

# MACCHINA ASTRATTA

insieme di strutture dati & algoritmi  
 x memorizzare  
 x eseguire programmi scritti in L

$\neq$  MACCHINA FISICA → una specifica implementaz.  
 fatta di circuiti logici & dispositivi elettronici

→ LINGUAGGIO MACCHINA il linguaggio compreso dal suo interprete L

## INTERPRETE

in L

per L

sistema di gestione della memoria  
 → CONTIENE

programma che realizza una funzione parziale

è un COMPONENTE della MACCHINA ASTRATTA

x eseguire istruzioni

ciclo FDE  
 e e x  
 t c e  
 c o c  
 n d e  
 e e

PRELEVARE      } un'istruzione  
 DECODIFICARE    }  
 ESEGUIRE

## IMPLEMENTAZIONE

### Firmware

- + veloce di SW
- [+ flessibile di HW]
- flessibile di SW

✗ x microprogrammi scritti con L molto basso

### Simulazione

### Software

- + molto flessibile
- velocità inferiore

Si basa su altre MA che dovranno essere implementate

### Realizzazione

### Hardware

- + estrema velocità di exec.
  - impl. costrutti di alto livello
  - poco flessibile
- ↳ edit al ling. ⇒ RIPROGETTAZ. della MA

x ling. BASSO LIVELLO

### COMPILATRICE PURA

- L<sub>1</sub>, L<sub>0</sub> → ① Tradotta ① istruzioni  
 ↳ in un gruppo di istr. di L<sub>0</sub>  
 ② Esecuzione

- perdita di info sulla struttura del prog. seq.
- + velocità esecutiva
- + istr. tradotte rivolta solo

### IMPLEMENTAZIONE INTERPRETATIVA PURA

- L<sub>1</sub> → ① decodifica  
 ↳ assegna ad ogni istr. L  
 ↳ un ins. di istr. di L<sub>0</sub>  
 ② eseguite

- scarsa efficienza
- I non genera codice → il cod. trad. è solo una descrizione delle operazioni
- + flessibilità → modifiche a run-time

# COMPILATORE

da  $L_1$ , a  $L_0$

realizzare una funzione:

$C_{L_1, L_0} : \text{Prog}^L \rightarrow \text{Prog}^{L_0}$  tale che,

dato un prog.  $p^L$  se:  $C_{L_1, L_0}(p^L) = p^{L_0}$

allora,  $\forall \text{Input} \in \mathbb{D}^S$

$$p^L(\text{Input}) = p^{L_0}(\text{Input})$$

M A C C H I N A M<sub>I</sub>

intermedia su  $M_I$

implementazione reale di una M.A.

$M_I$  scritta in  $L_I$

con  $C_{L_1, L_I}$  e  $T_{L_I}^{L_0}$

# DESCRIZIONE di un LINGUAGGIO

## GRAMMATICA

"Quali sono le frasi corrette?"

### Analisi LESSICALE

= TOKEN =

Sequenze di caratteri  
corrette dato un alfabeto  
↳ generando parole  
del linguaggio

### Analisi SINTATTICA

quali sequenze di token sono accettabili

↳ RELAZIONE TRA SEGNI

↪ Sottoinsieme di sequenze  
di parole

↳ costituendo → frasi di un  
linguaggio

↳ Costruire frasi  
dai TOKEN

## SEMANTICA

dare un significato ad ogni frase

## RELAZIONE TRA SEGNI E SIGNIFICATI

↪ = entità autonome, che esistono  
indipendenti dai segnificati usiamo x  
descrivere.

## PRAGMATICA

Come usare una frase

corretta e sensata in un

→ EVOLVE insieme al linguaggio

## E CONTESTO

in base al suo uso

## IMPLEMENTAZIONE

Come frasi "OPERATIVE" realizzano lo stato di  
cui stiamo parlando?

alfabeto = insieme finito di simboli

parola / stringa = sequenza corretta di simboli ∈ Alfabeto

$A^*$  = alfabeto numerabile

↳ STELLA di REENE  $A^* = \bigcup_{m \geq 0} A^m$  dove  $A^0 = \{\epsilon\}$  e  $A^m = A \cdot A^{m-1} = \{w \mid a \in A, w \in A^{m-1}\}$

[ GRAMMATICA LIBERA da CONTESTO ] = quadrupla  $(NT, T, R, S)$

insieme finito di non terminali

insieme finito di s. terminali

insieme delle produzioni

↳ Simbolo iniziale  
se NT

$V \rightarrow w$

dove  $V \in NT$  e  $w \in T^*$

1. Come si deriva una Stringa? 1. data una gram. libera e  $v, w$  due stringhe  
2A  $v \rightarrow w$  se  $w$  si ottiene da  $v$  sostituendo ad un NT  $V$  in  $v$  il corpo di una  
DER. IMMEDIATAMENTE produzione la cui testa sia  $V$

2B  $v \Rightarrow^* w$  se  $\exists$  una sequenza finita di der. immediate (2A)  
DERIVA che porta da  $v$  a  $w$ .

LINGUAGGIO GENERATO = (da una gram. libera G) è l'insieme

$\mathcal{L}(G) = \{ w \in T^* \mid S \Rightarrow^* w \} :=$  l'insieme delle stringhe finite ottenibili dal simbolo iniz.  $S$  con una o più derivazioni sui  $T$

## ALBERO di DERIVAZIONE o di PARSING

ogni nodo è etichettato con un simbolo  $NT \cup T \cup \epsilon$

la radice è etichettata con  $S$

ogni nodo interno è etichettato con  $NT$

Se il nodo  $n$  ha etichetta

$A$  in  $NT$  e le etichette dei figli  $\{x_i \in NT \cup T, \forall i \in [1, k]\}$

$\hookrightarrow \exists$  produzione  $A \rightarrow x_1 \dots x_k \mid A \in R$

ha etichetta  $\{\epsilon\} \Rightarrow n$  è una foglia

è figlio unico

è una prod. di  $R$  dato  $A$  suo padre  $A \rightarrow \epsilon$

È COMPLETO?

$\Leftrightarrow$  ogni nodo foglia è etichettato su  $T \cup \epsilon$

CORRISPONDENZA BIUNIVOCATA

## derivazioni canoniche

Si dice SINISTRA / DESTRA se ad ogni passo viene riscritto il NT più a SX / DX della stringa

$\downarrow$   
sx / dx generano lo stesso albero.

: date due DER. CANONICHE  $Sx$  per uno stesso NT,

ora se + al NT più a SX

vengono applicate due produzioni diverse.  $\Rightarrow$  alberi diversi

## GRAMMATICA AMBIGUA

$\Leftrightarrow \exists$  una stringa  $v \in \mathcal{L}(G)$  che ammette 2+ derivazioni dal simbolo  $S$

es: Grammatica delle espressioni

LINGUAGGIO (quando le G che lo generano sono ambigue)  
AMBIGUO

(1): Eliminare produzioni  $\epsilon$   
 ~~$S \rightarrow \epsilon$~~

?

Come  
DISAMBIGUARE

- decidendo precedenze delle deriv.
- manipolando la grammatica

(2): Eliminare prod. unitarie  
 $S \rightarrow aA \mid Aa \Rightarrow S \rightarrow aAa$   
 $A \rightarrow aAa \mid AaA$

(3): Eliminare simboli inutili

[che non danno modo di terminare]  
l'ordine  
è IMPORTANTE  
[che non vengono usati] ...

(4): fattorizzazione a SX

$A \rightarrow aBbC \mid aBd$

$\Rightarrow A \rightarrow aBA'$

$A' \rightarrow bC \mid d$

\* ridurre gli look-ahead

(4): Eliminare ricorrenze sinistre  
 $A \rightarrow A\alpha \mid A\beta \mid r \mid s$  con  $\alpha, \beta \in (T \cup NT)^*$   
 $\Rightarrow A \rightarrow rA' \mid sA'$   
 $A' \rightarrow \alpha A' \mid \beta A' \mid \epsilon$

## ALBERO di SINTASSI ASTRATTA

particolare ALBERO di DERIVAZIONE  
che soddisfa i vincoli contestuali di un linguaggio  
in cui appiamo solo i terminali T

## Zucchero sintattico

→ Simboli x chiarire solo la derivazione di una stringa

parametri formali  
della dichiarazione

Numero e Tipo di  
parametri attuali di una  
chiamata di procedura

identificatore deve essere  
dichiarato prima dell'uso

prima di usare una variabile  
deve esserci un assegnamento  
su di esse.

## GRAMMATICA

dipendente dal  
CONTESTO

ha produzioni del tipo  $uAv \Rightarrow uwv$

con  $u, v, w \in \{T, N\}$  e  $A \in NT$

La produzione  $S \Rightarrow E$  è ammessa solo se  $S$  non compare a destra di nessun altro p.

## GRAMMATICA MONOTOMA

la lunghezza del segmento  
della sequenza di simboli  
a SX = lunghezza di simboli  
a DX della produzione

## statica

o SINTASSI CONTESTUALE  
è descrivibile con Vincoli contestuali

→ verificabili staticamente  
all'interno del testo del  
programma

## SEMANTICA

## dinamica

è descrivibile con Vincoli contestuali

verificabili solo al momento  
dell'esecuzione del  
programma

## definita da S. di un LINGUAGGIO

Ricerca dell'esattezza

+ complesso della Sintassi

× mediare tra 2 istanze

flessibilità

× togliere le ambiguità  
per l'utente

× lasciare spazio  
all'implementazione

# Passi di un COMPILATORE

## 1 Analisi LESSICALE - scanning

- leggere sequenzialmente simboli di ingresso di cui è composto il pro.
- raggruppare i simboli in token

## 2 Analisi SINTATTICA - parsing

- costruire un albero di derivazione per la lista di token creata.
  - ogni foglia dell'albero è composta da un token
  - leggendo i token <sup>(foglie)</sup> da  $Sx \rightarrow Dx$  si vede una ~~FRASE~~ LEGALE del linguaggio
- Se non si riesce a costruire la stringa di token data non è corretta x la grammatica del linguaggio

## 3 Analisi SEMANTICA

- controllo dell'albero di derivazione ai vincoli contestuali
  - vengono inserite informazioni tipo, luogo delle dichiarazione... ottenute dai controlli

## 4 Generazione Forma Intermedia

- visitando l'albero si genera un CODICE INTERMEDIO
  - EASY TO PRODUCE (operazioni semplici)
  - indipendente da Architettura e Sorgente
  - EASY TO TRANSLATE (segue la struttura dell'albero sintattico)

## 5 Ottimizzazione Forma Intermedia

- rimosso il codice inutile
- espansioni in-line di chiamate di funzione
- fattorizzazioni di sotto espressioni → x non calcolare + volte le stesse op.
- togliere dai cicli le sotto espressioni che non variano.

## 6 Generazione del Codice

- generato CODICE OGGETTO x una specifica architettura
  - c arreguazione dei registri
  - c ottimizzazioni specifiche per l'architettura e il codice oggetto

~~METODI  
formali~~

## ~~SEMANTICHE OPERAZIONALI~~

specificava il comportamento della M.A.  
(con riferimento a formalismi di basso liv.)

definendone l'INTERPRETE

## ~~SEMANTICHE DENOTAZIONALI~~

applicazione ai linguaggi di p.  
di tecniche x la semantica  
del linguaggio logico - matematico,

- ↪ una FUNZIONE  
determina il significato di un programma
- ↪ esprime il comportamento I/O  
del programma

# ANALISI LESSICALE — linguaggi regolari

## Analizzatore lessicale

riconoscere nella Stringa in ingresso alcuni GRUPPI di CARATTERI

↓

corrispondenti a categorie sintattiche

Stringa trasformata  
in TOKEN

→ analizzatore sintattico ...

information astratta che rappresenta una → coppia (NOME, VALORE)

Stringa del testo in ingresso

simbolo astratto

[categoria sintattica]

sequenza di simboli  
del testo in ingresso

descrizione generale della forma  
che le sequenze possono assumere

specificate da

PATTERN

con ESPRESSIONI REGOLARI

↪ lessema

= stringa corrispondente ad ●

1.  $\epsilon$  è un'espr. regolare;  $L[\epsilon] = \{\epsilon\}$

2.  $a \in A$ ,  $a$  è un'espressione regolare,  $L[a] = \{a\}$

↪ Linguaggi ASSOCIAZIONI

3. Se  $r$  è un'espr. regolare,  $\Rightarrow (r)$  è un'espr. regolare,  $L[(r)] = L[r]$

4. Se  $r$  e  $s$  sono espr. regolari,  $(r) | (s)$  è un'espr. regolare,  $L[(r) | (s)] = L[r] \cup L[s]$

5. Se  $r$  e  $s$  sono espr. regolari,  $(r) \cdot (s)$  è un'espr. regolare,  $L[(r) \cdot (s)] = L[r] \cdot L[s]$

6. Se  $r$  è un'espr. regolare,  $(r)^*$  è un'espr. regolare;  $L[(r)^*] = L[r]^*$

7. Nient'altro è un'espr. regolare.

LINGUAGGI REGOLARI se  $L$  è vuoto  $\vee$  è espr. regolare  $S \mid L = L[S]$

Tutti i linguaggi finiti sono linguaggi regolari.

Esistono linguaggi infiniti che sono regolari.

## EQUIVALENZA TRA 2 espr. regolari

Se generano lo stesso linguaggio:  $r | s = s | r$ ;  $r^{**} = r^*$ ;  $r(\delta t) = (r \delta)t$

## Definizione REGOLARE

- lista di definizioni di simboli a cui ad ogni simbolo è associato un'espressione regolare.
- utile a esprimere espr. regolari complesse.  
↳ si divide in sottoespressioni a cui è associato un simbolo

# NFA automa finito non deterministico

è una quintupla:  $(\Sigma, Q, \delta, q_0, F)$

insieme finito di simboli

insieme finito di stati

insieme di funzioni definite  
es:  $\delta: Q \times (\Sigma \cup \{\epsilon\}) \rightarrow P(Q)$

insieme finito di stati finali  $\subseteq Q$

stato iniziale  $\in Q$

DIAGRAMMA di  
TRANSIZIONE

$\times$  descrivere  
in modo compatto  
ed efficace NFA

grafo orientato i cui nodi rappresentano gli stati

gli archi etichettati  
con i simboli dell'alfabeto  $\Sigma$   $\rightarrow$  transizioni possibili  
dell'input tra gli stati

## LINGUAGGIO ACCETTATO

da un NFA è l'insieme  $L(N) = \{x \in \Sigma^* \mid N \text{ accetta } x\}$

# DFA automa finito deterministico

è una quintupla  $(\Sigma, Q, \delta, q_0, F)$

insieme finito di simboli

insieme finito di stati

insieme di funzioni definite

$$\delta: Q \times \Sigma \rightarrow q$$

insieme finito di stati finali  $\subseteq Q$

stato iniziale

E' un particolare caso di NFA  $\rightarrow$  non si hanno archi etichettati con  $\epsilon$

nella f. di transizione si ha  
 $\delta(q, a)$  e SEMPRE esattamente  
costituita da 1 SOLO STATO

NFA

- uno stato di  $M_N$  è come un insieme di stati di  $N$

Se  $N$ , partendo dal suo  $S$  e consumando l'input  $a_1, \dots, a_k$ , si trova in uno stato  $q_1, \dots, q_k$   
 $\Rightarrow M$  si trova in  $\{q_1, \dots, q_k\}$  (stato)

Il DFA equivalente può avere fino a  $2^n$  stati

equivalente

DFA

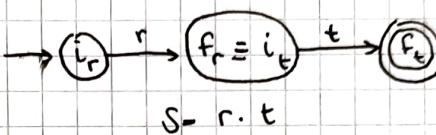
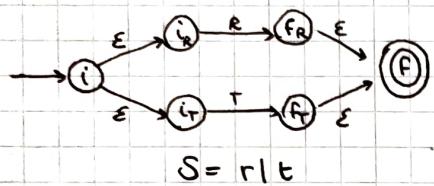
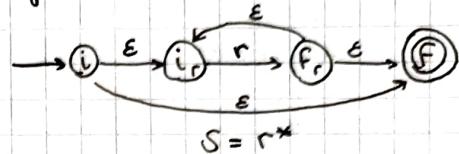
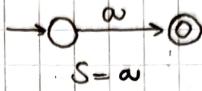
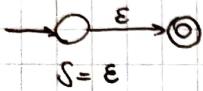
Thm

CLASSE DEI LINGUAGGI  
RICONOSCIUTI DA UN NFA = CLASSE dei LINGUAGGI  
RICONOSCIUTI DA UN DFA

Sia  $N = (\Sigma, Q, \delta, q_0, F)$  un NFA e  $M_N$  l'automa ottenuto con la costruzione per sottoinsiemi, allora  $M_N$  è un DFA e si ha  $\mathcal{L}(N) = \mathcal{L}(M_N)$

da EXPRESSIONE REGOLARE [in modo che il L riconosciuto da NFA sia lo stesso L associato a regex]

costruire per induzione in base agli schemi



Ricorda che  $a^* = aa^*$

## Grammatica REGOLARE

una grammatica libera è regolare se ogni produzione è della forma

$V \rightarrow aW$  oppure  $V \rightarrow a$

dove  $V, W \in NT$  e  $a \in T$ .

! Per il simbolo iniziale  $S$  è ammesso anche  $S \rightarrow \epsilon$ .

associarla ad un equivalente NFA

passando attraverso ad una EXPRESSIONE REGOLARE

da DFA →  
a GRAM. REGOLARE

Sia  $M = (\Sigma, Q, \delta, q_0, F)$  il DFA assegnato.

La grammatica  $G_M = (Q, \Sigma, R, q_0)$  ha:

1. come NT, gli stati di  $M$ ;
2. come  $T$ , l'alfabeto di  $M$ ;
3. come  $S$ , lo stato iniziale  $q_0$  di  $M$ ;
4. come produzione  $R$ :

a)  $\forall \delta(q_i, a) = q_j$  di  $M$

la produzione  $q_i \rightarrow aq_j \in R$ .

Se  $q_j \in F$ , allora  $q_i \rightarrow a \in R$ .

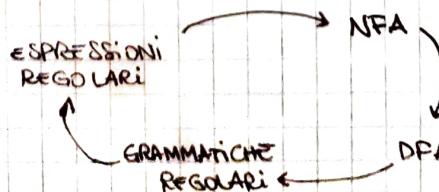
b) Se  $q_0$  di  $M \equiv q_f$  ( $q_0 \in F$ )

allora  $q_0 \rightarrow \epsilon \in R$

da GRAMM. REGOLARE →  
a EXPRESSIONE REGOLARE

si crea un'espressione regolare che riconosce il linguaggio generato dalla grammatica.

Riassumendo



tutti questi formalismi sono equivalenti e descrivono le classi dei linguaggi regolari

# STATI EQUIVALENTI di un DFA

Una stringa  $\alpha$  distingue due stati  $q_0, q_1$   
Se il cammino che parte in  $q_0$  e consuma  $\alpha$  arriva in uno Stato finale @  
e non finale

il cammino che parte in  $q_1$  e consuma  $\alpha$  arriva in uno stato non finale finale

2 stati sono equivalenti se non sono distinguibili

# STATI DISTINGUIBILI di un DFA

## TABELLA A SCALA x CLASSI di EQUIVALENZA di STATI di un DFA

1. Si crea una tavella con tutte le coppie di stati

2. Si marca ogni coppia: finale/non finale

3. Fino a quando c'è almeno 1 marcatura  
allora ① scorri tutte le coppie non marcate,  
② prova tutte le mosse possibili [alfabeto]

↳ se almeno 1 porta la coppia di stati su una coppia già marcata  
allora marca anche quelle.

> ORA POSSO Sviluppare un AUTOMA MINIMO

↳ perché gli stati delle coppie non marcate risultano essere equivalenti

# LEX

Software in grado di generare uno Scanner/analizzatore lessicale  
INPUT file.l contiene una serie di definizioni regolari  
e una serie di azioni corrispondenti

OUTPUT programma in C → realizza l'automa riconoscitore  
e associa ad ogni istanza di una def. la relativa azione

In Breve: analizzatore lessicale che prende in INPUT uno stream di caratteri  
e compie delle azioni ogni volta che incontra una determinata  
combinazione di caratteri

# PUMPING

## LEMMA theorem

Se  $\mathcal{L}$  è un linguaggio regolare,  
allora  $\exists N > 0 \mid \forall z \in \mathcal{L}$  con  $|z| \geq N$ ,  $\exists u, v, w$  tale che:

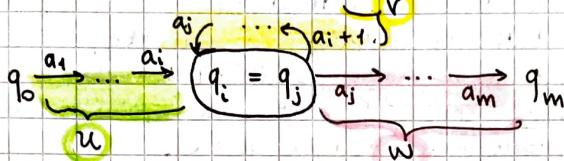
- >  $z = uvw$
- >  $|u|v| \leq N$
- >  $|v| \geq 1$
- >  $\forall k \geq 0, uv^k w \in \mathcal{L}$

Inoltre  $N$  è numero di stati del DFA minimo che accetta  $\mathcal{L}$ .

- dato  $N = |Q_M|$  : numero stati di  $M$  con  $M$  un DFA minimo che accetta  $\mathcal{L}$
- dato  $z = a_1 a_2 \dots a_m \in \mathcal{L}$  con  $m \geq N$  : stringa e lunghezza stringa  $[m]$
- Dunque  $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \dots \xrightarrow{a_m} q_m \in F$

in particolare il "tratto"  $0-m$  è dato da  $m+1$  stati con  $m+1 > N$   
 $\Rightarrow \exists i, j (i \neq j) \mid q_i = q_j$

Dunque:



- Poiché  $i \neq j$ , allora  $v = a_{i+1} \dots a_j \mid |v| \geq 1$  : il loop ha almeno 1 modo
- Inoltre  $|u| \leq N$ , poiché  $i$  e  $j$  sono scelti tra i primi  $N+1$  stati,  
 infatti se  $m$  fosse molto grande potrebbero esserci + cicli, ma noi prendiamo il 1°!

$$\left. \begin{array}{l} uv^0 w = uw \in \mathcal{L} \\ uv^1 w = uvw = z \in \mathcal{L} \\ uv^2 w = urvw \in \mathcal{L} \end{array} \right\} \Rightarrow \forall k \geq 0, uv^k w \in \mathcal{L}$$

perché il ciclo  $v$ , può essere percorso un numero arbitrario di volte.

NB per dimostrare che un linguaggio non è regolare, uso questo lemma  
a ROVESCI :

# PROPRIETÀ CHIUSURA ling. Regolari

UNIONE  $\cup$  → ling. reg. generati da un'espr. regolare (ovvio)

INTERSEZIONE  $\cap$  →  $L[M_1] \cap L[M_2] = L[\overline{M_1}] \cup L[\overline{M_2}]$

COMPLEMENTAZIONE →  $L[N]$  DFA  $N = (\Sigma, Q, \delta, q_0, F)$   
inversione stati finali e non finali  $L[\overline{N}]$  DFA  $\overline{N} = (\Sigma, Q, \delta, \overline{q_0}, Q \setminus F)$

CONCATENAZIONE  $\circ$  → ovvio x ling. reg generati da esp. reg.

RIPETIZIONE  $^*$  → ovvio x ling. reg generati da esp. reg.

DIFERENZA  $-$  → x essere chiusi per intersezione

## ANALISI SINTATICA

> fase del processo di compilazione

> descrive quali sequenze di Token sono legali per il linguaggio

### Parser

o ANALIZZATORE SINTATICO

costruito da una  
Grammatica LIBERTI

> riconoscitore di ling. libero

> programma (basato su PDA) che usa:

Se { INPUT x decidere le produzioni / TOKEN da usare  
OUTPUT → costruisce un albero di derivazione (se esiste)  
Token e d

a u t o m a

## PDA

- pila  
non deterministica

↳ x te puoi raggiungere + STATI a partire da 1 con la stessa transizione

> settuple  $(\Sigma, Q, \Gamma, \delta, q_0, \perp, F)$  dove:

$\Sigma$  alfabeto finito [simboli in input]

$Q$  insieme finito di Stati

$\Gamma$  (gamma) insieme finito dei simboli della pila

$\delta$  funzione di transizione che prende

· uno stato corrente

· un simbolo in INPUT  $\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow P_{fin}$

· un simbolo della pila

Return: insieme di coppie Stato

$q_0 \in Q$  stato iniziale

$\perp \in \Gamma$  simbolo iniziale sulla pila

$F \subseteq Q$  insieme degli Stati finali

### CONFIGURAZIONE

o descrizione istantanea

> tripla  $(q, w, \beta)$

Stato  $q \in Q$  |  $\epsilon \in \Gamma^*$  simbolo in pila  
Simbolo input  $w \in \Sigma^*$

### MOSSA di un passo

> transizione tra 2 configurazioni:

$(q_1, aw, x\beta) \rightarrow (q_2, w, \alpha\beta)$

tc  $q_1, q_2 \in Q$   $x, \beta \in \Gamma^*$   
 $aw \in \Sigma^*$

## LINGUAGGIO

da PDA

### ACCETTATO

non equivalenti  
dato uno stesso PDA;

$L[P] \neq M[P]$

ma la loro classe non cambia

se acc. x i 2 casi  $\Rightarrow$  (ling. liberi) =  
da controfro

x STATO FINALE  $L[P] = \{w \in \Sigma^* \mid (q_0, w, \perp) \xrightarrow{*} (q, \epsilon, a), q \in F\}$

stringhe che tramite una sequenza di mosse

accettano la stringa mentre sono in uno Stato finale

x PILA VUOTA  $c[P] = \{w \in \Sigma^* \mid (q_0, w, \perp) \xrightarrow{*} (q, \epsilon, \epsilon), q \in F\}$

stringhe

accettate sottraendo la pila

da GRAMMATICA  
a PDA

data una grammatica  $G = (NT, T, R, S)$   
è possibile costruire un PDA  $P = (T, \{q\}, NT, S, q, \emptyset)$   
EQUIVALENTE con 1 solo stato  $q$ :

- $\forall r = (V \rightarrow W) \in R \mid V \in NT, W \in (T \cup NT)$   
 $\exists \delta : (q, \epsilon, V) \rightarrow (q, \epsilon, W)$  mossa di espansione
- $\forall a \in T$   
 $\exists \delta : (q, a, a) \rightarrow (q, \epsilon)$  mossa di match

Il riconoscimento del PDA è di tipo top-down  
e fortemente non deterministico

Dato che (1) è possibile ottenere una gram. libera da un PDA  
(2) classe dei linguaggi liberi = classe dei ling. generati da G.L.

Dunque la classe dei linguaggi accettati da PDA  
è quella dei linguaggi liberi

PROPRIETÀ  
CHIUSURA  
ling. liberi

UNIONE  $U \rightarrow$  aggiungendo una produzione  $S \rightarrow S_1 | S_2$

CONCATENAZIONE  $\cdot \rightarrow$  aggiungendo una produzione  $S \rightarrow S_1 S_2$

RIPETIZIONE  $\rightarrow$  aggiungendo una produzione  $S \rightarrow \epsilon | S, S$

NB ling. libero  $\cap$  ling. regolare  $\Rightarrow$  non necessariamente libero

PUMPING  
THEOREM

Se un linguaggio  $L$  è libero,  
allora  $\exists N > 0 \mid \forall z \in L$  con  $|z| \geq N$

- $\exists u, v, w, x, y$  t.c.
1.  $z = uvwxy$
  2.  $|vwx| \leq N$
  3.  $|vx| \geq 1$
  4.  $\forall k \geq 0, uv^k w^k x^k y \in L$

a ROVERCIO se  $w \neq \emptyset \Rightarrow L$  non è libero

1.  $\forall N > 0 \ \exists w \in L \mid w$  scomposto in 5 sottostringhe  $uvwxy$  t.c:  
vincolo 2  $|vwx| \leq N$   
vincolo 3  $|vx| \geq 1$

2. Allora  $\forall k \geq 0, uv^k w^k x^k y \in L$   
Dunque il linguaggio non è libero.

## Pumping Thm

- Sia  $G$  una grammatica libera  $= (N, T, R, S) \mid L = L(G)$
- Sia  $b = \max \{ |w| \mid A \rightarrow w \in R\}$ : max fattore di ramificazione di un albero di derivazione  
►  $b \geq 2$ , altrimenti la grammatica sarebbe banale. (01)
- Un albero di altezza  $h$  e fattore di ramificazione  $b$ , ha al più  $b^h$  foglie.
- Fissato  $N = b^{|\text{INT}|+1}$ , dunque  $N > b^{|\text{INT}|}$  per O1  
Si ha che ogni albero di derivazione per  $z$ , con  $|z| \geq N$  deve avere altezza almeno di  $|\text{INT}| + 1$ .
- dato un qualsiasi  $z \in L$  con  $|z| \geq N$   
considero il suo albero di derivazione (minimi nodi) (H1)  
 $|z| \geq N \Rightarrow$  albero con altezza  $|\text{INT}| + 1$   
 $\Rightarrow$  è cammino da radice  $S$  ad una foglia con almeno  $|\text{INT}| + 2$  nodi  
 $\Rightarrow$  cammino attraverso  $|\text{INT}| + 1$  nodi interni etichettati con NT  
[la foglia è etichettata da  $T$ ]  
 $\Rightarrow$  almeno un NT si ripete nel cammino
- Sapendo questo, avremo che:  
 $S \Rightarrow^* z$  puo' essere diviso come  
 $S \Rightarrow^* uA^k y \Rightarrow^* u v A^k x y \Rightarrow^* u v w x y$   
Parte di derivazione ripetibile  $k$  volte con  $k \geq 0$

con i seguenti vincoli: (li verifichiamo ora)

VINCOLO 3  $|vx| \geq 1$  ovvio: se  $v = \epsilon = x$ , allora l'albero per  $k=0$  genererebbe ancora  $z$  e avrebbe meno nodi, contraddicendo H1

VINCOLO 2  $|vwx|^k \leq N$  ovvio: il cammino da  $A$  alla foglia è di lunghezza  $\leq |\text{INT}| + 1$   
 $\Rightarrow$  la  $A$  in alto non può generare parole + lunghe di  $b^{|\text{INT}|+1} = N$  (per O1)

## CLASSIFICAZIONE di Chomsky

gram. illimitate  $\rightarrow$  lingua. SEMI DECIDIBILI  $\rightarrow$  mac. Turing

gram. DIPENDENTI DA CONTESTO  $\rightarrow$  ling. CONTESTUALI  $\rightarrow$  aut. LIMITATO LINEARMENTE

gram. LIBERE DA CONTESTO  $\rightarrow$  ling. LIBERI  $\rightarrow$  automa a pila

gram. REGOLARI  $\rightarrow$  automa a stati finiti

automa

# DPDA

a pila  
deterministico

Un PDA  $P = (\Sigma, Q, \Gamma, \delta, q_0, \perp, F)$  è deterministico se:

1.  $\forall q \in Q, \forall z \in \Gamma, \text{ se } \delta(q, \varepsilon, z) \neq \emptyset \text{ allora } \delta(q, a, z) = \emptyset \quad \forall a \in \Sigma$   
 $\hookrightarrow$  se c'è una produzione  $\varepsilon \Rightarrow$  non ci deve essere nessun'altra p...
2.  $\forall q \in Q, \forall z \in \Gamma, \forall a \in \Sigma \cup \{\varepsilon\}, |\delta(q, a, z)| \leq 1$   
 $\hookrightarrow$  ci deve essere al massimo 1 mossa per ogni tripla

un linguaggio è libero deterministico se è accettato per stato finale da un DPDA

## PREFIX property

Dato un linguaggio  $L$ , questo gode della PREFIX PROPERTY se NON esistono 2 stringhe  $\epsilon \in L$  | una sia prefisso dell'altra

$\hookrightarrow$  un linguaggio gode della PREFIX PROPERTY  $\Leftrightarrow$  è riconosciuto da un DPDA

$L = \{a^n b^m \mid n \leq m\}$  NON GODE  $\rightarrow$  ab è prefisso di aab

$L = \{a^n b^m a^n \mid n, m \geq 1\}$  GODE della proprietà

## LINGUAGGIO

## LIBERO

## DETERMINISTICO

## \* Parser

a discesa ricorsiva

1. si associa una funzione ad ogni produzione di ogni simbolo NT

> end mark  $\delta$  x riconoscerlo  $\rightarrow$  DPDA riesce a rilevare i prefissi

> non è mai ambiguo

\* riconoscere se una parte dell'INPUT deriva dalla produzione associata

DETERMINISTICO  
lookahead

vs

NON DETERMINISTICO  
enumerazione

x decidere quale è l'unica opzione che permette di riconoscere NPOT

davanti una scelta, tenta tutte le opzioni

Come si Costruisce?  $\rightarrow$  2 TECNICHE

1 TOP DOWN

ricostruiscono una derivazione - = espandono tutte le foglie finché non fanno S sinistra (leftmost) dal S tutte etichettate con  $\square$  T

$\hookrightarrow$  !  $\rightarrow$  gr. con RICORSIONE a sx  $\rightarrow$  gramm. AMBIGUE

2 BOTTOM UP

ricostruiscono una derivazione - = partendo dalle foglie cercano di capire Destra (rightmost) cercando di quali sono le produzioni che le hanno generate RIDURA al S

$\hookrightarrow$  !  $\rightarrow$  grammatiche AMBIGUE

# FIRST

insieme di T che si trovano in PRIMA POSIZIONE in una stringa derivata da  $\alpha$ :

$$\text{first}(\alpha) = \{x \in T \mid \alpha \xrightarrow{*} x\beta \text{ per qualche } \beta \in (T \cup NT)^*\} \cup \{\$ \}$$

## Algoritmo per calcolarli

- se un NT derivabile in  $\epsilon$   $\Rightarrow \epsilon \in \text{first}$

- simboli T in testa alla stringhe

$$S \rightarrow cAB \mid AB \mid DA$$

$$\begin{aligned} \Rightarrow \text{First}(cAB) &= c \\ \text{First}(AB) &= \text{First}(A) \\ \text{First}(DA) &= \text{First}(D) \end{aligned} \quad \left. \begin{array}{l} \text{first}(S) \\ \text{first}(A) \end{array} \right\}$$

insieme dei T che possono comparire a DESTRA di un NT, in una produzione

$$\text{follow}(\alpha) = \{x \in T \mid S \xrightarrow{*} \alpha A x \beta \text{ per qualche } \alpha, \beta \in (T \cup NT)^*\}$$

$$\cup \{ \$ \mid \text{se } S \xrightarrow{*} \alpha A \$ \}$$

- $\$ \in \text{follow}(S)$

- se in una produzione un NT è seguito da un altro NT

$$\Rightarrow \text{Follow}(\text{primo}) = \text{follow}(\text{secondo})$$

- produzione termina con 1+ NT [etc]

$$NT \rightarrow \dots CB$$

$$NT \rightarrow \dots P$$

$$\text{Follow}(B/D) = \text{follow}(NT)$$

- B annullabile? — sì  $\Rightarrow \text{follow}(C) = \text{follow}(NT)$

- $\$$  nei NT alla fine di una produzione di  $S \rightarrow \dots$

- 2 NT vicini: se 1' espandibile (con NT)

$$\Rightarrow \text{il suo NT (interno) eredita quelli del 2'}$$

$$S \rightarrow AB \quad \text{follow}(S) = \$ + \text{follow}(A) + \text{follow}(B)$$

$$A \rightarrow CC \quad \text{follow}(C) = \text{follow}(B)$$

# TABELLA di Parsing LL(1)

Righe: NT della grammatica

Colonne: T della grammatica + { $\$$ }

Cafelle  $\rightarrow$  = produzioni che si hanno dall'espansione della produzione A e dal suo simbolo N.

$$M(A, a)$$

$$\begin{aligned} S &\rightarrow aAB \mid bS \\ A &\rightarrow a \\ B &\rightarrow b \end{aligned} \quad \left. \begin{array}{l} G \\ \uparrow \end{array} \right\}$$

1. guardo first e follow

	FIRST	follow
S	a, b	\$
A	a	b
B	b	\$

2. Se le produzioni  $\neq \epsilon$   $\Rightarrow$  le scrivo nelle colonne del tutti i suoi first  
 Se la produzione =  $\epsilon$   $\Rightarrow$  la metto nella colonna del follow del NT analizzato

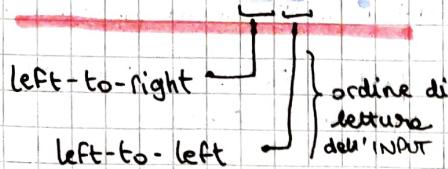
S	$\epsilon$	$a$	$b$	$\$$
S	$S \rightarrow aAB$		$S \rightarrow bS$	
A	$A \rightarrow a$			
B		$B \rightarrow b$		

nel nost caso non ci sono  $\{ \epsilon \} \in \text{First}(NT)$

# GRAMMATICA

## CLASSE LL(1)

=> se la sua tabella di parsing contiene in ogni cella al più una produzione.



=> nella grammatica  $G$ , la coppia di produzioni con la stessa testa  $A \rightarrow \alpha | \beta$  si ha:  
 $\rightarrow \text{First}(\alpha) \cap \text{First}(\beta) = \emptyset$

- a. Se  $\epsilon \in \text{First}(\alpha)$  allora  $\text{First}(\beta) \cap \text{Follow}(\alpha) = \emptyset$   
b. Se  $\epsilon \in \text{First}(\beta)$  allora  $\text{First}(\alpha) \cap \text{Follow}(\beta) = \emptyset$

! Ogni LINGUAGGIO REGOLARE è di classe LL(1).

→ Parser <sub>BOTTOM UP</sub> = Parser LR o parser Shift-reduce

( PDA con 2 azioni ) → SHIFT :  $T$  spostato dall'INPUT in cima alla pila  
 → REDUCE : insieme di simboli in cima alla pila ( $\equiv$  parte dx di una produzione) vengono eliminati e sostituiti con il NT a sx (che genera quelle produzione)

e generando → un albero di derivazione destro

se stringa in INPUT è associata al parser

! CONFLITTI → Shift/Reduce : quando si può fare entrambe le azioni  
 "Look-ahead" → Reduce/Reduce : la cima della pila  $\equiv$  parte dx di + produzioni dipendente dai simboli in INPUT

PREFISSO  
viable

Stringa  $w \in (T \cup NT)^*$  che può presentarsi in cima alla pila di un parser LR

Se per la grammatica  $G$  è prod. destra bc  $S \rightarrow^* \delta A y \rightarrow \delta \alpha \beta y = \gamma \beta y$

⇒ la stringa verrà accettata

NFA

dato da

1. uno stato  $s_0$  item  $LR(0)$  di  $G$  aumentata
2.  $s_0 \rightarrow s_1$  è lo stato iniziale
3. da  $A \rightarrow a \cdot x b$  c'è transizione ad  $A \rightarrow a x \cdot b$  con  $x \in T \cup NT$
4. da  $A \rightarrow a x b$  & produzione  $x \rightarrow y$  c'è una  $\epsilon$ -transizione su  $x \rightarrow y$
5. non serve def. gli stati finali.

DFA

si ottiene × la costruzione × sottoinsiemi dell'NFA

## item LR(0)

generati creando  
un item  $\in$  posizione  
di ogni produzione di  $G$

per una grammatica  $G$  è una sua produzione  
in cui è indicata esplicitamente una posizione nella sua parte dx  
x es: con un punto.

se l'item ha il punto in ultima posizione ( $A \rightarrow ab.$ )  
allora indica la presenza di una MANGIA

=  $T$  in cui ci si accorge di  
poter fare una 'Reduce'

## tavella LR(0)

è una matrice bidimensionale dove:

- Righe indizzate  $\times$  gli stati dell'automa LR(0)
- Colonne indizzate  $\times$  simboli ( $T \cup T^*$ ) di  $G$   $\cup \{\$\}$

l'elemento  $[S, X]$  della matrice

$\hookrightarrow$  l'azione da eseguire quando il parser LR  
si trova nello stato  $S$  e in cima alla pila c'è  $X$ .

$\Rightarrow$  se la tabella non presenta conflitti  
allora la grammatica  $G$  è di classe LR(0)

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow (S)S \\ S &\rightarrow E \end{aligned}$$

look-ahead  
 $\sim$  tavelle LR(0)

$\hookrightarrow$  Reduce inserite solo in  $M[a, x]$  con  $x \in T \Leftrightarrow x \in \text{follow}(A)$   
con  $A \rightarrow a \in S \wedge A \neq S$

$\hookrightarrow$  Shift (Reduce)

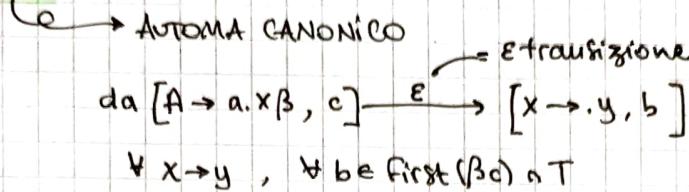
$\Rightarrow$  se nella tabella in ogni casella si trova al più un elemento  
allora la grammatica è di classe SLR(1)

altrimenti no

perché  $SLR(1) \subset LR(1)$

# item LR(1)

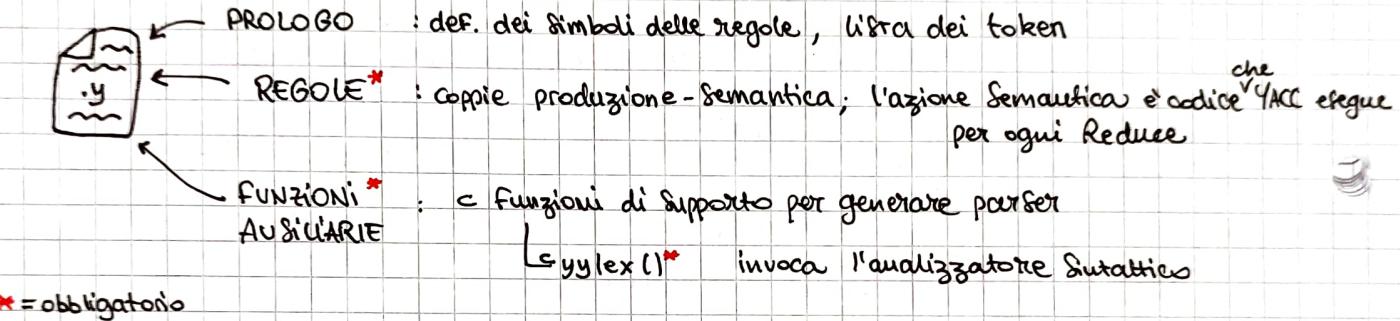
coppia formata da (item LR(0), simbolo di look-ahead)  
 $L \in (T \cup \$)$



# YACC

generatore di analizzatori sintattici

input: descrizione di una grammatica → output: Parser LALR(1) scritto in C



# Halting Problem

È programma H | ricevuti in input un programma P ind e stringa x  
Termina stampando "SI" → se  $P(x)$  termina, "NO" altrimenti.

- Supponendo di avere tale programma H

- lo sfruttiamo x costruire un programma R con 1 INPUT e:
  - $H(P) : \text{print "SI"} \rightarrow H(P, P) : \text{print "NO"}$
  - $H(P) \text{ va in loop } \rightarrow H(P, P) : \text{print "SI"}$

- eseguiamo R sul suo stesso testo:

- $R(R) : \text{print "SI"} \rightarrow$  Se  $R(R)$  non termina
- $R(R) : \text{print "NO"} \rightarrow$  Se  $R(R)$  termina

} Assurdo!

in quanto H è un programma impossibile  
⇒ non si termina il calcolo

# Turing-Completo →

Se calcola l'insieme di funzioni calcolabili con una Macchina di Turing

la macchina di T.  
è un formalismo matematico, è un'ipotetica macchina costituita da: ① nastro infinito diviso in celle nelle quali c'è un solo simbolo; ② testina che scrive sul nastro.  
INPUT: descrizione di una macchina  
e input e alfabeto codificato  
esegue la computazione sull'una data.

# tesi di Church

"l'insieme delle funzioni calcolabili della macchina di Turing, corrisponde a quello delle funzioni intuitivamente calcolabili,"

# MODULO 2 e 3

**parametri**: var. che compaiono nella def. (es in una funzione)

**attuali**: valori attribuiti ai  $\uparrow$  nella chiamata di funzione



## OGGETTI

un ogg di una Sottoclasse  $\leftarrow$  parametri della sua classe  
 $\leftarrow$  parametri della sua Superclasse

ES: A a = new B();  
B b = (B) a;

crea un "puntatore" allo stesso ogg  
a e b puntano alle stesse variabili



tabella che contiene dei metodi di una classe

ogni classe ha degli ogg con campi e puntatori ad una VTable

quando viene creato

Come Sottoclasse di un altro

«Superclasse = Sottoclasse »

ogg (Sottoclasse) eredita VT di S

Se c'è un override di metodi si usano quelli di S.

## Exceptions



- se ho un gestore di exc Y che ESTENDE un altro gestore X, un CATCH di tipo Y è in grado di catturare un throw X
- un throw interrompe l'esecuzione della function e fare CATCH il codice rimanente della function NON viene eseguito.

## POLIMORFISMO

?

Tipi

1. verificare se i tipi sono confrontabili
2. Supertipo = Sottotipo  $\rightarrow$  assegnazione legale
3. Non posso assegnare una lista di un tipo ad una lista di un altro, A MENO CHE: i 2 tipi siano compatibili
4. Non posso fare assegnamenti se a SX c'è un Sottotipo di oper. a DX

Allineamento  
alla parola

:= le variabili, anche se + piccole, occupano sempre almeno X bat.

# Semantica Operazionale Strutturata

= Studio logico sull'operatore logico dato

ESTERNA: si valuta solo 1 dato e se non è sufficiente, si passa al Successivo

INTERNA: si valutano tutti i dati

} DISCIPLINE di VALUTAZIONE

dx

st

: da dove si parte x la valutazione

viene attribuito un valore definito  $b'_1$

<valutazione del dato>

<valutazione dell'operazione>

dove avviene lo studio dei casi &

$\langle b'_1, \sigma' \rangle \rightarrow \langle b'_1, \sigma' \rangle$

rappresenta lo STORE

dato  $b'_1$ , lo STORE può cambiare  $\Rightarrow$  definiamo  $\sigma'$

$\langle b_0 \text{ XOR } b_1, \sigma \rangle \rightarrow \langle b_0 \text{ XOR } b'_1, \sigma' \rangle$

1. valutare tutti i dati (in base alle discipline di valutazione)

2. assegnare un valore <sup>a</sup> dato definito  
e scrivere l'output dell'operatore

es:  $\langle b_0 \text{ XOR } tt, \sigma \rangle \rightarrow \langle \sim b_0, \sigma' \rangle$

3. Se ottengo delle valutazioni senza dati →  
allora mi fermo

$\langle b_0 \text{ XOR } tt, \sigma \rangle \rightarrow \langle b_0, \sigma' \rangle$

altrimenti prosegua valutando il dato successivo  
e procedendo con lo studio.

:

END. Quando si ha un passaggio  $\nrightarrow$  risultato possibile delle porte logiche

**Esempio** per l'espressione booleana  $b_0 \text{ XOR } b_1$ , secondo le discipline di valutazione DESTRA-ESTERNA.

NB lo XOR vale V se solo 1 dei suoi argomenti vale vero

② cosa cambia se la si valutasse con INTERNO-SINISTRA?

1. valuto il dato a destra  $\rightarrow b_1$  lo valuto attribuendogli  $b_1'$   
 $\hookrightarrow$  valuto  $b_0$  e  $b_1'$  come conseguenza della valutazione dei dati

2. analizzo XOR: se  $b_1'$  è TRUE  $\Rightarrow$  negazione del 1° dato  
 $L = tt$

3. analizzo XOR: se  $b_1'$  è FALSE  $\Rightarrow$  risultato = 1° dato  
 $L = ff$

4. defuisco  $b_0$  con  $b_1'$ , ho finito di valutare il 1° dato,  
passo al successivo

5. analizzo XOR: se  $b_0'$  è TRUE  $\Rightarrow$   
se  $b_0'$  è FALSE  $\Rightarrow$

+ tt	$t'$ ff
ff	tt



② In questo caso, si nota differenza tra l'uso delle 2 discipline di valutazione,  
dato che abbiamo analizzato entrambi i dati anche con ED.

$$\frac{\langle b_1, \sigma \rangle \rightarrow \langle b_1', \sigma' \rangle}{\langle b_0 \text{ XOR } b_1, \sigma \rangle \rightarrow \langle b_0 \text{ XOR } b_1, \sigma' \rangle}$$

$$\frac{\langle b_0 \text{ XOR } b_1, \sigma \rangle \rightarrow \langle b_0, \sigma \rangle}{\langle b_0 \text{ XOR } ff, \sigma \rangle \rightarrow \langle b_0, \sigma \rangle}$$

$$\frac{\langle b_0, \sigma \rangle \rightarrow \langle b_0', \sigma' \rangle}{\langle \sim b_0, \sigma \rangle \rightarrow \langle \sim b_0', \sigma' \rangle}$$

$$\frac{\langle \sim b_0, \sigma \rangle \rightarrow \langle \sim b_0', \sigma' \rangle}{\langle \sim t, \sigma \rangle \rightarrow \langle \sim t', \sigma'' \rangle}$$