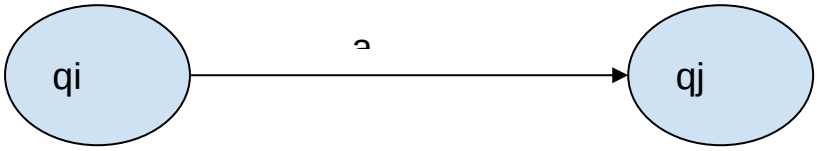
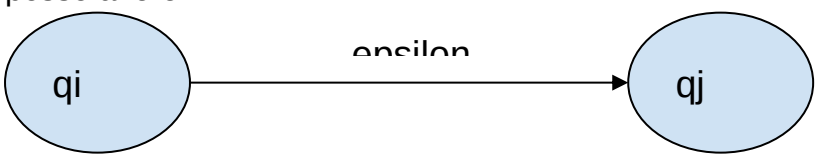
DOMANDE SECONDO ESONERO FLT1

I linguaggi deterministici sono chiusi rispetto all'unione? Giustificare la risposta
I linguaggi deterministici non sono chiusi rispetto all'unione(e nemmeno intersezione mentre sono chiusi rispetto il complemento), ad esempio, dati i linguaggi deterministici
$L1 = \{a^n b^n \setminus n \geq 1\}$
$L2 = \{a^n b^{2n} \setminus n \geq 1\}$
Il linguaggio generato dalla loro unione, $L1 \cup L2$ è un linguaggio non deterministico.
Un automa a pila dovrebbe iniziare ad impilare per i primi n caratteri a e, se la stringa appartiene a L1 dovrebbe disimpilare ad ogni b, mentre se la stringa appartiene ad L2 dovrebbe disimpilare ogni due b. Il ché rende il nuovo linguaggio non deterministico.
Non sono chiusi neppure rispetto all'intersezione, ma lo sono rispetto al complemento.

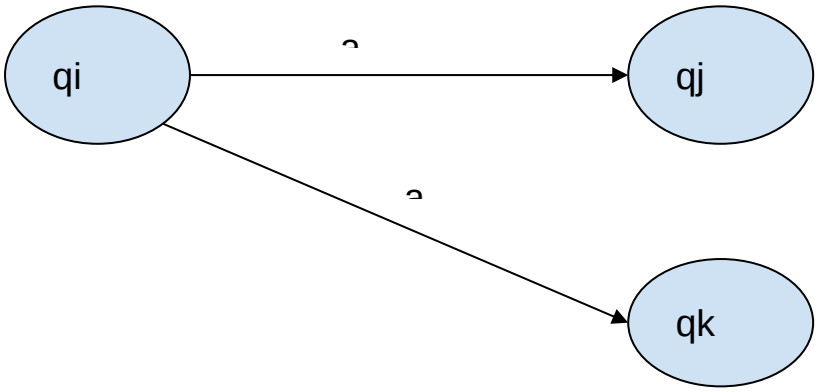
Descrivere le regole per la costruzione di una grammatica lineare destra (o strettamente lineare dx, è uguale!) equivalente ad un automa finito M
Dato un automa N = <Q, ∑, δ, q0, F> //questo automa può essere deterministico o //Non deterministico non fa differenza
voglio costruire una grammatica lin dx
G = <V, ∑, P, S>
L'idea è che ho un automa e sull'automa tipicamente passo da uno stato all'altro riconoscendo un certo simbolo.



Gli stati sono fatti per riconoscere, la grammatica per generare usa i non terminali quindi:
Dovunque c'è uno stato nell'automa io scrivo un non terminale della grammatica.
Se da qi riconosco "a" e arrivo a qj nella grammatica dirò che il non terminale qi genera "a" e poi passa la palla al non terminale qj
La grammatica non può che venire lineare dx perchè posso solo avere situazioni di questo genere.
Se io ho che da qi riconosco a e passo a qj nella grammatica dirò :
qi→ aqj
posso avere:



Se ho non determinismo posso anche scrivere:
qi→ aqj
qi→ aqk



Finisce di riconoscere l'automa quando è in uno stato finale(non necessariamente si interrompe potrebbe dallo stato finale avere altre mosse, però potrebbe anche fermarsi)
Come faccio a dire che ogni volta che sono in uno stato finale devo fermarmi?
Devo aggiungere la regola che da q' vado in ε
q'--> ε
Per ogni stato finale devo aggiungere q'--> ε (q'∈ V)
Definizione formale:
V = Q// associo a ogni stato Q un non Terminale V.
∑ = ∑
S = q_0 //lo start symbol è lo stato iniziale.
Per definire P (per considerare sia automa deterministico che non deterministico)
q,q' ∈ V
∀ a ∈ (∑∪{ε})
∀ q, q' ∈ Q, se q'∈δ(q, a) allora q → aq' ∈ P
Stati finali dell'automa generano ε nella grammatica:
∀ q ∈ F q → ε
Quello che abbiamo visto non è solo l'algoritmo per trasformarlo, ma è la prova che per ogni possibile automa a stati finiti si trova una grammatica lin dx equivalente.
Si dimostrerebbe per assurdo perché per costruzione è equivalente.

Data una grammatica strettamente lineare destra, definire formalmente l'automa a stati finiti equivalente.
G = <V, ∑, P, S>
N = <Q, ∑, δ, q0, F>
L'idea è la stessa di prima all'incontrario.
Formalmente:
Ad ogni non terminale della grammatica associo uno stato dell'automa.
Q=V U {T} //T è uno stato aggiuntivo che serve in caso di terminazione con un terminale.
∑ = ∑
q_0=S //lo stato iniziale è lo start symbol.
δ: $\forall_{regola}$ B→ aC ∈ P allora C ∈**δ**(B,a) //appartiene e non uguale perché consideriamo il non deterministico
∀ B → a ∈ P T ∈δ(B, a)
∀ B →ε ∈ P B∈ F
Quindi:

F= {T}U{B \ B → ε ∈ P} Se B → ε, lo stato B è finale.
Saranno finali gli stati che producono ε nella grammatica, e lo stato aggiuntivo T.

Dimostrare che i linguaggi regolari sono tutti deterministici (nota: la dimostrazione si ottiene
"combinando" alcune dimostrazioni fatte nel corso, delle quali si richiede solo l'enunciato)

Tutti i linguaggi regolari sono deterministici perché posso riconoscerli con un automa a stati finiti. Ogni automa a stati finiti puó anche non essere deterministico ma è sempre trasformabile in un automa a stati finiti deterministico che a sua volta è un caso particolare di un automa a pila deterministico, che non usa la pila.

Definire formalmente gli automi a pila deterministici e quelli non deterministici

Automa a pila non deterministico:
N= <Q,∑,Γ, δ, q_0,Z_0,F>
Q= stati {q_0,q_1,q_2}
∑ = alfabeto di input {a,b}
Γ(gamma maiuscolo) = alfabeto della pila {A,B}
δ = Qx∑xΓ → $2^{(QxΓ^*)}$ *// Dato uno stato, un simbolo in input e uno sulla pila mi manda in un insieme di coppie, dove in ogni coppia il primo elemento è uno stato e il secondo è una stringa su Gamma maiuscolo* Γ
q_0= stato iniziale
Z_o= simbolo pila vuoto
F = Stati finali {q_2} // può esserci o non esserci se riconosce per pila vuota non c'è.

Se eliminiamo il non determinismo, dando una sola possibilità di mossa dato lo stato ed il simbolo otteniamo un automa a pila deterministico. Non possiamo avere più di una ε-mossa. Quindi le condizioni sono:

1. | δ(q,a,X) | <= 1
1. δ(q,a,X)

2. δ(q,ε,X) // con una “a” posso prendere 2 strade diverse, non voglio queste 2 situazioni insieme
3. | δ(q,ε,X) | <= 1 // non voglio più di una ε mossa

Quindi questo automa a pila è deterministico.I deterministici ci interessano molto perché sono riconoscibili in tempo lineare.

Dato un automa a stati finiti deterministico M, dare le regole per la costruzione di un automa finito che riconosce il complemento di L(M)

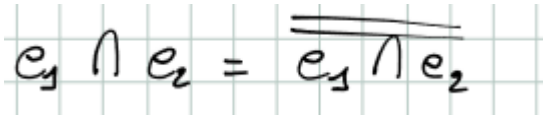
Tutti i linguaggi regolari sono deterministici perché posso riconoscibili con un automa a stati finiti. Ogni automa a stati finiti puó anche non essere deterministico ma è sempre trasformabile in un automa a stati finiti deterministico che a sua volta è un caso particolare di un automa a pila deterministico, che non usa la pila.

Tutti gli stati finali diventano non finali e viceversa(quindi gli stati invertono la polarità), si aggiunge uno stato Pozzo per il quale si definiscono le mosse per ogni terminale che non era definito da ogni stato. Le transizioni rimangono tali, per qualunque transizione mancante dell’alfabeto da ogni stato, la mando nel pozzo.
Quando si è nello stato pozzo, qualunque transizione da P ritorna in essa dato che non sono definite nell’automa di partenza.

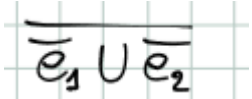
$\underline{M} = <\underline{Q}, \sum, \underline{\delta}, q_0, \underline{F}>$
$\underline{Q} = Q \cup \{p\}$ // gli stati sono quelli di prima + stato pozzo
$\underline{F} = (Q-F) \cup \{p\}$ // quelli che prima non erano finali + lo stato pozzo
Definiamo la nuova funzione δ:

$\forall q \in Q, \forall a \in \sum$ se $\delta(q, a)$ è definita $\underline{\delta}(q, a) = \delta(q, a)$.
se $\delta(q, a)$ non definita $\underline{\delta}(q, a) = P$ *// vai nel pozzo*

$\forall a \in \sum$ $\underline{\delta}(p, a)=p$ Ogni simbolo dell’alfabeto a partire dallo stato pozzo torna a se stesso in quanto nell’automa iniziale ovviamente non era definito.
Su un automa non deterministico questo es non funziona più, però sappiamo che sono equivalenti basterebbe prima trasformarlo.
Deduco che se io ho 1 qualunque Linguaggio appartenente alla classe dei Linguaggi regolari il suo complemento è anch'esso un Linguaggio appartenente alla classe dei linguaggi regolari, perché il linguaggio è chiuso rispetto a quell’operazione, infatti riesco a costruire un automa a stati finiti che lo denota.
Possiamo dimostrare che i linguaggi regolari sono chiusi rispetto all’intersezione:



(sono delle L)
usando De Morgan



se ho la negazione dell’intersezione di 2 insiemi, equivale all’unione della negazione dei 2 insiemi.

E quindi conoscendo come costruire gli automi per unione e complemento, può ottenere l'automa dell'intersezione(non deterministico con epsilon mosse, poi da minimizzare e rendere deterministico)

I linguaggi delle palindrome (con e senza marca di centro) sono deterministici? Commentare le risposte.
I linguaggi delle palindrome senza marca di centro non sono deterministici: un automa a pila inizia ad impilare ma non sa quando disimpilare, deve provare tutte le alternative. Con marca di centro invece impila fino alla marca di centro, e poi disimpila.

Il linguaggio delle palindromi (con e senza marca di centro) è regolare? E' deterministico? Commentare le risposte
Il linguaggio delle palindromi con e senza marca di centro non sono regolari perché ci sono delle dipendenze che con un linguaggio generato da una grammatica regolare non riesco a ottenere mentre per quanto riguarda il determinismo, quello con marca da centro è deterministico perché si può prendere una sola strada per esempio abcba dato che abbiamo la marca da centro riusciamo a dare un'unica strada cioè a destra della marca di centro il primo termine non ancora visitato è una b poi vado a sinistra e il primo termine non ancora visitato dopo la marca di centro deve essere una b e così via. Mentre se siamo in assenza di marca da centro abbiamo 2 possibilità ad ogni iterazione o il simbolo che viene considerato fa parte della seconda parte della stringa e quindi bisogna iniziare a disimpilare oppure non lo è quindi dovremo tenere in memoria le 2 possibilità e fare una fork .

1)Quali sono gli stati finali degli automi a stati finiti che "guidano" il parser shift/reduce nell'analisi LR(0)? Commentare la risposta
2)Quali linguaggi sono riconosciuti dagli automi a stati finiti usati nel parsing LR(0)? Quali degli stati di tali automi sono finali? Perché?
1/2)Il parsing LR(0) riconosce i linguaggi regolari costituiti da prefissi ascendenti. Ogni stato è finale perché l'automa deve riconoscere i prefissi ascendenti, e anche la ε lo è. È perciò necessario che l'automa si possa fermare ad ogni stato.

Definire le nozioni di "configurazione" e "mossa" per gli automi a pila.
Γ=Alfabeto della pila
Una configurazione di un automa a pila definisce una descrizione in un certo istante di
$(q,y,\gamma) \in Q x \Sigma^* x \Gamma^*$
ovvero lo stato in cui sono arrivato, la porzione di input ancora da esaminare e lo stato della pila.
Ho bisogno di queste informazioni per sapere esattamente dove mi trovo e riprendere in qualsiasi momento.
La mossa permette di passare da una configurazione ad un'altra basata sul delta δ, e dipende da tre valori:
dal carattere in input,
dalla cima della pila,
dallo stato dell'unità di controllo.
$(qi,ay,\gamma Z) \rightarrow (qj,y,\gamma \eta)$ se $(qj,\eta) \in \delta(qi,a,Z)$

Dove qi è lo stato in cui mi trovo, a è il simbolo in input, mentre y è quello che rimane dell'input oltre ad "a",Z è il simbolo in cima alla pila mentre ɣ è quello che c'è sulla pila oltre a Z.

(qj,y,ɣɧ) è la configurazione in cui si passa nel caso in cui il delta sia definito nel modo seguente: (qj,ɧ) ∈ δ(qi,a,Z)
In quanto semplicemente si è tolto il simbolo in input ed è rimasto y, si è passati allo stato qj come definito nel delta, e sulla pila oltre a ɣ che era già presente anche a seguito della eventuale pop, abbiamo aggiunto ɧ perchè era definito nel delta.
Possiamo trovarci anche in una situazione del genere nel quale si considerano le epsilon mosse, cioè non si guarda l'input:
(qi,y,ɣZ)--> (qj,y,ɣɧ) se (qj,ɧ) ∈ δ(qi,ε,Z)
Infatti l'input rimane invariato.

Se X è la stringa in input e la configurazione iniziale è (qo,X,Zo) allora una frase viene riconosciuta se:
riconosco per stato finale → (qo,X,Zo) $\rightarrow^*_n$ (q,☑,ɣ) con q∈F
Quindi non mi importa cosa rimane sulla pila.
Oppure riconosco per pila vuota→ (qo,X,Zo) $\rightarrow^*_n$ (q,ε,ε) con q∈Q

Definire le nozioni di stato e di "mossa" per un automa a pila
Uno stato indica una particolare configurazione dell'automa a pila.
La mossa permette di passare da una configurazione ad un'altra basato sul delta δ, e dipende da tre valori:
dal carattere in input,
dalla cima della pila,
dallo stato dell'unità di controllo.
(qi,ay,ɣZ)--> (qj,y,ɣɧ) se (qj,ɧ) ∈ δ(qi,a,Z)
Dove qi è lo stato in cui mi trovo, a è il simbolo in input, mentre y è quello che rimane dell'input oltre ad "a",Z è il simbolo in cima alla pila mentre ɣ è quello che c'è sulla pila oltre a Z.

(qj,y,ɣɧ) è la configurazione in cui si passa nel caso in cui il delta sia definito nel modo seguente: (qj,ɧ) ∈ δ(qi,a,Z)
In quanto semplicemente si è tolto il simbolo in input ed è rimasto y, si è passati allo stato qj come definito nel delta, e sulla pila oltre a ɣ che era già presente anche a seguito della eventuale pop, abbiamo aggiunto ɧ perchè era definito nel delta.
Possiamo trovarci anche in una situazione del genere nel quale si considerano le epsilon mosse, cioè non si guarda l'input:
(qi,y,ɣZ)--> (qj,y,ɣɧ) se (qj,ɧ) ∈ δ(qi,ε,Z)
Infatti l'input rimane invariato.

Quando uno stato I è adeguato nel parsing LR(0)?
Uno stato I è adeguato nel parsing LR(0) quando non presenta conflitti di tipo shift/reduce o reduce/reduce.

Quando uno stato I è adeguato nel parsing LR(0)? Fare un esempio di un conflitto shift/reduce per un linguaggio di programmazione.

Uno stato I è adeguato nel parsing LR(0) quando non presenta conflitti di tipo shift/reduce o
reduce/reduce.
Un esempio di conflitto shift/reduce nei linguaggi di programmazione è lo statement if/then/else:
<block> → if <cond> then <block> else <block>| if <cond> then <block>
Supponiamo di aver messo nella pila:
... if <cond> then if <cond> then <block>
e di avere in input
else ...
Questo è il classico esempio di conflitto shift reduce:

- posso fare una reduce, prendo la parte destra e la riduco a block. Se scelgo di fare una reduce al passo dopo avrò if <cond> then <block> ed in input continuo ad avere l'else
- Se invece faccio uno shift continuo ad avere sulla pila quel che avevo prima più l'else ... if <cond> then if <cond> then <block> else ...In questo caso sto legando l'else all'if più vicino, mentre nel caso precedente lo legavo all'if più lontano.

Quando un linguaggio è deterministico? I linguaggi finiti sono deterministici? E i linguaggi
regolari? E i linguaggi context free? Motivare le risposte.
risposta edo: mia
Un linguaggio è deterministico quando può essere riconosciuto da un automa a pila deterministico in cui sono presenti queste regole:
1| δ(q,a,X) | <= 1
1 δ(q,a,X)
2δ(q,ε,X) // con una “a” posso prendere 2 strade diverse, non voglio queste 2 situazioni insieme
3| δ(q,ε,X) | <= 1 // non voglio più di una ε mossa

I linguaggi finiti sono regolari, ed i linguaggi regolari sono deterministici.
I linguaggi context free non sono deterministici, un esempio di linguaggio context free è il linguaggio delle palindrome senza marca di centro, che può essere riconosciuto solo con un automa a pila non deterministico, però i linguaggi deterministici sono contenuti all’interno dei context free perchè sono riconosciuti da un automa a pila, però non tutti i linguaggi context free sono deterministici , ad esempio i linguaggi inerentemente ambigui e la palindrome senza marca di centro.

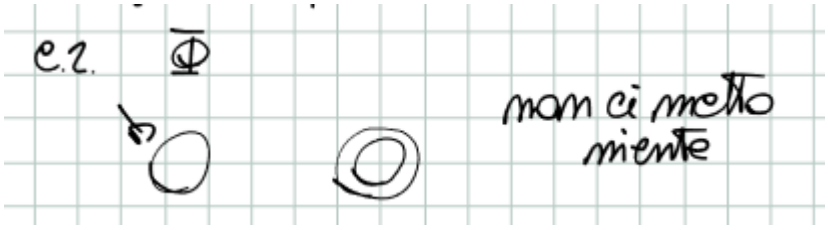
(A) Quale relazione (eguaglianza/inclusione propria/inclusione impropria/contenimento proprio/contenimento improprio c’è tra i linguaggi regolari e quelli deterministici? (B) Fornire una dimostrazione per la risposta.

(A) la classe dei linguaggi regolari è propriamente inclusa in quella dei linguaggi deterministici (B) Tutti i linguaggi regolari sono deterministici perché posso esprimerli con

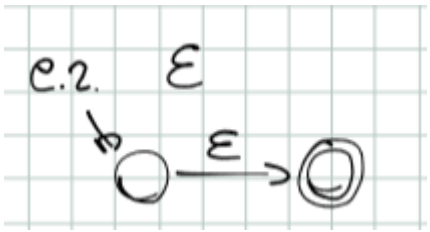
un automa a stati finiti. Ogni automa a stati finiti puó anche non essere deterministico ma è sempre trasformabile in un automa a stati finiti deterministico che a sua volta è un caso particolare di un automa a pila deterministico, che non usa la pila.
Ma non tutti i linguaggi deterministici sono regolari, basta un solo esempio: il linguaggio delle palindrome con marca di centro, oppure il linguaggio a^nb^n, sono deterministici ma non regolari.

Il linguaggio a* è LR(0)? Motivare la risposta.
Non è LR(0) perchè ci sono dei conflitti shift reduce, in quanto non so se devo fare lo shift di un'altra "a" oppure l'input è finito e quindi ridurre.

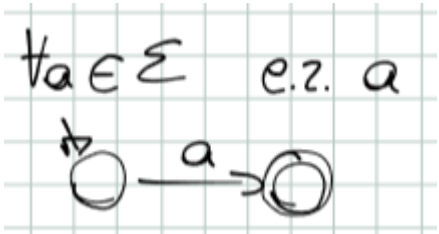
3 passo dimostrazione: passare da e.r a automa a stati finiti :
Per fare questa dimostrazione utilizzando le ε mosse è più semplice:
Per ogni espressione regolare corrisponde un automa a stati finiti equivalente.
Lo definiamo per casi:
1)Linguaggio vuoto è una e.r e corrisponde a un 'automa così:



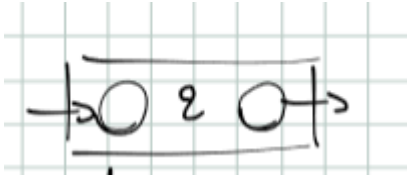
2) ε è una e.r:



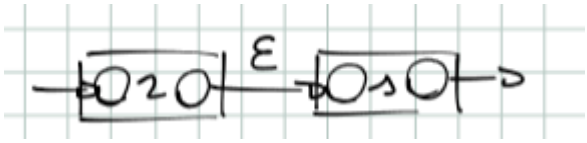
3) per ogni simbolo appartenente all'alfabeto:



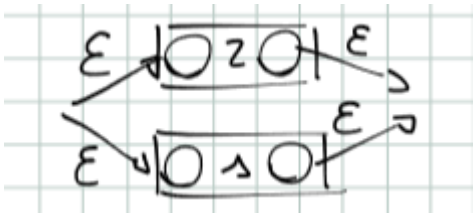
Se r è e.r, lo definisco come :



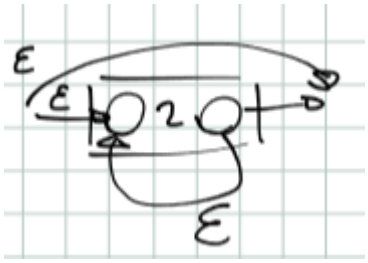
4)Se r e s sono e.r allora anche:
r**.**s è una e.r:



5)r U s è una e.r:



6)r^* è una e.r:

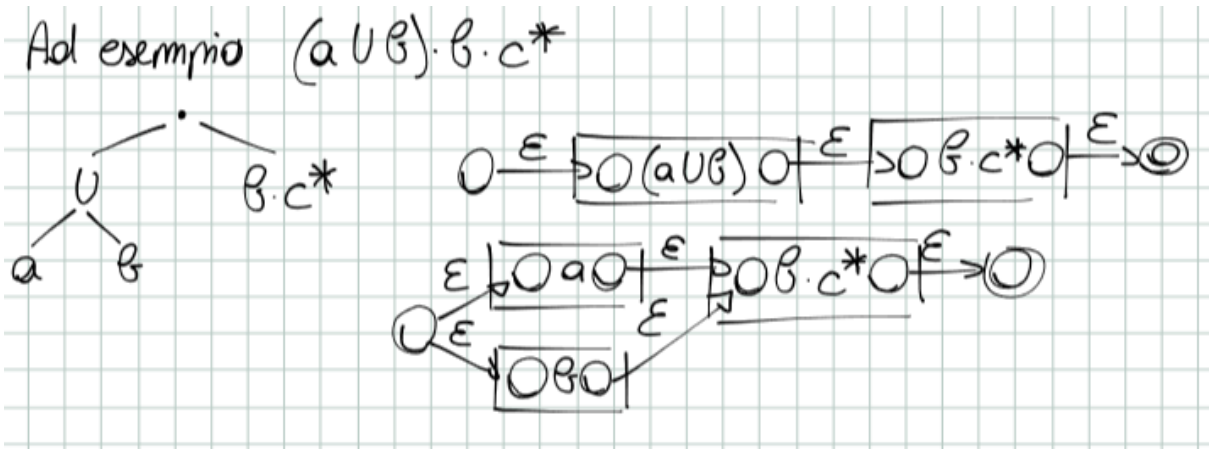


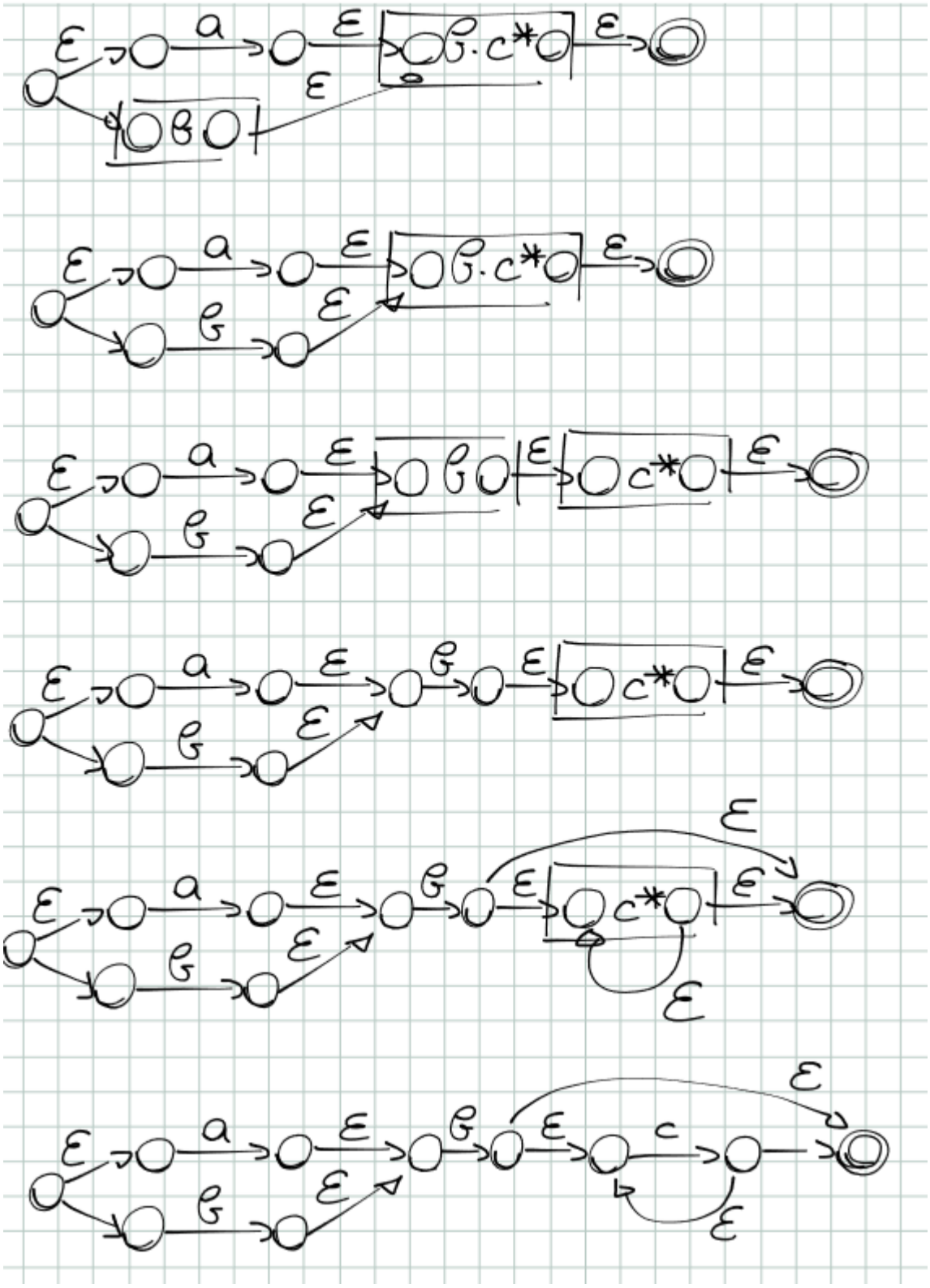
senza la freccia grossa che va in epsilon dovremmo generare almeno una r e sarebbe quindi r^+.

E' una dimostrazione induttiva e per casi.

Per dimostrare che per ogni e.r c'è un automa corrispondente seguo questa traccia che ho appena visto.

Es:





Ps:Manca la freccia per indicare l'inizio in ognuno.

Per qualunque linguaggio regolare esiste un automa a stati finiti deterministico in grado di riconoscerlo.

Dimostrazione che da qualunque gram. context free riesco a generare un automa a pila che la riconosce.

Dovremmo dim che L(context free)≡L(Automa a pila)

dovremmo dimostrare il doppio contenimento ma dimostreremo solo C.

L'idea di base è usare la pila per simulare la generazione della grammatica, partendo dallo start symbol e sostituendolo con le parti destre delle regole(usando il non determinismo se ci sono degli or).
Si fanno tutte le generazioni possibili e se corrisponde con la frase in input quest'ultima appartiene al linguaggio.
In realtà non appena sulla pila ho un terminale posso verificare se è uguale a quello che ho in input, se non è uguale fallisco localmente.
Si usa la pila per fare tutte le derivazioni possibili facendo delle fork e prendendo varie strade parallele e si verifica il prima possibile al primo terminale.Questa è appunto un' ottimizzazione per evitare di valutare tutto solo alla fine dell'input.
Dimostrazione:
G=(V,Σ,P,S)
M=(Q,Σ,Γ,δ,qo,Zo,/) "/"non ci sono stati finali e quindi accetta per pila vuota
Q={qo} usa solo la pila per le transizioni//abbiamo solo uno stato perché accetta per pila vuota
Γ={Zo}U Σ U V le forme di frase cioè terminali e non terminali
Inizializzazione:
{(qo,↙S)} ∈δ(qo,ε,Zo)
Per iniziare una derivazione devo mettere lo start symbol e il simbolo di fine stringa.
Terminazione:
{(qo,ε)} ∈δ(qo,↙,↙)
Questo automa accetta per pila vuota.
∀A→ α ∈ P (per ogni regola della grammatica) abbiamo due casi:

1) α=aB (qo,B^R) ∈ δ(qo,a,A) con a∈Σ e B∈$(\Sigma UV)^*$ si applica la regola solo dopo aver verificato se in input abbiamo "a" e sulla pila abbiamo "A"
2) α=XB (qo,B^RX) ∈ δ(qo,ε,A) con X∈V e B∈$(\Sigma UV)^*$ in questa situazione non guardo l'input devo andare alla cieca.

Per ogni simbolo dell'alfabeto:
∀a∈Σ {(qo,ε)} ∈δ(qo,a,a)
se sulla cima della pila ho lo stesso simbolo in input faccio una pop e proseguo.Per fallire basta non mettere nessuna regola.
In conclusione applicando queste regole, da una grammatica context free riusciamo a costruire in modo automatico un automa a pila.
ps: Fino ad ora abbiamo parlato di automa a pila non deterministici che a differenza degli automi a stati finiti per gli automi a pila c'è differenza tra deterministici e non deterministici.

I linguaggi deterministici sono chiusi rispetto all'intersezione? Giustificare la risposta
No non lo sono ad esempio dati i linguaggi
L1=$a^nb^nc^*$
L2=$a^*b^nc^n$
La intersezione è addirittura context sensitive

Definizione parte riducibile nel parsing bottom up shift\reduce.
La riduzione deve corrispondere a un passo all'indietro della mia derivazione.

Definizione parte riducibile:
Una parte riducibile di una forma sentenziale destra ɣ è una porzione di ɣ dove c'è la forma sentenziale β e la riduzione A→ β produce la forma sentenziale destra precedente nella derivazione right most di ɣ.

Ovvero: $S \rightarrow^{*}_{R\,M} \alpha A W \rightarrow_{RM} \alpha \beta W$

Dove αβW sarebbe ɣ mentre $\alpha,\beta \in (\Sigma \cup V)^{*}$ e $W \in \Sigma^{*}$
Forma sentenziale è un misto di terminali e non.
W sono un insieme di terminali perchè è right most.
In sostanza si ha un solo passo di derivazione right most al contrario per passare dalla forma sentenziale β a A e poi un numero * di derivazioni right most al contrario per tornare allo start symbol

Parser shift reduce LR(0), definizione automa a pila deterministico per il parser LR(0)(più in particolare si mostra l'interazione tra parser e automa).
Partiamo dalla definizione di automa a stati finiti deterministico (oracolo):
Md<Q,Σ U V, δ, I o,Q>

Σ U V alfabeto input più quello della pila.
δ=GOTO
Avendo questo devo definire in modo formale l'automa a pila per il parser LR(0)
Q'={qo}
Σ'=Σ
Γ=Σ U V U Q quindi ha anche gli stati.
δ abbiamo 2 casi:
1)

Lo stato I_S è uno stato di shift:

$\delta'=(qo,a, I_S)=(qo, I_S a I_k)$ se $\delta(I_S,a)= I_k$

2)

Lo stato I_S è uno stato di reduce $A\rightarrow \beta$ $|\beta|=n$ quindi ha n oggetti x1,x2,x3,...,xn sulla pila.

$\delta'=(qo,\varepsilon, I X_1 I_1 ... X_n I_n)=(qo, I A I')$ se $\delta(I,A)= I'$

epsilon perchè non guardo l'input per fare la riduzione, è una epsilon mossa.
Se devo fare la riduzione di n oggetti, nella pila ho n oggetti inframezzati da stati, tolgo gli n oggetti e aggiungo lo stato in cui vado con A (Cioè I ')

Appunto
Un linguaggio con prefissi come per esempio a* non è LR(0) perche bisogna considerare l'input per evitare conflitti shift/reduce o reduce/reduce

Definizione candidato valido LR(0):
L'idea è avere sulla cima della pila quello che ho prima del ∇, e di leggere il simbolo successivo.Una epsilon produzione ha un unico candidato LR(0), X→∇

Dobbiamo collegare il concetto di prefisso ascendente e candidato.
Posso proseguire solo utilizzando i candidati LR(0) che sono validi date quelle forme sentenziali sulla pila.Definizione:

Un candidato LR(0) : $A\rightarrow \beta_1 \nabla \beta_2$ con $\beta_1 , \beta_2 \in (\Sigma \cup V)^*$
è un candidato valido per un prefisso ascendente $\alpha\beta_1$ se e solo se c'è una derivazione del tipo: $S\rightarrow^*_{RM} \alpha A w \rightarrow_{RM} \alpha\beta_1\beta_2 w$

Con $w\in(\Sigma)^*$ e $\alpha\in(\Sigma \cup V)^*$.
Ogni stato dell'automa deve riconoscere un po di prefissi ascendenti(per es riconoscere a^+ Si descrive lo stato per tutti i candidati validi di un prefisso.Bisogna farlo per tutti i prefissi ascendenti. Uno stato di un automa sarà quindi un insieme di candidati validi.
Ogni stato riconosce un insieme di prefissi ascendenti.Come descrivo lo stato? Inserisco tutti e soli i candidati LR(0) validi per quei prefissi.
Le transizioni ci permettono di passare da uno stato all'altro.
appunto: Si è dimostrato che le forme sentenziali presenti nell'oracolo(automa) usano un linguaggio appartenente alla classe dei linguaggi regolari, quindi è un automa a stati finiti il cui compito è dire se fare shift o reduce.Ovviamente questo automa è più potente a seconda della k usata. Es (LR(0) , Lr(1) ecc).
Definizione di stato chiusura e goto:
Stato:Tutti e soli i candidati LR(0) validi per i prefissi ascendenti riconosciuti da quello stato li.
Funzione di Chiusura:Serve per trovare tutti e soli i candidati LR(0) validi per quello stato.
Funzione GOTO: Mi permette di passare da uno stato all'altro, una volta passato ad un 'altro stato ci applica la funzione di chiusura perché vogliamo tutti i candidati validi per quel nuovo stato.

Ps:
Linguaggi appartenenti alla classe dei linguaggi regolari sono chiusi rispetto all'unione, complemento, intersezione , star,concatenamento ecc.
Linguaggi deterministici sono chiusi rispetto al complemento ma non rispetto a unione e intersezione.
Inoltre se L è deterministico e R è regolare allora L/R (L meno R) è deterministico
Tolgo delle stringhe di L come suffisso R.
Ciò vuol dire che possiamo aggiungere a un linguaggio deterministico un suffisso regolare in fondo e non cambia nulla.