-stringa "01":

La grammatica context-free equivalente è: Q0 → 0Q1 | 1Q2 Q1 → 0Q1 | 1Q3 | ε Q2 → 0Q2 | 1Q3 Q3 → 0Q3 | 1Q3 | ε

1. Sottostringhe collegate: Supponiamo di voler generare il linguaggio L = {x#y#z | x, y, z ∈ {0,1}\*, |x| = |z|, y è una sotto-stringa di x}. Possiamo definire una grammatica che genera prima una stringa x, poi una sotto-stringa y di x, e infine una stringa z con la stessa lunghezza di x.

S → AxBzC A → 0A | 1A | ε B → 0B0 | 0B1 | 1B0 | 1B1 | ε C → 0C | 1C | ε x → 0x | 1x | ε z → 0z | 1z | ε

1. Strutture ricorsive: Consideriamo il linguaggio L = {0^n 1^m 0^n 1^m | n, m ≥ 1}, che descrive stringhe con due copie identiche di sottostringhe della forma 0^n 1^m. Possiamo definire una grammatica ricorsiva che genera prima una sotto-stringa 0^n 1^m, e poi la ripete.

S → AB A → 0A0 | 0B B → 1B1 | ε

Altro caso particolare:

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamenteImmagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamenteEsercizi PL

Soluzione

con

Si scelga un tale che .

Funziona con

Soluzione

con

Si scelga un tale che .

Funziona con

Immagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamenteEsercizi dimostra che L sia regolare:

Per dimostrare che se L è un linguaggio regolare sull'alfabeto {0, 1}, allora anche la rotazione a destra di L, indicata come ROR(L), è regolare, possiamo utilizzare la proprietà di chiusura dei linguaggi regolari rispetto all'operazione di concatenazione e all'operazione di riflessione.

Definiamo formalmente la rotazione a destra di L:

ROR(L) = {aw | wa ∈ L, w ∈ {0, 1}\*, a ∈ {0, 1}}

La rotazione a destra di L genera tutte le stringhe ottenute concatenando una stringa w di L con un singolo simbolo a alla sua sinistra.

Ora, poiché L è un linguaggio regolare, possiamo costruire un DFA M che riconosce L.

Possiamo utilizzare M per costruire un nuovo DFA che riconosce ROR(L). L'idea è di riflettere il DFA M rispetto al suo asse verticale (invertendo le transizioni destra-sinistra) e quindi concatenare una transizione aggiuntiva per ogni stato che porta al nuovo stato iniziale (quello che era lo stato finale di M) con l'input a per ogni a in {0, 1}.

Questo nuovo DFA riconoscerà esattamente le stringhe di ROR(L), dimostrando che ROR(L) è regolare.

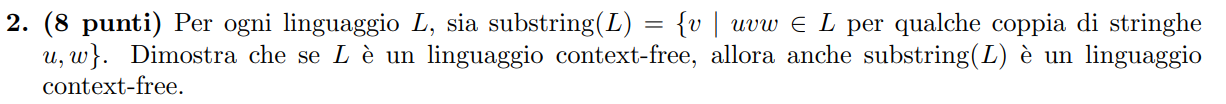
In conclusione, se L è un linguaggio regolare, allora anche ROR(L) è regolare.

Immagine che contiene testo, Carattere, schermata, algebra

Descrizione generata automaticamente

Immagine che contiene testo, Carattere, schermata, numero

Descrizione generata automaticamente



Supponiamo che L sia generato dalla grammatica context-free G = (V, Σ, R, S), dove:

* V è l'insieme dei non-terminali
* Σ è l'alfabeto dei terminali
* R è l'insieme delle regole di produzione
* S ∈ V è il non-terminale iniziale

Costruiamo una nuova grammatica context-free G' = (V', Σ, R', S') che genera substring(L) nel seguente modo:

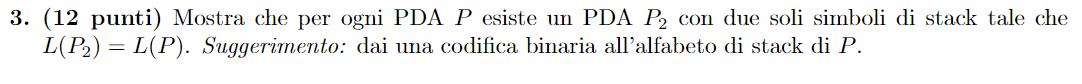
1. V' = V ∪ {S'} ∪ {A, B} (dove A e B sono nuovi non-terminali)
2. R' contiene le seguenti regole:
   * S' → ASB
   * A → aA | ε, per ogni a ∈ Σ
   * B → bB | ε, per ogni b ∈ Σ
   * Tutte le regole di R

La grammatica G' funziona nel seguente modo:

* La regola S' → ASB divide la stringa da generare in tre parti: la prima parte (generata da A) rappresenta la sottostringa u, la seconda parte (generata dalle regole di R) rappresenta la sottostringa v che appartiene a substring(L), e la terza parte (generata da B) rappresenta la sottostringa w.
* Le regole A → aA | ε e B → bB | ε generano tutte le possibili sottostringhe u e w rispettivamente.
* Le regole di R vengono aggiunte direttamente a R' per generare le sottostringhe v appartenenti a L.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteQuindi, G' genera esattamente il linguaggio substring(L) = {v | uvw ∈ L per qualche coppia di stringhe u, w}.

Altri esercizi: PDA

Dato un PDA P = (Q, Σ, Γ, δ, q0, Z0, F), dove Q è l'insieme degli stati, Σ è l'alfabeto di input, Γ è l'alfabeto di stack, δ è la funzione di transizione, q0 è lo stato iniziale, Z0 è il simbolo di stack iniziale e F è l'insieme degli stati finali, possiamo costruire un PDA P2 = (Q', Σ, {0, 1}, δ', q0, 0, F) con due soli simboli di stack {0, 1} nel seguente modo:

1. Q' = Q (gli stati rimangono invariati)
2. Σ rimane invariato
3. L'alfabeto di stack di P2 è {0, 1}
4. La funzione di transizione δ' è definita come segue:
   * Per ogni transizione δ(q, a, X) = (p, γ) in P, dove q, p ∈ Q, a ∈ Σ, X ∈ Γ e γ ∈ Γ\*:
     + Codifichiamo X e γ in binario come e rispettivamente
     + Aggiungiamo a δ' le seguenti transizioni: δ'(q, a, ) = (p, )
   * Per ogni transizione δ(q, a, ε) = (p, γ) in P, dove q, p ∈ Q, a ∈ Σ e γ ∈ Γ\*:
     + Codifichiamo γ in binario come
     + Aggiungiamo a δ' le seguenti transizioni: δ'(q, a, ε) = (p, )
   * Per ogni transizione δ(q, ε, X) = (p, γ) in P, dove q, p ∈ Q, X ∈ Γ e γ ∈ Γ\*:
     + Codifichiamo X e γ in binario come e rispettivamente
     + Aggiungiamo a δ' le seguenti transizioni: δ'(q, ε, ) = (p, )

In questa costruzione, abbiamo codificato i simboli di stack di P in una rappresentazione binaria per poterli gestire utilizzando solo i simboli di stack 0 e 1 nel PDA P2. La funzione di transizione δ' è definita in modo da simulare le transizioni di P utilizzando questa codifica binaria.

Possiamo dimostrare che L(P2) = L(P) utilizzando un argomento di simulazione: ogni mossa di P può essere simulata da una sequenza di mosse corrispondenti in P2, e viceversa. Quindi, una stringa w è accettata da P se e solo se è accettata da P2.

Questa dimostrazione mostra che, per ogni PDA P, esiste sempre un PDA P2 con due soli simboli di stack che riconosce lo stesso linguaggio L(P)

Immagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamente

Per dimostrare che DROPOUT(A) è un linguaggio regolare, possiamo utilizzare la proprietà di chiusura dei linguaggi regolari rispetto alle operazioni regolari, in particolare l'operazione di proiezione (o cancellazione di simboli).

La dimostrazione procede come segue:

1. Poiché A è un linguaggio regolare, esiste un automa a stati finiti deterministico (AFD) M che riconosce A.
2. Costruiamo un nuovo DFA M' che riconosce DROPOUT(A) a partire da M. M' simulerà M, ma con la possibilità di "saltare" (cancellare) un simbolo in qualsiasi momento durante l'elaborazione della stringa di input.
3. Il DFA M' è definito come M' = (Q', Σ, δ', q'0, F'), dove:
   * Q' = Q × (Q ∪ {q0}) (gli stati di M' sono coppie di stati di M più un possibile stato di partenza)
   * q'0 = (q0, q0) (lo stato iniziale di M' è la coppia formata dallo stato iniziale di M e dallo stato iniziale di M)
   * F' = F × Q (gli stati finali di M' sono le coppie formate da uno stato finale di M e uno stato qualsiasi di M)
   * δ' è definita come segue:
     + δ'((q, p), a) = (δ(q, a), δ(p, a)) per ogni q, p ∈ Q, a ∈ Σ
       1. (se non vengono cancellati simboli, M' simula M)
     + δ'((q, p), ε) = (q, δ(p, ε)) per ogni q ∈ Q, p ∈ Q ∪ {q0}
       1. (M' può cancellare un simbolo passando al prossimo stato di M a partire dal secondo stato della coppia)
4. È possibile dimostrare che L(M') = DROPOUT(A) utilizzando un ragionamento per induzione sulla lunghezza delle stringhe.

Poiché M' è un DFA, DROPOUT(A) è un linguaggio regolare.

Dimostriamo che L(M') = DROPOUT(A) per induzione sulla lunghezza delle stringhe.

Base dell'induzione: Consideriamo le stringhe di lunghezza 0.

* La stringa vuota ε appartiene a DROPOUT(A) se e solo se ε appartiene ad A (poiché non ci sono simboli da cancellare).
* Dall'altro lato, M' accetta ε se e solo se (q0, q0) è uno stato finale di M', cioè q0 è uno stato finale di M. Quindi M' accetta ε se e solo se ε appartiene ad A.

Passo induttivo: Supponiamo che la proprietà valga per tutte le stringhe di lunghezza minore di n, dove n > 0. Dimostriamo che vale anche per le stringhe di lunghezza n.

Sia w = a1a2...an una stringa di lunghezza n, con ai ∈ Σ per 1 ≤ i ≤ n.

(⇒) Supponiamo che w appartenga a DROPOUT(A). Allora esiste una stringa z = a1a2...aj-1ajaj+1...an+1 in A tale che w = a1a2...aj-1aj+1...an+1 (cioè, w è ottenuta da z cancellando il simbolo aj).

Per ipotesi induttiva, la stringa a1a2...aj-1aj+1...an+1 è accettata da M'. Quindi, esiste una sequenza di stati r0, r1, ..., rn in M' tale che r0 = q'0 e rn ∈ F', con δ'(ri-1, ai) = ri per 1 ≤ i ≤ n-1 e δ'(rn-1, an) = rn.

Poiché M' può cancellare un simbolo in qualsiasi momento durante l'elaborazione, significa che esiste un indice k tale che rk-1 = (p, q) e rk = (p', q') con δ'((p, q), ak) = (p', δ(q, ak)) = (p', q').

Quindi, M' può accettare la stringa w seguendo la sequenza di stati r0, r1, ..., rk-1, (p', q'), rk+1, ..., rn, che inizia con q'0, finisce in uno stato finale, e passa attraverso la configurazione (p', q') dopo aver letto i primi k-1 simboli e cancellato ak.

(⇐) Supponiamo che w sia accettata da M'. Allora esiste una sequenza di stati r0, r1, ..., rn in M' tale che r0 = q'0, rn ∈ F', con δ'(ri-1, ai) = ri per 1 ≤ i ≤ n.

Poiché M' può cancellare un simbolo, esiste un indice k tale che rk-1 = (p, q) e rk = (p', q') con δ'((p, q), ak) = (p', δ(q, ak)) = (p', q').

Consideriamo la stringa z = a1a2...ak-1ak+1...an. Poiché rk-1 = (p, q), possiamo vedere che la sequenza di stati p, δ(q, a1), δ(q, a1a2), ..., δ(q, a1a2...ak-1), q', δ(q', ak+1), ..., δ(q', a1a2...ak-1ak+1...an) è una sequenza di stati valida in M che porta M ad accettare z.

Quindi, z appartiene ad A. Inoltre, w = a1a2...ak-1ak+1...an, cioè w è ottenuta da z cancellando il simbolo ak. Pertanto, w appartiene a DROPOUT(A).

Una possibile soluzione usando espressioni regolari qui:

<https://scholar.harvard.edu/files/harrylewis/files/2011review_soln.pdf>