

Thompson ✓

Parsing LALR

subset NFA \rightarrow DFA
 \rightarrow Linguaggio deterministico ✓

SLR

DFA minimization ✓

SOD \rightarrow S att, L att, R att

Parsing DLL

Pumping \rightarrow libero e regolare

Parsing LR

Free grammar = tutte le produzioni sono in forma $A \rightarrow \dots$, quindi con solo un non terminale nella parte sinistra

Regular grammar = grammatica libera in forma
 $A \rightarrow \alpha$
 $A \rightarrow \alpha\beta$
 $A \rightarrow \epsilon$

Regular expressions = forme che usano $()$, $/$, $*$ e w, w_1

Pumping Lemma

scelta una certa pumping length, il linguaggio è regolare se:

- contiene una sotto stringa che può essere ripetuta ($\forall i \geq 0, xy^iz \in A$)
- la lunghezza della stringa è maggiore di 0 ($|y| > 0$)
- la sotto stringa deve essere lunga al massimo p (dove p è la pumping length) e deve apparire nei primi p caratteri ($|xy| \leq p$)

Ese: $a^h b^h \mid h \geq 3 \Rightarrow p = 3 \quad n = p$

\overbrace{aabb}^p

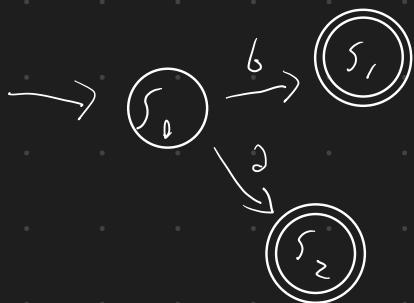
$\xrightarrow{\downarrow} aaaaabbb \rightarrow$ non fa parte del linguaggio

Esempio: $a^h b^h, h, h > 0$ è un linguaggio regolare?

S.

$a^* b^*$

Finit state automata



transition function:

$$(S_0, a) = \{S_2\}$$

$$(S_0, b) = \{S_1\}$$

NB:

NFA
accetta \emptyset

e multiple
transizioni con
la stessa etichetta

DFA

non accetta \emptyset
e multiple
transizioni con
la stessa etichetta

The morphic construction
algorithm can generate Una NFA patch
da un'espressione regolare

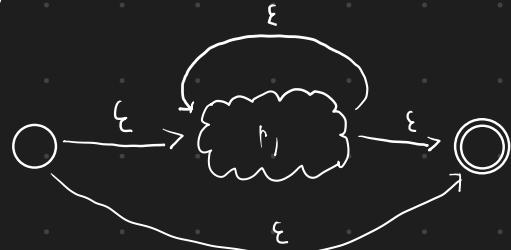
$$n = \partial$$



$$n = h_1, h_2$$

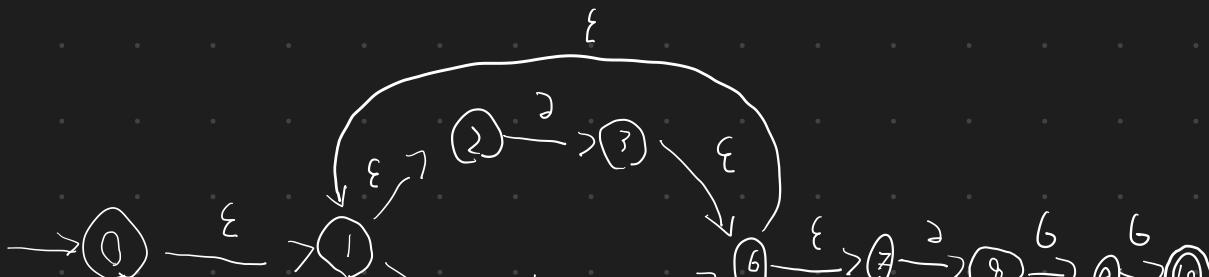


$$n = n^*$$



Selection: applies Thompson's algorithm to a sequential expression

Example: $(ab)^* ab^*$





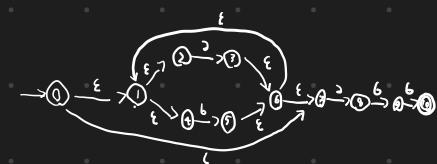
ϵ - closure

Sono l'insieme di stati raggiungibili con transizioni
in ε partendo da s

Simulazione di una NFA

Si va alla ricerca degli stati che compongono la NFA.
Usando come esempio l'esercizio precedente con la
parola ababb

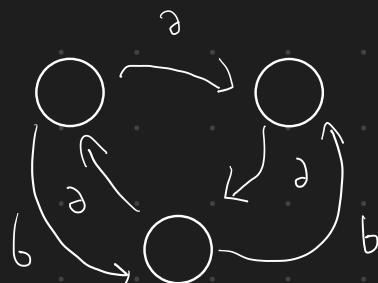
$S \delta T_1$	symbol.	$\epsilon\text{-closure}$	move
$T_0 = \epsilon\text{-closure}(Q)$ $= \{0, 1, 2, 4, 7\}$	∂	$\epsilon\text{-closure}(\overline{T}_0 + \partial) = \{0, 1, 2, \underline{3}, 6, 4, \underline{7}, \underline{8}\} = T_1$	$\{3, 8\}$
\overline{T}_1	6	$\{0, 1, 2, 4, \underline{5}, 6, 7, \underline{9}\} = T_2$	$\{5, 9\}$
T_2	∂	$\{0, 1, 2, 4, 7, \underline{3}, 6, 8\} = \overline{T}_1$	$\{3, 8\}$
\overline{T}_1	6	$\{0, 1, 2, 4, 7, \underline{5}, 6, \underline{9}\} = T_2$	$\{5, 9\}$
T_2	6	$\{0, 1, 2, 4, 7, \underline{5}, 6, \underline{10}\} = \overline{T}_1$	$\{5, 10\}$
\overline{T}_1	\$		



DFA

transizioni: totale: ogn: stato esiste una transizione con x

transizioni parziale: ogni stato ha al massimo una transizione con x



a è transizione totale, b è
transizione parziale

BOH, STA PARTE NON

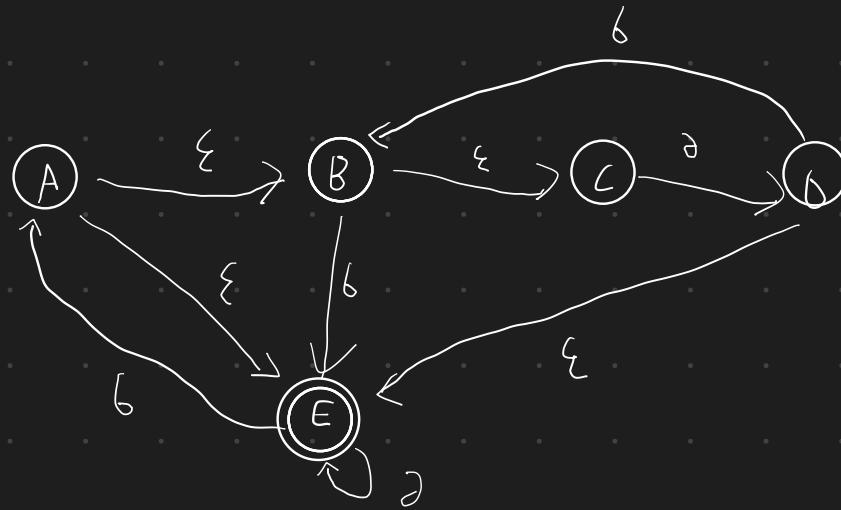
HA SENSO

Subiect: esistenziale (conclusio: NFA \rightarrow DFA)

Conversion of NFA to DFA

Use ϵ -closure for mapping multiple states NFA in single state DFA

Ex:



State

a

b

$$T_0 = \epsilon\text{-closure}(A)$$
$$= \{A, B, C, E\}$$

$$\epsilon\text{-closure}(T_0 + a) \rightarrow \{E, D\} = T_1$$

$$\epsilon\text{-closure}(T_1 + b) \rightarrow \{A, B, C, E\} = T_2$$

$$T_1 = \{0, E\}$$

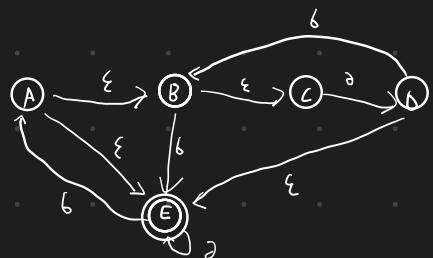
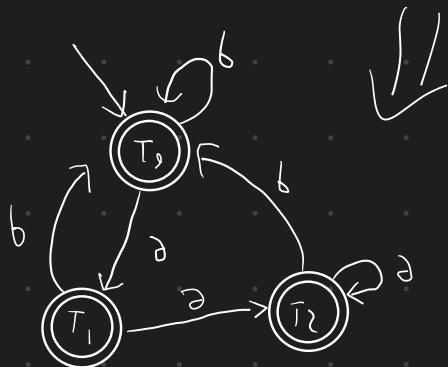
$$\{E\} \xrightarrow{\xi} \{E\} = T_2$$

$$\{A, B, E\} \xrightarrow{\xi} \{A, B, C, E\} = T_0$$

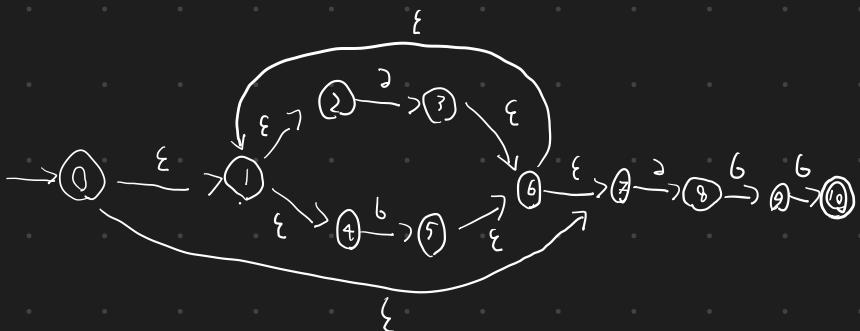
$$T_2 = \{E\}$$

$$\{E\} \xrightarrow{\xi} \{E\} = T_2$$

$$\{A\} \xrightarrow{\xi} \{A, B, C, E\} = T_0$$



Ex



$S \vdash \tau;$

$$T_0 = \{0, 1, 2, 4, 7\}$$

$$\{3, 8\} \xrightarrow{\delta} \{2, 8, 6, 1, 2, 4, 7\} = T_1$$

$$T_1 = \{1, 2, 3, 4, 6, 7, 8\}$$

$$\{3, 8\} = T_1$$

$$T_2 = \{1, 2, 4, 5, 6, 7\}$$

$$\{3, 8\} = T_1$$

$$T_3 = \{1, 2, 4, 5, 6, 7, 9\}$$

$$\{3, 8\} = T_1$$

$$T_4 = \{1, 2, 4, 5, 6, 7, 10\}$$

$$\{3, 8\} = T_1$$

6

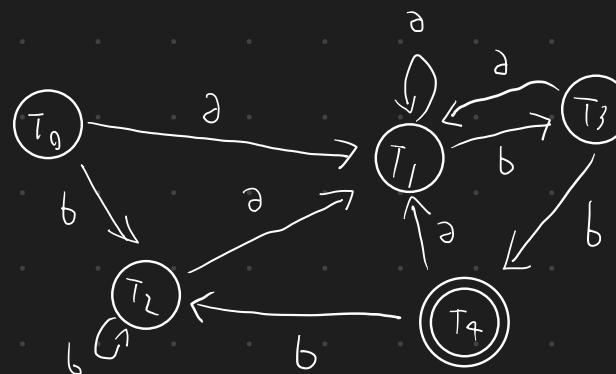
$$\{5\} \xrightarrow{\delta} \{1, 2, 4, 5, 6, 7\} = T_1$$

$$\{5, 9\} \xrightarrow{\delta} \{1, 2, 4, 5, 6, 7, 9\} = T_3$$

$$\{5\} = T_2$$

$$\{5, 10\} \xrightarrow{\delta} \{1, 2, 4, 5, 6, 7, 10\} = T_4$$

$$\{5\} = T_2$$



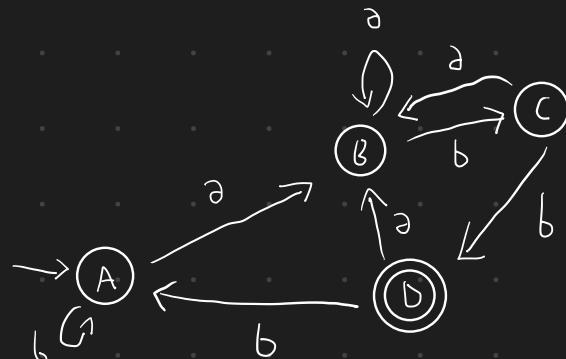
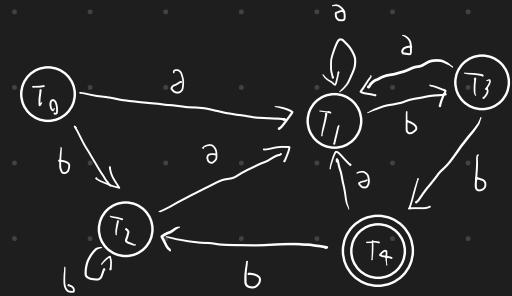
DFA

min. realization

segue un processo di partizionamento, mettendo in ordine i simboli

La lista di stati finali è non, al violenzo in base a/ complemento,
di cui gruppi

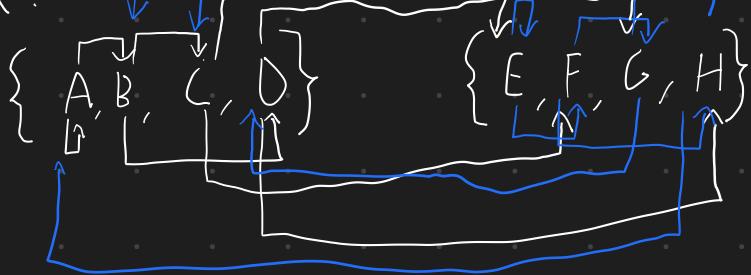
Ese:



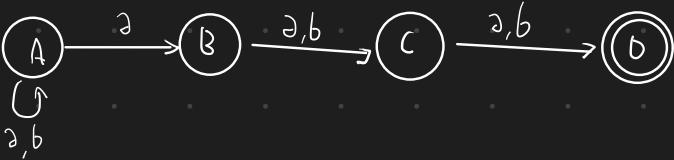
Ese:



$\rightarrow Q^b$



history confusion: \emptyset , g, a , $m, m, m, m, 0$



States:

$$T_0, \{A\}$$

a

$$\{A, B\} = T_1$$

b

$$\{A\} = T_0$$

$$T_1, \{A, B\}$$

$$\{A, B, C\} = T_2$$

$$\{A, C\} = T_3$$

$$T_2, \{A, B, C\}$$

$$\{A, B, C, D\} = T_4$$

$$\{A, C, D\} = T_5$$

$$T_4, \{A, B, C, D\} = T$$

$$\{A, B, D\} = T$$

$$\{A, B\} = T$$

$$\{A, B\}$$

$$\{A, B, C\} = T_6$$

$$\{A, C\} = T_7$$

$$\overline{T}_4 \{A, B, C, D\}$$

$$\{A, B, C, D\} = T_4$$

$$\{A, C, D\} = \overline{T}_5$$

$$\overline{T}_5 \{A, C, D\}$$

$$\{A, B, C, D\} = T_4$$

$$\{A, D\} = T_7$$

$$\overline{T}_6 \{A, B, D\}$$

$$\{A, B, C, D\} = T_4$$

$$\{A, C\} = T_7$$

$$\overline{T}_7 \{A, D\}$$

$$\{A, B\} = T_1$$

$$\{A\} = \overline{T}_9$$

parse,ing

processo di determinazione dell'appartenenza di una parola a una grammatica tratta da costituzioni delle alfabeti di den. Vggi, che

Top Down parsing = leftmost



$$W = i_1 + i_2 * i_3$$

$$E \rightarrow TE'$$

$$E' \rightarrow +TE' | \epsilon$$

$$T \rightarrow FT'$$

$$T' \rightarrow *FT' | \epsilon$$

$$F \rightarrow (E) | i$$

1

2

3

4

5

Predictive TO parsing

accepts language $LL(1)$ (legge l'input da sinistra a destra ($LL(1)$))
applies leftmost derivation ($LL(1)$) e usa un lookahead buffer di dimensione 1 ($LL(1)$)

Una grammatica $LL(1)$ non deve essere left recursive, ambigua o fattorizzabile a sinistra

possiamo costruire una parsing table usando i concetti di first e follow

$$\begin{cases} \text{First}(E') = +, \epsilon \\ \text{First}(F) = (, \text{id} \\ \text{First}(T) = \text{First}(F) = (, \text{id} \end{cases} \quad \begin{cases} \text{Follow}(E) = \$,) \\ \text{Follow}(T) = \text{Follow}(E) \cup \text{First}(E') - \epsilon = \$,), + \\ \text{Follow}(E') = \text{Follow}(E) = \$,) \end{cases}$$

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \epsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \epsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

Calcolare First e Follow, creando la parsing table inserendo le alternative per ogni simbolo in First. In caso di ϵ , allora inseriremo la alternativa per ogni follow. Parsing table

	id		+		*		()		\$	
--	----	--	---	--	---	--	---	--	---	--	----	--

Symbol	First	Follow		$E \rightarrow TE'$	$E' \rightarrow +TE'$	$E \rightarrow FE'$	$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
$E \rightarrow TE'$	(, id	\$,)		E	$E' \rightarrow +TE'$	$E \rightarrow FE'$	$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
$E' \rightarrow +TE'/\epsilon$, , ε	\$,)		T	$T \rightarrow FT'$	$T \rightarrow FT'$		
$T \rightarrow FT'$	(, id	\$,), +		T'	$T' \rightarrow \epsilon$	$T \rightarrow FT'$		
$T' \rightarrow *FT'/\epsilon$, ε	\$,), +		F	$F \rightarrow id$	$F \rightarrow (\epsilon)$	$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
$F \rightarrow id/(E)$	(, id	\$,), +, *						

Implementation

W = id + id * id

Stack	Input	Output	derivation
E	id + id * id	$E \rightarrow TE'$	$E \rightarrow TE'$
$T E'$	id + id * id	$T \rightarrow FT'$	$\rightarrow FT' E'$
$FT' E'$	id + id * id	$F \rightarrow id$	$\rightarrow id T' E'$
$T' E'$	+ id * id	$T' \rightarrow \epsilon$	$\rightarrow id E'$
E'	+ id * id	$E' \rightarrow +TE'$	$\rightarrow id + TE'$
TE'	id * id	$T \rightarrow FT'$	$\rightarrow id + FT' E'$
$FT' E'$	id * id	$F \rightarrow id$	$\rightarrow id + id T' E'$

E	id	$+$	$*$	$($	$)$	$$$
E'	$E \rightarrow TE'$	$E' \rightarrow +TE'$	$E \rightarrow FE'$	$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$	
T	$T \rightarrow FT'$		$T \rightarrow FT'$			
T'	$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$	$T' \rightarrow \epsilon$			
F	$F \rightarrow id$		$F \rightarrow (\epsilon)$	$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$	

$T'E'$ $* i_0$ $T \rightarrow *FT'$ $\rightarrow i_0 + i_0 * FT'E'$ $FT'E'$ i_0 $F \rightarrow i_0$ $\rightarrow i_0 + i_0 * i_0 T'E'$ $T'E'$ $$$ $T' \rightarrow \epsilon$ $\rightarrow i_0 + i_0 * i_0 E'$ E' $$$ $E' \rightarrow \epsilon$ $\rightarrow i_0 + i_0 * i_0$

Ex

 $S \rightarrow \alpha A B \beta$ $A \rightarrow A c | d$ $B \rightarrow C D$ $C \rightarrow c | \epsilon$ $D \rightarrow f | \epsilon$ $S \rightarrow \alpha A B \beta$ $A \rightarrow \alpha A'$ $A' \rightarrow c A'$ $A' \rightarrow \epsilon$ $B \rightarrow C D$ $C \rightarrow c$ $C \rightarrow \epsilon$ $D \rightarrow f$ $D \rightarrow \epsilon$ F_i F_o α $$$ d c, e, f, b c e, f, b ϵ e, f, b e, f, ϵ b e f, b ϵ f, b f b ϵ b α b c i_0 c f $$$ S $\alpha A B \beta$ A ϵ B ϵ C ϵ D ϵ

Left recursion elimination

 $A \rightarrow A \alpha | \beta \Rightarrow \begin{cases} A \rightarrow \beta A' \\ A' \rightarrow \alpha A' | \epsilon \end{cases}$

Left factorization

$$A \rightarrow \alpha\beta, |\alpha\beta_2 \Rightarrow \begin{cases} A \rightarrow \alpha A' \\ A' \rightarrow \beta, |\beta_2 \end{cases}$$

Bottom up parsing

Consistono nell'utilizzo di grammatiche estese ($S \rightarrow S'$), usano algoritmo shift/reduce e usano automi caratteristici

LR(0)

è un automata composto da stati, ossia derivazioni con l'aggiunta di:

un indicatore di posizione ($A \rightarrow B \bullet a$). La posizione iniziale è $S \rightarrow \bullet S_1$. Possiamo applicare la chiusura degli stati, ossia applicare l'indicatore di posizione a tutte le derivazioni con non-terminali processabili dall'indicatore di posizione.

Es

$$S \rightarrow S_1 \quad \text{closure}(S) : S \rightarrow \bullet S_1$$

$$S_1 \rightarrow A c$$

$$A \rightarrow d B F$$

$$B \rightarrow e$$

$$\begin{cases} S_1 \rightarrow A c \\ A \rightarrow d B F \end{cases}$$

Da qui si può riconoscere le generate di stati

Ese:

$$\begin{cases} S_1 \rightarrow a A B c \\ A \rightarrow A b c \\ A \rightarrow b \\ B \rightarrow d \end{cases}$$

$$\begin{cases} S_1 \rightarrow S_1 \xrightarrow{S_1} 1 \\ S_1 \rightarrow a A B c \xrightarrow{a} 2 \\ S_1 \rightarrow S_1 \cdot \xrightarrow{} 1 \end{cases}$$

$$\begin{cases} S_1 \rightarrow S_1 \cdot \\ S_1 \rightarrow a A B c \xrightarrow{a} 3 \\ A \rightarrow A b c \xrightarrow{a} 3 \\ A \rightarrow \cdot b \xrightarrow{b} 4 \end{cases}$$

$$\begin{cases} S_1 \rightarrow a A B c \xrightarrow{a} 3 \\ A \rightarrow A b c \xrightarrow{a} 3 \\ A \rightarrow \cdot b \xrightarrow{b} 4 \\ S_1 \rightarrow a A B c \xrightarrow{a} 5 \\ A \rightarrow A b c \xrightarrow{a} 6 \\ B \rightarrow \cdot d \xrightarrow{d} 7 \end{cases}$$

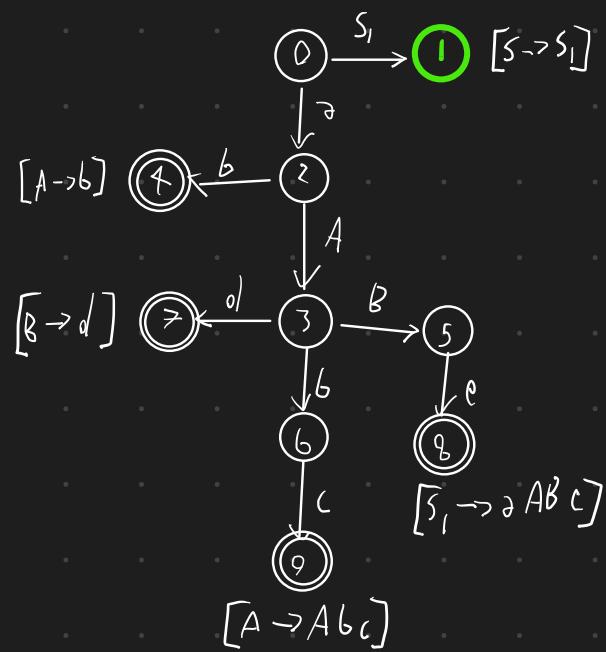
$$\begin{cases} A \rightarrow b \cdot \\ S_1 \rightarrow a A B c \xrightarrow{a} 8 \end{cases}$$

$$\begin{cases} S_1 \rightarrow a A B c \xrightarrow{a} 8 \\ B \rightarrow d \cdot \end{cases}$$

$$\begin{cases} A \rightarrow A b c \cdot \xrightarrow{c} 9 \\ S_1 \rightarrow a A B c \cdot \end{cases}$$

$$\begin{cases} B \rightarrow d \cdot \xrightarrow{d} 8 \\ S_1 \rightarrow a A B c \cdot \end{cases}$$

$$\begin{cases} A \rightarrow A b c \cdot \xrightarrow{c} 9 \\ B \rightarrow d \cdot \xrightarrow{d} 7 \end{cases}$$



Shift & Reduce (SLR(1))

Vediamo un esempio di come si possa implementare questo tipo di associazione. Supponiamo che abbiamo una classe `Carta` con i seguenti attributi:

```

class Carta {
    int numero;
    String titolo;
    String autore;
}

```

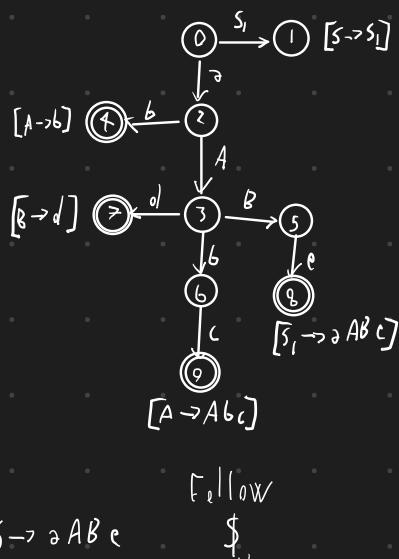
Una classe `Carta` ha un attributo `titolo` e un attributo `autore`. Ora, supponiamo che abbiamo un'altra classe `Autore` con l'attributo `titoli`:

```

class Autore {
    String nome;
    List<String> titoli;
}

```

Un oggetto `Autore` ha un attributo `titoli` che è una lista di stringhe. Ora, vogliamo che ogni oggetto `Carta` abbia un collegamento a un oggetto `Autore`, dove il titolo della cartolina è uno dei titoli dell'autore. Per fare questo, possiamo aggiungere un attributo `autore` alla classe `Carta` e utilizzare la classe `Autore` per gestire i titoli.



	a	b	c	d	e	\$	S	A	B
0	s2								
1							l		
2		s4							
3		s6		s7					
4		r3		r3					
5						s8			
6									
7									
8							h7		
9							h1		
		t2		t2					

$$\begin{array}{l}
 A \rightarrow Abc \\
 A \rightarrow b \\
 B \rightarrow 0 \\
 \end{array}
 \quad
 \begin{array}{l}
 \text{a} \\
 \text{b} \\
 \text{e}
 \end{array}$$

→ PPL: (2 Einst.): $W = abbc_01e$

$abbc_01e$

0 X A 3 b A 9 7 5 8 1 acc

2 b A 6 A 01 B e S f acc

0	a	b	c	d	e	f	g	h	A	B
1	s2									
2		s4								
3			s6							
4				s3						
5					s7					
6						s9				
7							s8			
8								m4		
9									m1	

Risolvono i conflitti

5	+	x	
6	s3, h1	s4, f1	

5	s3, h1	s4, f1	
6	s3, h2	s4, f2	

Forscello spiegate alla Dylan

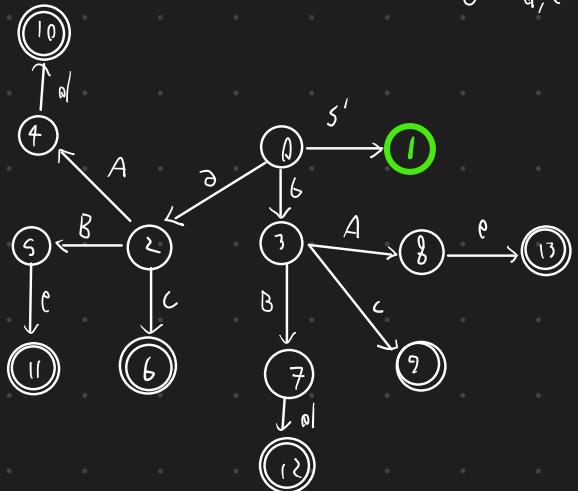
LR(1)

Una variante di LR dove oltre alle posizioni, ha gli stati registrando anche i follow (lookahead) function

$S \rightarrow aA_0 \mid bB_0 \mid aBe \mid bAe$

$A \rightarrow c$

$B \rightarrow c$



FOLLOW

S	\$
A	a, e
B	a, e

$$0 \begin{cases} S \rightarrow S' \{ \$ \} \xrightarrow{s'} 1 \\ S' \rightarrow a \cdot A_0 \{ \$ \} \xrightarrow{a} 2 \\ S' \rightarrow b \cdot B_0 \{ \$ \} \xrightarrow{b} 3 \\ S' \rightarrow a \cdot Be \{ \$ \} \xrightarrow{a} 2 \\ S' \rightarrow b \cdot Ae \{ \$ \} \xrightarrow{b} 3 \end{cases}$$

$$1 [S \rightarrow S' \cdot \{ \$ \}]$$

$$2 \begin{cases} S' \rightarrow a \cdot A_0 \{ \$ \} \xrightarrow{A} 1 \\ S' \rightarrow a \cdot Be \{ \$ \} \xrightarrow{B} 5 \\ A \rightarrow \cdot c \{ a \} \xrightarrow{c} 6 \\ B \rightarrow \cdot c \{ e \} \xrightarrow{e} 6 \end{cases}$$

$$3 \begin{cases} S' \rightarrow b \cdot B_0 \{ \$ \} \xrightarrow{B} 7 \\ S' \rightarrow b \cdot Ae \{ \$ \} \xrightarrow{A} 8 \\ B \rightarrow \cdot c \{ a \} \xrightarrow{c} 9 \\ A \rightarrow \cdot C \{ e \} \xrightarrow{e} 9 \end{cases}$$

$$4 [S' \rightarrow a \cdot A_0 \{ \$ \} \xrightarrow{a} 10]$$

$$5 [S' \rightarrow aB \cdot e \{ \$ \} \xrightarrow{e} 11]$$

$$6 \begin{bmatrix} A \rightarrow \cdot c \{ a \} \\ B \rightarrow \cdot c \{ e \} \end{bmatrix}$$

$$7 [S' \rightarrow b \cdot B_0 \cdot \{ \$ \} \xrightarrow{b} 12]$$

$$8 [S' \rightarrow b \cdot Ae \{ \$ \} \xrightarrow{e} 13]$$

$$9 \begin{bmatrix} B \rightarrow \cdot c \{ a \} \\ A \rightarrow \cdot C \{ e \} \end{bmatrix}$$

$$10 [S' \rightarrow a \cdot A_0 \cdot \{ \$ \}]$$

$$11 [S' \rightarrow aB \cdot e \cdot \{ \$ \}]$$

$$12 [S' \rightarrow b \cdot B_0 \cdot \{ \$ \}]$$

$$13 [S' \rightarrow b \cdot Ae \cdot \{ \$ \}]$$

Unparsable LR is left infinite Simple LR (SLR) spoken phrasal
also can conflicts shift/reduce or reduce/reduce

Un conflitto S/R si presenta in forma

$$\begin{bmatrix} \text{x} \text{x} \text{x}, \text{x} & \xrightarrow{\delta} \\ \text{y} \text{y} \text{y}, \cdot \text{-} \text{a} \end{bmatrix}$$

mentre un R/R ha forma:

$$\begin{bmatrix} \text{x} \text{x} \text{x}, \cdot \text{-} \text{a} \\ \text{y} \text{y} \text{y}, \cdot \text{-} \text{a} \end{bmatrix}$$

avviomata LR(1)

Verificare se LR(1) avverte in caso di stati abbondanti
le medesime transizioni, ma con lookahead set diversi,
possiamo unire gli stati in un unico stato.

Ese:

$$4 \left[L \rightarrow \cdot \text{a} \cdot \{ =, \$ \} \right]$$
$$11 \left[L \rightarrow \cdot \text{a} \cdot \{ \$ \} \right]$$

$$4 \& 11 \left[L \rightarrow \cdot \text{a} \cdot \{ =, \$ \}, L \rightarrow \cdot \text{a} \cdot \{ \$ \} \right]$$

LALR(1)

Ugårda sätta till lookahead och LRm(1).

Röba con dyhan

Un händelse alternativt per att göra LALR(1) parsing tabell
är tillräcklig att automatiskt symboli, ossif LR(1) slupp if
lookahead set räcke kompakt där variabili symbolische.

Ex:

$$\begin{array}{l} S \rightarrow L = R | R \\ L \rightarrow * R | ; o | \\ R \rightarrow L \end{array}$$

$$\begin{array}{l} \text{look} \\ S \$ \\ L = \\ R = \end{array}$$

$$\begin{array}{l} x_0 = \$ \\ x_1 = x_2 = \$ \\ x_3 = x_4 = \$ \\ x_5 = x_6 = \$ \end{array}$$

$$Q \left[\begin{array}{l} S \rightarrow S^1 \{x_0\} \stackrel{S^1}{\Rightarrow} 1 \\ S^1 \rightarrow \cdot L = R \{x_0\} \stackrel{R}{\Rightarrow} 2 \\ S^1 \rightarrow \cdot R \{x_0\} \stackrel{R}{\Rightarrow} 3 \\ L \rightarrow \cdot * R \{=, x_0\} \stackrel{*}{\Rightarrow} 4 \\ L \rightarrow \cdot ; o \{=, x_0\} \stackrel{; o}{\Rightarrow} 5 \\ R \rightarrow \cdot L \{x_0\} \stackrel{L}{\Rightarrow} 2 \end{array} \right]$$

$$1 [S \rightarrow S^1 \cdot \{x_1\}]$$

$$2 [S^1 \rightarrow L \cdot = R \{x_2\} \stackrel{=} {\Rightarrow} 6 \\ R \rightarrow L \cdot \{x_3\}]$$

$$5 [L \rightarrow ; o \cdot \{x_6\}]$$

$$6 \left[\begin{array}{l} S^1 \rightarrow L = \cdot R \{x_2\} \stackrel{R}{\Rightarrow} 9 \\ R \rightarrow \cdot L \{x_2\} \stackrel{L}{\Rightarrow} 8 \\ L \rightarrow \cdot * R \{x_7\} \stackrel{*}{\Rightarrow} 4 \\ L \rightarrow \cdot ; o \{x_7\} \stackrel{; o}{\Rightarrow} 5 \end{array} \right]$$

$$7 [L \rightarrow * R \cdot \{x_8\}]$$

$$8 [R \rightarrow L \cdot \{x_9\}]$$

$$9 [S^1 \rightarrow L = R \cdot \{x_{10}\}]$$

$x_6 = \{=\} x_0, x_5 = \{=, \$\}$
 $x_7 = x_2 = \$$
 $x_8 = x_5 = \{=, \$\}$
 $x_9 = x_5, x_7 = \{=, \$\}$
 $x_{10} = x_7 = \$$

$\exists [S^1 \rightarrow R \cdot \{x_4\}]$
 $\forall \begin{cases} L \rightarrow \text{if } .R \{x_5\} \Rightarrow 7 \\ R \rightarrow .L \{x_5\} \stackrel{L}{\Rightarrow} 8 \\ L \rightarrow .*R \{x_5\} \stackrel{*}{\Rightarrow} 4 \\ L \rightarrow .!R \{x_5\} \stackrel{!}{\Rightarrow} 5 \end{cases}$

Ex

	F	F'	S	a	b	\$
$S \rightarrow AB$	a	$\$, a$	AB	a	b	\$
$S \rightarrow b$	b	$\$, a$	A	A'		
$A \rightarrow A'$	a	$b, \$$	A'	$aBSA$		
$A' \rightarrow aBSA'$	a	$b, \$$	B	ϵ	b	ϵ
$B \rightarrow b$	b	$a, \$$				
$B \rightarrow \epsilon$	ϵ	$a, \$$				

Analisi semantica

Il processo di aggiunta di informazioni, alla struttura semantica, non obiettiva, tramite la grammatica. Genera/riente symbol table population e type checking

Syntax-directed definition

Chiamata anche attribute grammar, è il metodo di implementazione degli attributi

per effetto negativo: se una sintattica attribuita a un simbolo appartiene già a una grammatica, allora non può appartenere a quella grammatica.

La SSD si basa su un'algebra sintattica astratta (una relazione compatta dell'algebra di derivazioni).

GL: attributi possono essere sintetici o ereditati (il valore è determinato da nodi figli) o ereditati (il valore è determinato da nodi genitori o fratelli). GL: attributi dei non-terminali sono esclusivamente sintetizzati.

Es	$S \rightarrow E$	$\{S.\text{val} = E.\text{val}\}$
	$E \rightarrow E' + T$	$\{E.\text{val} = E'.\text{val} + T.\text{val}\}$
	$E \rightarrow T$	$\{E.\text{val} = T.\text{val}\}$
	$T \rightarrow T' * F$	$\{T.\text{val} = T'.\text{val} * F.\text{val}\}$
	$T \rightarrow F$	$\{T.\text{val} = F.\text{val}\}$
	$F \rightarrow (E)$	$\{F.\text{val} = E.\text{val}\}$
	$F \rightarrow \text{cifra}$	$\{F.\text{val} = \text{digit. lexical}\}$

Evaluation of SSD

Consiste nella creazione di un grafo delle dipendenze, ossia aggiungendo una freccia che indica che "X ha bisogno di Y per esistere"





E_K

$S \rightarrow TL$	$\{L.type = T.type\}$
$T \rightarrow int$	$\{T.type = integer\}$
$T \rightarrow float$	$\{T.type = float\}$
$L \rightarrow L, : al$	$\{L.type = L.type, al.type = L.type\}$
$L \rightarrow al$	$\{al.type = L.type\}$



E_S

$S \rightarrow E$	$T \rightarrow FT'$	$\{T'.; = F.s; T.s = T'.;\}$
$E \rightarrow TE'$	$T' \rightarrow *FT'$	$\{T'.s = T'.s; T'.; = T'. * F.s\}$
$E' \rightarrow +TE'$	$T' \rightarrow E$	$\{T'.s = T'.;\}$
$E' \rightarrow E$	$F \rightarrow c.ifra$	$\{F.s = c.ifra.lexval\}$
$F \rightarrow (E)$		

3 * 5





Evaluation durante BDU parsing è implementata tramite l'integrazione dell'algoritmo shift/reduce e l'SSD S-attributi.

Symbol Tables

Sono strutture dati simili a dizionari, e implementate funzioni di insert, delete e lookup. Generalmente sono implementate sotto forma di hash tables.

Interpreti

Un interprete si avvale di symbol tables e albeni sintattici anziché. Un interprete usufruisce di una symbol table per connettere una variabile al suo valore ($vtab$) e una symbol table per connettere una funzione al suo abstract tree ($ftab$)

Intermediate code generation

La rappresentazione intermedia del codice può essere in forma
di alberi, rappresentati in un altro linguaggio oppure in codice a tre indirizzi
($x = y$ op z)

Sintax directed/ translation of expressions

attributi e funzioni ausiliarie

E.adm → è il temperatura che contiene il valore di E

S.codice → il codice messo per S

gen(str) → emette la stringa str

newtemp() → mette un nuovo temperatura

▷ → concatenare frammenti di codice intermedio

Controlli di flusso

S.next → Segna l'inizio del codice che deve essere eseguito al termine di S

S.code → la sequenza di codice che implementa S e termina con S.next

B.true → segna l'inizio del codice che deve essere eseguito in caso B sia vero

B.false → identico a B.true ma gestisce il caso falso

B.code → la sequenza di codice che determina la condizione di B