Domande esame orale Linguaggi 2020/2021:

-Espressioni Regolari.
-Parsing: predittivo (in ampiezza e profondità) e deterministico.
-Modello astratto di macchina su cui si basano i due automi: automi a pila.
-Grammatica a struttura di frase (cos'è una conseguenza, derivazione...)
-Come si definisce generato da una grammatica a struttura di frase
-Automa a stati finiti non deterministico, differenza col deterministico
-Lemma di iterazione per i linguaggi regolari
-Com'è fatta la gerarchia di Chomsky
-Grammatiche regolari (di tipo 3)
-Algoritmo CYK
-Cosa si intende per grammatica ambigua e non ambigua
-cos'è la derivazione sx
-Come si passa da una grammatica regolare ad un automa a stati finiti
-Analisi lessicale
-Automa a pila
-Nella gerarchia di chomsky quale tipo di linguaggi è accettato dagli automi a pila? non contestuali
-Come associamo un grafo ad un automa a stati finiti?
-funzione delta estesa
-Automa minimo
-Teorema di Nerode

Operazioni Compilazione:

- **Processing:** cod. sorgente modificato.
- **Compilazione:** cod. Assembly.
- **Assembly**
- **Linker/Loader:** librerie, moduli, file oggetto.

Fasi Compilazione:

- **ANALISI**:

1. **Gestione Input**.
2. Analisi **Lessicale**: scanner, lessemi, classe lessicale (Token), attributi, Tabella Simboli.
   Teoria degli Automi a Stati Finiti.
3. Analisi **Sintattica**: analizzatore sintattico (parser) organizza i token e realizza un Albero Sintattico Astratto.
4. Analisi **Semantica Statica**: usa l'Albero e la Tabella dei Simboli per verificare che il programma sia semanticamente coerente con la definizione del Linguaggio. (Type Checking)
   Statica perché rileva gli errori senza eseguire il programma.

- **SINTESI**:

1. **Generazione Codice Intermedio:** dall'Albero e Tabella Simb. si ottiene codice intermedio in istruzioni elementari, indipendenti dall'Architettura.
2. **Ottimizzazione:** ridurre tempo e spazio.
3. **Generazione Codice Oggetto:** dipendente dall'Architettura ma indipendente dal Linguaggio di Alto Livello.
4. **Ottimizzazione Peep-Hole**.

Parsing (L non-cont)

input analizzato	input da analizzare
analisi	predizione

Input: G e w.
Output: derivazione di w del L generato daG.
$O(n^3)$, derivazioni sx, endmaker (S'→S#), esplorazione in **ampiezza/profondità** (look-ahead, backtracking), **ricursioni dx** (Greibach).

Teoria Linguaggi Formali:

- **Alfabeto, lettere, parole, parola vuota.**
- **Fattore:** lettere consecutive all'interno della parola. (Prefisso, Suffisso, o Fattore Proprio).
- **Linguaggio Formale**: insieme di parole. Finito o infinito, come definirlo (Teorema di Cantor).

Grammatica a Struttura di Frase:

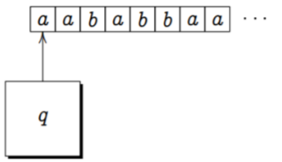
- **Quadrupla:** Vocabolario Totale, Simboli terminali, Variabili e Simbolo Iniziale, Produzioni
- **Conseguenza diretta:** se β si ottiene individuando un fattore in α che è lato sx di una produzione e sostituendolo con il lato dx.
  Conseguenza: seq. di parole, ognuna conseg. diretta della precedente, in cui la prima è α e l'ultima è β.
- **Forme sentenziali**: conseguenze del simbolo iniziale.
- **Linguaggio generato**: forme sentenziali che non contengono variabili.
- Risolvere problemi di **Ricognizione** e **Parsing**. **Classificazione Grammatiche**.

Gerarchia di Chomsky: in base alla complessità delle forme sentenziali.

- **Tipo 0:** L ricorsivamente enumerabili, massima espressività. G. a Struttura di Frase.
- **Tipo 1:**
  - **G. Contestuali:** sx V → dx parola != ε
    Sensibili al contesto, Alg. molto costosi.
    Particolari grammatiche monotòne.
  - **Monotòne:** lato dx ≥ sx.
    Per ogni G. Monotona si può costruire una G. sensibile al contesto equivalente.
- **Tipo 2:** G.**Non Contestuali:** sx 1 V.
  Sottoclasse dei L. Tipo1. **Alg**. moderat. efficienti.
- **Tipo 3:** G.**Regolari:** sx 1 V → dx t/t+V.
  Sottoclasse dei L. Tipo 2. **Alg**. molto efficienti.

Automa a Stati Finiti Deterministico:

Quintupla:



- **Q,** insieme stati
- **Σ,** Alfabeto Input
- **q_0,** stato iniziale
- **F,** insieme stati Finali.
- **δ : Q x Σ = Q,** funzione transizione.
  Estendere funzione per lavorare con le parole.
  $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$
  $\hat{\delta}(q, \varepsilon) = q$,

Grafo diretto con frecce etichettate: limite al numero di frecce che escono da uno stato per una data parola.
Parole accettate: esiste cammino da q_0 a uno stato finale.

Automa a Stati Finiti NON Deterministico:

No limitazione al num. di frecce uscenti da uno stato per una parola. **Ventaglio di possibilità** o nessuna.

- **δ : Q x Σ = ℘(Q)** (stato x lettera= insieme Stati)

Parola accettata: almeno un cammino che da q_0 termina in uno stato finale.

Esiste un **A. Deterministico equivalente**, costruibile con **Alg. di Determinazione. A'**:

- **Q' = ℘(Q)**
- **s_0 = {q_0}**
- **stati finali:** tutti i sottoinsiemi di Q che contengono almeno un elemento di **F**.
- **δ' : Q' x Σ = Q',** eseguita per tutti gli stati contenuti in **s**.
  Se s non appartiene alla lista degli stati lo appendo alla lista degli stati. Se s contiene stato finale di A, aggiungo s a F'.

Stati accessibili e Inaccessibili.

Automa a stati finiti NON Det. con ε Transizioni:

Quintupla: **$\delta : Q \times (\Sigma \cup \{\varepsilon\}) = \wp(Q)$**
(stato*lettera o stato*ε)

Parola accettata: se esiste cammino, e se **w** può essere scritta come seq. di lettere e parole vuote.

Potenziali infinite ε-transizioni: **ε-chiusura** di uno stato. **Insieme stati** che da uno stato q si possono raggiungere solo con ε-transizioni.

Esiste un **A. Deterministico equivalente. A':**

- **$Q' = \wp(Q)$**
- **$s_0 = C_\varepsilon \{q_0\}$,** ε chiusura dello stato iniziale.
- **stati finali:** tutti i sottoinsiemi di Q che contengono almeno un elemento di **F**.
- **$\delta' : Q' \times \Sigma = Q'$,** eseguita per tutti gli stati contenuti in **s**. Poi faccio l'unione di tutti questi stati, e infine ne faccio **C_ε.**

Espressioni Regolari:

Rappresentare **linguaggi infiniti** con una **descrizione finita,** ovvero le operazioni che compio su di essi.
Con le **operazioni**: unione, intersezione, complemento, concatenazione, potenza, chiusura di Kleene (unione di tutte le potenze di un L. Tutte le parole ottenute concatenando un numero arbitrario di parole di L) {**∅**} {**ε**}.
Operazioni regolari: unione, concatenazione e Kleene

Si aggiungono a **Σ** le lettere **∅**, **+**, *****, **(**, **)**.
Esp. regolari: le parole ottenute applicando le regole:

- ogni **lettera** è un'espressione regolare, **∅** è espressione regolare.
- Se E e F sono esp. Regolari allora **(E+F)**, **(EF)** e **E***, sono espressioni regolari.

A ogni espressione regolare è **associato** un **linguaggio** denotato da tale espressione. Definito dalle regole:

- Ogni **lettera** denota il linguaggio {lettera}.
  ∅ denota il linguaggio vuoto.
- E e F denotano **L_E** e **L_F**. I Linguaggi denotati dalle esp. regolari di unione, concat. e Kleene sono:
  $L_E \cup L_F$, **$L_E L_F$**, **L_E***.

Teorema di Kleene:

Un **L** è regolare se e solo se è riconosciuto da un **A** a stati finiti.
Corrispondenza effettiva quindi:

- Da esp. regolare a Automa (**sintesi**)
- Da Automa a Esp. regolare (**analisi**)

SINTESI: se E e F sono esp. regolari, un esp. regolare può essere:

- **Lettera:** restituisco **A** corrispondente.
- **∅, ε:** restituisco **A** corrispondente.
- **Unione:** calcolo Automi di E e F, poi costruisco E+F
- **Concatenazione:** " " " EF.
- **Kleene:** " " " E*.

ANALISI: **L** costituito solo dalle etichette dei cammini che hanno origine e fine in uno stato fissato e che attraversano solo stati di indice **≤ k**. Cerchiamo di trovare un esp. regolare **E_{ijk}** per questo linguaggio.

- **k=n stati**: no limitazione degli stati da attraversare
- **k=0**: non ci sono stati di **indice ≤ 0**, quindi deve esserci un **arco diretto** da q_i a q_j quindi E_{ij0}= somma **etichette archi**, altrimenti E_{ij0}=**∅.**
- **k>0**: i cammini possono essere di due tipi
  1. origine in q_i e termine in q_j, attraversano solo stati con indice **≤ k-1.**
  2. concatenazione di un **segmento iniziale** (q_i-q_k), zero o più cammini con origine e termine in q_k, e un **segmento finale** (q_k-q_j). Attraversano solo stati di indice **≤ k-1.**

E_{ijk}= somma etichette cammini dei tipi citati sopra.

Conseguenze Kleene:

La classe dei L regolari è un'algebra booleana.
È infatti chiusa per:

- **complemento:** le parole prima accettate ora non lo sono più, e viceversa. Si scambiano gli stati finali con quelli non finali e viceversa.
- **Intersezione:** unione e complemento.
  $L \cap M = \Sigma^* - ((\Sigma^* - L) \cup (\Sigma^* - M))$
- **differenza insiemistica:** unione e complemento
  $L - M = \Sigma^* - ((\Sigma^* - L) \cup M)$

Automa Minimo/di Nerode:

A det. col minimo num. di stati **m**, che riconosce lo stesso **L** riconosciuto da un **A** con num. di stati **n>m**.

Equivalenza ∼: insieme sul quale è definita una relazione Riflessiva, Simmetrica e Transitiva.
Relazione di Equivalenza su un insieme L permette di ripartire gli elementi di L in sottoinsiemi (classi di equivalenza) che contengono solo elementi In relazione tra loro.
Congruenza dx: **∼ su Σ*** se ogni volta che concateno una lettera a dx di due parole equivalenti, ottengo due parole equivalenti. Se è contemporaneamente dx e sx, si dice **Congruenza**.

Equivalenza di Nerode: ∼ su Σ*
$u \mathcal{N}_L v$ se per ogni $y \in \Sigma^*$, $uy \in L$ se e solo se $vy \in L$.
Cioè le parole devono avere gli stessi completamenti a dx in L.
$\mathcal{N}_L$ congruenza dx e **L è l'unione** delle classi di equivalenza di $\mathcal{N}_L$.
Teorema: dato L, le seguenti preposizioni sono equivalenti:

- **L è regolare:** è possibile dimostrare che L è l'unione di classi di congr. dx di indice n.

Quindi, se abbiamo una parola **w∈L** di lunghezza **≥n,**

- L è **unione di classi di una congruenza dx su Σ^* di indice finito:** è possibile dimostrare che l'indice di $\mathcal{N}_L$ è minore o ugual a quello di **~.**
- ***N_L* ha indice finito:** è possibile costruire un A det. che riconosce L col minimo numero di stati possibile (n).

Costruzione Automa di Nerode:
- Tutti gli stati sono **accessibili**.
- Gli **stati** dell'A sono le classi di $\mathcal{N}_L$ ma anche unione di linguaggi L_q ($q\in Q$). Cioè la partizione di Σ^* nelle classi di $\mathcal{N}_L$ induce una **partizione di Q**. Le classi di tale partizione ci consentono di costruire l'automa

Questa **partizione** gode delle seguenti **proprietà**:
- **F** è l'unione di classi
- Se due elementi si trovano nella stessa classe, la loro **immagine** deve stare nella stessa classe (e il complementare)
- È la **meno fine** tra le partizioni che soddisfano le condizioni precedenti (minor numero di classi possibile)

Quindi, il **problema della minimizzazione** consiste nella ricerca della partizione di Q che soddisfa tali condizioni.

Si costruisce la sequenza di partizioni di Π_n di Q nel modo seguente:
- Π_0 ha gli stati finali separati da quelli non finali.
- Si **spezzano** poi le classi in modo da separare le coppie di stati che non soddisfano la **condizione 2**.
- Continuo a spezzare classi finché non posso spezzare più nulla e lì mi fermo. Ottenendo così il minor numero di classi possibile

Automa minimo avrà:
stati= classi della partizione Π.
Stato iniziale= classe C_0 che contiene lo stato iniziale q_0 di A.
Stati finali= classi contenute in F.

Linguaggi non contestuali:

Grammatiche Regolari: (può essere lineare dx o sx)
produzioni: V→t / tV. S→ **ε** (se non compare a dx).
Data una **G regolare** si può costruire un $\mathcal{A}$ a stati finiti che ne accetta il linguaggio, e viceversa.

G → $\mathcal{A}$
- per semplicità supponiamo non abbia S→**ε.**
- **Stati**= variabili di G e la parola vuota.
- per ogni **X→aY**, nel grafo di $\mathcal{A}$ ci sarà una freccia che va da X a Y con etichetta a.
- per ogni **X→a**, nel grafo di $\mathcal{A}$ ci sarà la freccia da X ad ε con etichetta a.
- **stato iniziale= S.**
- **unico stato finale = ε.**

Se tra le produzioni di G c'è anche **S→ε**, allora **S** va aggiunta all'insieme degli **stati finali** di $\mathcal{A}$.

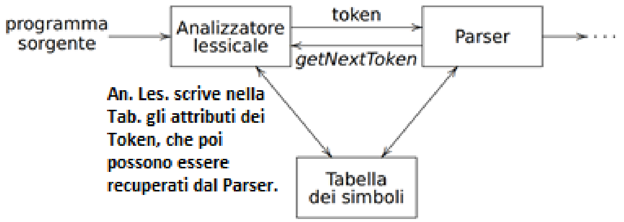
$\mathcal{A}$→G
- supponiamo che $\mathcal{A}$ sia **privo di ε-transizioni** e non accetti la parola vuota.
- **Variabili**= stati di $\mathcal{A}$.
- Per ogni freccia da X a Y, con etichetta a, si aggiunge a G la produzione X→aY.
- Se Y è uno stato finale si aggiunge anche X→a.
- **Simbolo iniziale**= stato iniziale **q_0**.

Se il L accettato da $\mathcal{A}$ contiene **ε** dobbiamo modificare la G in modo da fargli accettare **ε.** Se però S appare a dx di qualche produzione bisogna introdurre S' le cui produzioni avranno gli stessi lati dx di quelle di S, e S'→ ε.

Lemma di Iterazione L Regolari (Proprietà essenz.)
Sia L regolare, si può trovare un **intero n** per cui tutte le **parole** di L **più lunghe** di n, si possono **scomporre** in 3 pezzi: **xyz**, con **$y\neq\varepsilon$**, **y=fattore** che può essere ripetuto n volte senza uscire dal linguaggio.
(Pumping) n= num. stati.

ci sarà in $\mathcal{A}$ un cammino **da q_0** a uno **stato finale**, con **etichetta w**.
Dato che la lunghezza di $w\geq n$, troveremo uno **stato ripetuto** (y). Pertanto $\mathcal{A}$ accetta tutte le parole **xy^nz** con $n \geq 0$.

Analisi Lessicale
- **Token**: simbolo astratto, unità lessicale (es:



  parola chiave, identificatore, ecc…). Ad ogni token è associato un Pattern.
- **Lessema**: sequenza di caratteri associato ad un token.
- **Pattern**: descrizione della forma che i lessemi devono avere per poter essere associati ad un determinato Token. (un'**espressione regolare**)

Se più lessemi sono associati ad un Token, bisogna aggiungere un **attributo** nella Tab. dei Simboli.

L'Analizzatore Lessicale identifica il **lessema** e restituisce il **token** il cui **pattern** matcha il lessema. Il token restituito sarà quello di **massima priorità.**

Automa: si costruiscono gli **automi** per i vari **pattern** e poi si **collegano** tutti i loro **stati iniziali** con un nuovo stato iniziale comune.
Una parola **w** è **accettata** dall'$\mathcal{A}$ di uno dei **Pattern**, se esiste un percorso da q_0 allo specifico stato finale dell'$\mathcal{A}$ di quel Pattern (quindi f_j, dove **j** è il più piccolo indice, quindi quello di priorità maggiore).
Si ferma la ricerca quando avviene l'ultimo transito da uno stato finale prima dell'ingresso in **∅** (**pozzo**) che individua prefisso più lungo di w accettato da$\mathcal{A}$
Ricerca V unarie: creo l'insieme **T_n** di coppie di V,

Strumento necessario per eseguire l'**Analisi Sintattica**.
Produzioni: **1V** → t, le **G Lineari** quindi sono non-Contest. perché a sx hanno 1V.

Linguaggio di Dick:

Archetipo dei L non-Cont.
Sia G_n={ Vocabolario, Σ_n, Produzioni, S(unica variab.)} sull'**alfabeto** Σ_n={a_1,a_2,…,a_n, b_1, b_2,…, b_n}
e con le **produzioni**:

- S→ a_iSb_i, (i=1,2,…n) **seq. ben bilanciata**.
- S→ SS, **concatenazione** di due seq. ben bilanciate.
- S→ ε, **ε** è in sé una **seq. ben bilanciata**.

Il linguaggio generato da G_n è chiamato semi-Dick.

Alberi di Derivazione:

Rappresentazione della derivazione di una parola in una G non-Cont.
Albero: insieme di nodi, radice, figli, foglie, ordinamento, priorità.
Etichette nodi: radice=S, nodi interni=V, foglie= t/ε.
Se foglia= ε, allora deve essere figlia unica.
Se un nodo X ha figli α_1… α_k, allora in G deve esserci la produzione X→ α_1…α_k.

La **parola associata all'Albero** è la parola che si ottiene leggendo in ordine le etichette delle foglie.
Una **parola** quindi **appartiene a L** se e solo se esiste un Albero di derivazione associato a suddetta parola.

G non-Ambigua: ad ogni w del L generato da G, corrisponde un unico Albero di Derivazione.
G Ambigua: ad ogni w del L generato da G, corrispondono più Alberi di Derivazione.
Non sempre è possibile aggiustare G Ambigue quindi esistono L **internamente ambigui**.

Forma Normale di Chomsky:

Semplificazione G NON Contestuali:

Problemi Grammatiche: **parsing** e **ricognizione**.
Produzioni che danno fastidio:
X→ ε (ε-produzioni), X→Y (produzioni unarie)

Cerchiamo di creare **G equivalenti** senza queste prod.:
Se α→ β, allora **|β|>|α|**, oppure **|β|=|α|** ma avere un **t in più** di α.
Dobbiamo anche aggiungere **S→ε** per poter generare la parola vuota, ma solo se S non compare a dx.
Una parola di lunghezza n, può essere generata in **2n-1 passi**.

Eliminare ε-Produzioni

Una **V** è **annullabile** se mi produce ε. Non sono direttamente, ma anche con produzioni concatenate.
X→ ε, X→Y Y→ ε.

1. **S non è annullabile** (non si genera ε):
   costruiamo **G'** aggiungendogli tutte le produzioni che si ottengono cancellando nei lati dx delle prod di G, le V annullabili. Poi cancelliamo le ε-prod.
2. **S è annullabile** (si genera ε):
   costruiamo G' seguendo il procedimento del punto precedente e poi se S non compare a dx, si aggiunge S→ ε, altrimenti S'→ ε e S'→S.

Ricerca V annullabili: creo un insieme che contiene tutte le V che mi derivano direttamente ε, poi aggiungo le V che hanno a dx solo variabili annullabili. Ripeto finché l'insieme non smette di crescere.

Eliminare Produzioni Unarie

Creiamo **G'** eliminando le **ε-prod**., poi per ogni coppia di V: **A→*B**, aggiungo ai lati dx di A, tutti i lati dx di B.
Cancello poi le prod. unarie.
S può avere o **1 figlio** o **più figli**.

che contiene tutte le **prod. unarie dirette**.
Ad ogni passo aggiungo le coppie **X,Y** per cui ci sono già **X,Z** e **Z,Y** nell'insieme perché so che X→Z, e Z→Y, quindi X→Y.
Si continua finché possiamo aggiungere coppie.
(T_1 insieme di coppie X,Y per cui X→Y in 1 passo)

V improduttive e Inaccessibili

V è **produttiva** se da essa si può derivare una parola costituita interamente da **simboli terminali**.
V è **accessibile** se compare in qualche **forma sentenziale**.
Possiamo quindi **eliminare** le V improduttive e inaccessibili e creare una **G equivalente**.

Ricerca V produttive: creo insieme costituito da tutte le variabili direttamente produttive, poi aggiungo le V che hanno a dx terminali o V che già so essere produttive. Ripeto finché l'insieme non cresce più. Tutte le V rimaste fuori dall'insieme sono improduttive.
Ricerca V accessibili: creo insieme costituito da S che sappiamo essere accessibile, poi aggiungiamo tutte le V che compaiono nei lati dx di S.
Infine aggiungiamo le V che hanno a dx variabili che sappiamo essere accessibili. Ripetiamo finchè l'insieme smette di crescere.

Procedura di Riduzione:

1. Determinare le **V produttive**.
2. Eliminare le **V improduttive** e le prod. che hanno tali variabili.
3. Determinare **V accessibili.**
4. Eliminare le **V inaccessibili** e le prod. che contengono tali variabili.

- **Kleene**: G conterrà: **V**, **t**, e **P** della grammatica di partenza, più un **nuovo simbolo iniziale** S che

G con prod.: **X→YZ**, **X→t**, **S→ε** (se non compare a dx).
Negli Alberi di derivazione di G, le **foglie** sono **figli unici**, e i **nodi** sono un **albero binario completo**.

Ogni linguaggio non-Cont. è generabile da una G in forma normale di Chomsky seguendo i seguenti passaggi:

1. Eliminare **ε-prod.** e **prod. unarie.** Rimangono solo le prod. **X→t** e **X→ γ** (gamma) con **|γ|≥2** (almeno 2 lettere).
2. Dobbiamo ridurci al caso in cui **γ** contenga **solo V**.
   Per ogni **t** che compare **NON da solo** in qualche lato dx, si introduce una **nuova V**, e una produzione **V→t**.
   Si sostituiscono poi tutte le occorrenze del terminale con la nuova V, salvo che non sia l'intero lato dx.
3. Vogliamo che tutte le **V dx** siano **solo 2**. Raggruppiamo quindi coppie di V di lati dx con lunghezza >2, in una **nuova V** che me le produce.
   Es.: X→ABC | X→AZ Z→BC.

Forma Normale di Greibach:

G con prod.: **X→tγ** (γ= seq. di V), **S→ε** (se non compare a dx)
Ogni **derivazione** di una **parola** di **lunghezza n**, richiede esattamente **n passi.**
Ogni **L non-Cont**. è generato da una **G non-Cont. Greibach**.
Ogni G può essere messa in forma normale di Greibach.

Lemma Iterazione L NON-Contestuali:

Proprietà fondamentale dei L non-Cont.
Sia L non-Cont., esiste un **intero n** tale che ogni parola **w∈L**, di **lunghezza >n**, si può **fattorizzare** come
w= xuyvz con **uv≠ε** e $xu^kyv^kz \in L$ per ogni $k\geq 0$.
Iteriamo quindi parallelamente u e v senza uscire dal linguaggio.

Teorema di Rappresentazione

Un **L è non-Cont**. se e solo se esistono un intero **k>0**, un L regolare **R**, e un morfismo ***f***, tali che:
$L= f(D_k \cap R)$ dove D_k è il L di Dick.

Automa a Pila:

Teorema di Caratterizzazione L NON-Cont.

Vuole dimostrare che un L è non-Cont. se e solo se è riconosciuto da un $\mathcal{A}$ a pila.

- **G→$\mathcal{A}$ con 1 stato** (equivale al **parsing**)
  $\mathcal{A}$ sarà costituito da:
  - **Σ**, **N** (alfabeto pila), **S** (simbolo iniziale pila),
  - **δ(ε, A)** funz. trans. per le **ε-transizioni:** estraggo A dalla pila e scrivo sulla pila il lato dx di una produzione.
  - **δ(a, A)** funz. trans. per **terminale:**
  se c'è la produzione **A→a** non scrivo nulla sulla pila, altrimenti non è definita.
  - Ogni L non-Cont. è accettato per pila vuota da un $\mathcal{A}$ a pila con 1 stato.
- **$\mathcal{A}$ con 1 stato → G**
  G sarà composta da:
  - **V**= Alfabeto di Pila **Γ**
  - **t**= Alfabeto input **Σ**
  - **Simbolo iniziale**= simbolo iniziale della pila Z_0.
  - per ogni transizione da q0 a q0 con freccia "a,Z/γ", G avrà la produzione Z→aγ.
- **$\mathcal{A}$ generale → $\mathcal{A}$ con 1 stato**
  Il problema è che se $\mathcal{A}$ non fa il push, non posso registrare lo stato. L'$\mathcal{A}$ con 1 stato sarà costituito da celle che contengono:
  - **simbolo di pila** di $\mathcal{A}$
  - uno **stato arbitrario** (cioè tutti per il non-det.)
  - lo **stato arbitrario** contenuto nella **cella sottostante**
  - solo la **cella in cima** contiene lo **stato dell'$\mathcal{A}$ simulato**
  invece dello stato arbitrario.

Proprietà Chiusura L NON-Contestuali:

Classe L non-Cont. **chiusa per**: unione, concatenazione, Kleene e morfismi e Sostituzioni non-Cont.
NON chiusa per: intersezione e complemento.

- **Unione**: prendo le due G e suppongo non abbiano V in comune (sennò gli cambio nome). Costruisco poi la **nuova G** che conterrà: le **V**, **t**, e **P** delle G di partenza, più **due nuove prod. di S** che mi produce il simbolo iniziale della prima G (S') e della seconda G (S'').
- **Concatenazione:** G conterrà: **V**, **t**, e **P** delle G di partenza, più **S→S' S''**.
- **Kleene**: ... produrrà la **concatenazione** di se stesso con il vecchio simbolo iniziale $\mathbf{S \to S_1 S}$, e **S→ε.**
- **Intersezione:** se si fa ∩ di due L non-Cont., non è detto che venga fuori un L non-Cont.
  Se però sappiamo che uno dei due è **regolare** e l'altro è **non-Cont**. allora sappiamo con sicurezza che verrà fuori un **L non-Cont**.
  Infatti il **L regolare** è accettato da un $\mathcal{A}$ a stati finiti, mentre il **L non-Cont**. da un $\mathcal{A}$ a Pila con 1 stato.
  L'∩ tra questi due L sarà **accettata per stato finale e pila vuota** da un **$\mathcal{A}$ a Pila** che simula i due automi in parallelo.

Morfismo:

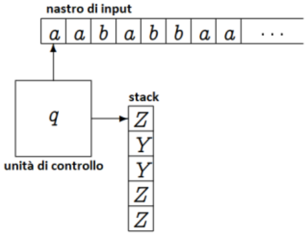
funzione che definita sull'alfabeto **Σ** mi restituisce **parole** sull'alfabeto **Γ**.
Il **morfismo** di **ε** è **ε.**
Il **morfismo** di una **parola** è uguale al morfismo di ogni lettera che compone la parola.
Un morfismo è completamente determinato dalle immagini delle lettere.

Il morfismo di un L non-Cont. è sempre un L non-Cont.:
creo **G'** che avrà:
- **V**= V e t della G originaria.
- **t**= lettere di **Γ.**
- **S**= S di G.
- **P**= quelle di **G**, con in aggiunta **a→f(a)** cioè una prod. che mi restituisce l'immagine dei terminali.

Sostituizioni:

funzione che definita sull'alfabeto **Σ** mi restituisce **insiemi di parole** sull'alfabeto **Γ**. (linguaggi)
Sostituzione di **ε** è il L che contiene solo ε.
Sostituzione di **linguaggio** è l'unione delle immagini delle parole che lo compongono.
Una **Sos. è non-Cont.** se l'immagine di ogni lettera è un **L non-Cont**.

Costruiamo $\mathcal{A}$' con un unico stato finale **f** che come primo passo scrive un **simbolo speciale #**



Utile per fare il **parsing** top-down, dove nello stack vengono scritte le varie predizioni che facciamo.

Ad ogni passo, a seconda dello stato in cui si trova la macchina, a seconda del simbolo letto dal nastro di input e dal simbolo prelevato dalla cima dello stack (**pop**), la macchina assumerà un nuovo stato, e verranno scritte (**push**) in cima allo stack 0 o più lettere.

Settupla:

- Q, q_0, Σ, F
- Γ (gamma), alfabeto di pila
- $Z_0 \in \Gamma$, simbolo iniziale della pila
- $\boldsymbol{\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma = \wp_F(Q \times \Gamma^*)}$,
  input una tripla: Stato, lettera dell'alfabeto di input (anche ε), e lettera dell'alfabeto di pila.
  output: insieme finito di coppie (nuovo-stato, parola sull'alfabeto di pila) (q, γ) gamma minuscol.
  Quindi l'automa si può portare nel nuovo stato q, facendo il push di γ sulla pila.

Descrizione automa: descrizione istantanea data dalla tripla (**stato**, **contenuto pila**, **contenuto nastro input**).
In base a questa istantanea possiamo sapere quale sarà la descrizione istantanea successiva.
Una relazione tra descrizioni istantanee si denota con $\vdash_{\mathcal{A}}$. Avendo per es. $A \vdash_{\mathcal{A}} B$, se prima la macchina era nello stato A, nell'istante successivo sarà nello stato B.
Una sequenza di descrizioni istantanee, ognuna consecutiva alla precedente, si denota con $D \vdash_{\mathcal{A}} D'$.
Quindi si ha $D \vdash_{\mathcal{A}} D'$, se esiste una computazione che porta l'automa dalla descrizione ist. D, a D'.

Metodi di accettazione: parola accettata per

- **stato finale:** se c'è una computazione che parte con la tripla: mia parola sul nastro di input, stato iniziale, e simbolo iniziale pila. Poi mano mano legge tutta la parola consumando il nastro di input e finisce in uno stato finale.
  $(q_0, w, Z_0) \vdash_{\mathcal{A}}^* (f, \varepsilon, \gamma)$ $\mathbf{f}$ stato finale, $\boldsymbol{\varepsilon}$ nastro input vuoto, e $\boldsymbol{\gamma}$ parola.
- **pila vuota:** se c'è una computazione che parte con la solita configurazione e termina in uno stato non-finale perché sia il nastro-input che la pila sono vuote.
  $(q_0, w, Z_0) \vdash_{\mathcal{A}}^* (p, \varepsilon, \varepsilon)$
- **stato finale e pila vuota:** se si verificano entrambe le condizioni precedenti: si termina in uno stato finale perché sono finiti il nastro-input e la pila.
  $(q_0, w, Z_0) \vdash_{\mathcal{A}}^* (f, \varepsilon, \varepsilon)$

L'insieme delle parole accattate per stato-finale/pila-vuota/entrambi, si dice linguaggio accettato dall'automa per stato-finale/pila-vuota/entrambi, e viene denotato con:
$\mathbf{L_F}(\mathcal{A})$ / $\mathbf{L_P}(\mathcal{A})$ / $\mathbf{L}(\mathcal{A})$.

Automa a Pila Deterministico:
Quando per ogni: **stato**, **simbolo input**, e **simbolo pila**, l'insieme $\boldsymbol{\delta(q, a, Z) \cup \delta(q, \varepsilon, Z)}$ contiene al più un elemento.
L'A può quindi fare **un solo movimento possibile** ad ogni passo.
Se A è **deterministico**, ogni descrizione ist. D ammette al più una descrizione ist. successiva.

Metodi di accettazione:
Dimostrare che le classi dei L riconosciuti dai seguenti metodi di accettazione coincidono.

1. **Pila vuota e Stato finale:** se L è accettato da un $\mathcal{A}_0$ per pila/stato, esiste un automa a pila $\mathcal{A}'$ tale che L è accettato da un altro automa anche per stato f. e da un altro ancora per pila v.
   all'inizio della pila, poi simula il funzionamento di $\mathcal{A}_0$. Se mentre si trova in uno stato finale di $\mathcal{A}_0$ estrae # dalla pila, passa in f senza scrivere nulla sulla pila. La pila si svuota se e solo se raggiunge lo stato finale quindi: $\mathbf{L}(\mathcal{A}) = \mathbf{L_P}(\mathcal{A}) = \mathbf{L_F}(\mathcal{A})$.
   Inoltre, ciò avviene se e solo se $\mathcal{A}_0$ col medesimo input raggiunge uno stato finale con pila vuota, quindi: $\mathbf{L} = \mathbf{L}(\mathcal{A}) = \mathbf{L_P}(\mathcal{A}) = \mathbf{L_F}(\mathcal{A})$.
2. **Pila vuota:** sia **L** accettato da $\mathcal{A}_0$ per pila vuota esiste $\mathcal{A}'$ tale che L è accettato da un altro automa per stato f. e da un altro per pila/stato.
   Ciò si ottiene andando a **sostituire in $\mathcal{A}_0$, F con Q**, quindi tutti gli stati sono finali.
3. **Stato Finale:** sia **L** accettato da $\mathcal{A}_0$ per stato f. esiste $\mathcal{A}'$ tale che L è accettato da un altro automa per pila v. e da un altro per pila/stato.
   Costruiamo $\mathcal{A}'$ aggiungendo a $\mathcal{A}_0$ lo stato e le istruzioni per **svuotare la pila** quando raggiunge uno stato finale: si aggiunge quindi un ulteriore **stato finale f**, e **ε-transizioni** che mi portano da ogni stato finale in f, per qualunque simbolo estratto dalla pila, senza scriverci nulla (nella pila).

Algoritmo CYK: O(n^3)
Moderatamente efficiente per risolvere il problema di Ricognizione, e (con opportune modifiche) anche quello di Parsing.
Ideale per le G in forma normale di Chomsky.

Siano **G** una grammatica in forma normale di **Chomsky** e **w** una parola, scriviamo w come **stringa di lettere** $w = a_1 a_2 \ldots a_n$,
Cerchiamo di **calcolare**, **per $\mathbf{0 \le i < j \le n}$**, tutte le **variabili** da cui si può derivare il **fattore** $a_i{+}1a_{i+2} \cdots a_j$, quindi vado a cercare per ogni fattore della mia parola, quali sono le variabili.
Per **i=0, j=n**, avrò le **variabili** da cui si può **derivare w**; quindi $w \in L(G)$ se e solo se tra esse il simbolo iniziale **S** è una di tali **variabili**.
L'output dell'algoritmo è una **matrice di ricognizione** N_{ij} associata a w. Ritorna TRUE se w appartiene a L,

		FALSE altrimenti. Per fare il **Parsing** basta fare il **procedimento inverso** di quello di ricognizione: se da X riesco a derivare a_{i+1} a_{i+2} ... a_j, vuol dire che ci sono le variabili Y e Z per cui $X \rightarrow YZ$, e poi da Y derivo $a_{i+1}...a_h$, e da Z derivo $a_{h+1}...a_j$.