

Linguaggi Formali e Compilatori

Argomento 01

Prof. Giuseppe Psaila

Laurea Magistrale in Ingegneria Informatica
Università di Berga

Un **TRADUTTORE** è un componente software che legge un testo scritto in uno specifico **Linguaggio Formale** e lo traduce

- o in un altro linguaggio formale
- o in azioni.

Traduttori e Compilatori

- Un **COMPILATORE** è un traduttore che traduce un linguaggio di programmazione nella corrispondente versione binaria.
- Un **INTERPRETE** è un traduttore che traduce un programma scritto in un linguaggio di programmazione in azioni (esegue direttamente il programma).

Traduttori e Compilatori

Organizzazione di un **COMPILATORE**

- **FRONT-END**: la parte del compilatore che analizza il testo.
- **BACK-END**: la parte che genera il codice binario della specifica piattaforma.

In questo modo, lo stesso Front-End può essere accoppiato a Back-End diversi (es. il g++ della GNU).

- **Scanner (Lexer)**: riconosce gli elementi lessicali del linguaggio
- **Parser (Analizzatore Sintattico)**: riconosce la struttura sintattica del linguaggio
- **Analizzatore Semantico**: dà un significato al testo, riconoscendo gli errori semantici

Ci occupiamo del Front-End, cioè della parte *linguistica* dei traduttori:

- **Ambito formale**
all'interno del quale le tecniche sono definite
- **Automi a Stati Finiti**
La tecnica usata per il riconoscimento lessicale
- **Strutture Sintattiche e Tecniche di Parsing**
Definire e riconoscere le strutture sintattiche di un linguaggio
- **Semantica dei Linguaggi**
Definire il significato dei linguaggi formali

Libri di Testo

(non necessari)

- Crespi-Reghizzi
Linguaggi Formali e Compilazione
Pitagora Editrice
- Aho, Lam, Sehi, Ullman
Compilatori
Principi, Tecniche e Strumenti
Pearson - Addison Wesley

Versioni del Corso

- **6 CFU Linguaggi Formali**
Esame scritto
- **9 CFU Linguaggi Formali e Compilatori**
Esame scritto (fino a 26 punti)
Progetto semi-facoltativo (6 punti), impegno massimo 2 settimane
 - 18 ore di esercitazioni per imparare a usare gli strumenti di generazione automatica dei traduttori e impostare il progetto
 - 18 ore di tutorato (con me) per preparavi all'esame scritto

Aspetti Formali

- **Alfabeto**

Un **Alfabeto** è un **INSIEME FINITO** di **SIMBOLI**
o **CARATTERI TERMINALI**.

$$\Sigma = \{a_1, a_2, \dots, a_k\}$$

$|\Sigma| = k$ è la **Cardinalità** di Σ

- **Stringa**

Una **STRINGA** o **FORMA** è una **sequenza di caratteri terminali**.

Alfabeto e Stringa

Esempio: sia $\Sigma = \{a, b\}$

- aa
- $abab$
- b
- baa

sono stringhe

Stringa

- La **lunghezza** di una stringa s , indicata come $|s|$, è il numero dei suoi caratteri
- La stringa ϵ (**Stringa VUOTA**) ha lunghezza 0, cioè $|\epsilon| = 0$
- Quindi, data una stringa s , $|s| \geq 0$.
- Date due stringhe $x = a_1a_2\dots a_h$ e $y = b_1b_2\dots b_k$, esse si dicono **UGUALI** se $h = k$ e $a_i = b_i$ per ogni $i \in [1 \dots h]$.
Esempio: $aba \neq baa \neq ba$.

Un **Linguaggio** è un **Insieme di Stringhe**.

Esempi: $\Sigma = \{a, b\}$

- $L_1 = \{aa, aaa\}$
- $L_2 = \{aba, aab\}$
- $L_3 = \text{insieme delle stringhe che contengono un ugual numero di } a \text{ e di } b$
 $= \{ab, ba, aabb, \dots\}$

- La **CARDINALITÀ** di un linguaggio L è il numero delle sue stringhe. Si indica come $|L|$.
- Un linguaggio avente cardinalità **FINITA** (risp. **INFINITA**) è detto **Linguaggio FINITO** (risp. **Linguaggio INFINITO**).
- Una stringa s appartenente ad un linguaggio L è detta **Frase di L** ; una stringa s che **non appartiene** ad un linguaggio L è detta **NON FRASE** di L , o **Stringa Scorretta** per L .
- $L = \emptyset$ è un caso particolare di linguaggio finito, detto **Linguaggio VUOTO** ($|\emptyset| = 0$).

Esempi

- $L_1 = \{aa, aaa\}$
 $|L_1| = 2$, finito.
 $s_1 = aa$ è frase di L_1 .
 $s_2 = ab$ è non frase di L_1 .
- $L_3 = \text{insieme delle stringhe contenenti un ugual numero di } a \text{ e di } b$
 L_3 è infinito.
 $s_1 = aa$ è non frase di L_3 .
 $s_2 = ab$ è frase di L_3 .

Concatenamento

Date due stringhe $x = a_1a_2\dots a_h$ e $y = b_1b_2\dots b_k$,
 $x \bullet y = a_1a_2\dots a_h b_1b_2\dots b_k$
che si scrive anche $xy = x \bullet y$.

Proprietà del Concatenamento

- Il concatenamento **non è commutativo**.

$xy \neq yx$ in generale.

- Il concatenamento **è associativo**.

$$(xy)z = x(yz)$$

- La lunghezza del concatenamento è **la somma delle lunghezze**.

$$|xy| = |x| + |y|$$

La stringa vuota ϵ è **l'elemento neutro rispetto al concatenamento**, $x \bullet \epsilon = \epsilon x = x$, con

$$|x \bullet \epsilon| = |x| + |\epsilon| = |x| + 0 = |x| \text{ e}$$

$$|\epsilon \bullet x| = |\epsilon| + |x| = 0 + |x| = |x|$$

SottoStringhe

Sia $x = uyz$, dove u , y e z sono stringhe.

- u , y , z sono **sottostringhe** di x .
- u è un **Prefisso** di x .
- z è un **Suffisso** di x .

Una sottostringa si dice **PROPRIA** se non coincide con la stringa che la contiene.

Prefissi

- Sia x una stringa di lunghezza **almeno** k , cioè $|x| \geq k$.
- Con $k : x$ si indica il prefisso di x avente lunghezza k .
- $u = k : x$ e $x = uz$, con $|u| = k$.
- $k : x$ è detto **l'inizio di x di lunghezza k** .

Esercizio 1.1: $\Sigma = \{a, b, c\}$, $x = aabacba$

- Prefissi:
- Suffissi:
- Altre sottostringhe (non prefissi e non suffissi):
- 3 : $aabacba =$

Esercizio 1.1: $\Sigma = \{a, b, c\}$, $x = aabacba$

- Prefissi: $a, aa, aab, aaba, aabac, aabacb, aabacba$
- Suffissi: $a, ba, cba, acba, bacba, abacba, aabacba$
- Altre sottostringhe (non prefissi e non suffissi): $a, b, c, ab, aba, abac, abacb, ba, bac, bacb, ac, acb, cb$
- 3 : $aabacba = aab$

Proprietà

- $(x^R)^R = x$
- $(xy)^R = y^R x^R$
- $\epsilon^R = \epsilon$

Esempi

$x = ROMA$	$x^R = AMOR$
$x = OMAR$	$x^R = RAMO$
$x = OTTO$	$x^R = OTTO$
$x = UNO$	$x^R = ONU$

Ripetizioni - Elevamento a Potenza

La **POTENZA** n -sima (con $n \geq 0$ intero) di una stringa

 x

è il **concatenamento di x con se stessa $n - 1$ volte**.

Formalmente:

- $x^n = x^{n-1} \bullet x$ per $n > 0$
- $x^0 = \epsilon$

L'elevamento a potenza e la riflessione hanno precedenza rispetto al concatenamento.

Esercizio 1.2: $\Sigma = \{a, b\}$

- $x = ab$

$$x^0 =$$

$$x^1 =$$

$$x^2 =$$

- $y = a^2$

$$y^3 =$$

- $\epsilon^0 =$

$$\epsilon^3 =$$

Esercizio 1.2: $\Sigma = \{a, b\}$

- $x = ab$
 $x^0 = \epsilon$
 $x^1 = x = ab$
 $x^2 = (ab)^2 = abab$
- $y = a^2$
 $y^3 = (a^2)^3 = (aa)^3 = a^2a^2a^2 = a^6$
- $\epsilon^0 = \epsilon$
 $\epsilon^3 = \epsilon$

Esercizio 1.3: $\Sigma = \{a, b\}$

- $x = ab^2 =$
 $y = (ab)^2 =$
- $x = ab^R =$
 $y = (ab)^R =$

Esercizio 1.3: $\Sigma = \{a, b\}$

- $x = ab^2 = abb$
 $y = (ab)^2 = abab$
- $x = ab^R = ab$
 $y = (ab)^R = ba$

Concatenamento

Dati **due linguaggi** L_1 e L_2 ,
 $L = L_1 \bullet L_2 = \{xy \mid x \in L_1 \text{ AND } y \in L_2\}.$

Inizi

Dato un linguaggio L e un numero $k > 0$ intero,
 $k : L = \{y \mid y = k : x \text{ AND } x \in L\}$,
detto **Linguaggio degli Inizi di Lunghezza k di L** .

Prefissi

Dato un linguaggio L ,

$\text{Prefissi}(L) = \{y \mid x = yz \text{ AND } x \in L \text{ AND } z \neq \epsilon\}$,
detto il **Linguaggio dei Prefissi (Propri) di L** .

Un linguaggio L è detto **PRIVO DI PREFISSI** se
Nessuno dei Suoi Prefissi è contenuto in L .
Formalmente: $\text{Prefissi}(L) \cap L = \emptyset$.

Esercizio 1.4

- Dato $\Sigma = \{a, b\}$, e $L = \{a^n b^n \mid n \geq 1\}$,
 $\text{Prefissi}(L) \cap L =$
- Dato $\Sigma = \{a, b\}$, e $L = \{a^m b^n \mid m > n \geq 1\}$,
 $\text{Prefissi}(L) \cap L =$

Esercizio 1.4

- Dato $\Sigma = \{a, b\}$, e $L = \{a^n b^n \mid n \geq 1\}$,
 $\text{Prefissi}(L) \cap L = \emptyset$
- Dato $\Sigma = \{a, b\}$, e $L = \{a^m b^n \mid m > n \geq 1\}$,
 $\text{Prefissi}(L) \cap L = \neq \emptyset$

Riflessione

Dato un linguaggio L ,

$$L^R = \{x \mid y \in L \text{ AND } x = y^R\}.$$

Ripetizione / Potenza

Dato un linguaggio L e un numero $n \geq 0$ intero,
la **Potenza n -sima di L** è

- $L^n = L^{n-1} \bullet L$ se $n > 0$
- $L^0 = \{\epsilon\}$

Operazioni sui Linguaggi

Operazioni sui Linguaggi

Casi Particolari

- $L^0 = \{\epsilon\} \neq \emptyset$
infatti $|\{\epsilon\}| = 1$, mentre $|\emptyset| = 0$
- $\emptyset^0 = \{\epsilon\}$
- $L \bullet \emptyset = \emptyset$
 \emptyset è l'elemento **ZERO** rispetto al concatenamento
- $L \bullet \{\epsilon\} = L$
 $\{\epsilon\}$ è l'elemento **NEUTRO** rispetto al concatenamento

Esercizio 1.5

Si considerino i linguaggi

- $L_1 = \{a^i \mid i \geq 0 \text{ PARI}\} = \{\epsilon, aa, aaaa, \dots\}$
- $L_2 = \{b^j a \mid j \geq 1 \text{ DISPARI}\} = \{ba, bbba, \dots\}$

Che cosa Risulta?

- $L_1 \bullet L_2 =$
- $L_1 \bullet L_1 = L_1^2 =$

Esercizio 1.5

Si considerino i linguaggi

- $L_1 = \{a^i \mid i \geq 0 \text{ PARI}\} = \{\epsilon, aa, aaaa, \dots\}$
- $L_2 = \{b^j a \mid j \geq 1 \text{ DISPARI}\} = \{ba, bbba, \dots\}$

Risulta

- $L_1 \bullet L_2 = \{a^i b^j a \mid i \geq 0 \text{ PARI AND } j \geq 1 \text{ DISPARI}\} = \{\epsilon ba, a^2 ba, \epsilon b^3 a, \dots\}$
- $L_1 \bullet L_1 = L_1^2 = L_1$, perché per ogni coppia di numeri pari h e k , $(h+k)$ è pari, quindi $a^{(h+k)} \in L_1$

Esempio

Dato l'alfabeto $\Sigma = \{a, b\}$, il linguaggio delle stringhe x con lunghezza $0 \leq x \leq 2$ è definito come
 $L = \{\epsilon, a, b\}^2$.

Per un generico $n \geq 1$
 $L = \{\epsilon, a, b\}^n$.

Operatore STELLA**Chiusura Riflessiva al Concatenamento**

$$L^* = \bigcup_{h=0, \dots, \infty} L^h = \{\epsilon\} \cup L^1 \cup L^2 \cup \dots$$

Esercizio 1.6

- Dato $L = \{ab, ba\}$,
 $L^* =$
- Dato $L = \{a^{2n} \mid n \geq 0\}$,
 $L^* =$

Esercizio 1.6

- Dato $L = \{ab, ba\}$,
 $L^* = \{\epsilon, ab, ba, abab, abba, ababab, bababa, \dots\}$,
INFINITO
- Dato $L = \{a^{2n} \mid n \geq 0\}$,
 $L^* = \{a^{2n} \mid n \geq 0\} = L$

Monoide Libero

Dato un alfabeto Σ e un linguaggio $L = \Sigma$ (linguaggio dei simboli terminali)

Σ^* contiene tutte le stringhe che possono essere costruite concatenando i caratteri terminali.

Ogni linguaggio formale L di alfabeto Σ è incluso in Σ^* ,
 $L \subseteq \Sigma^*$.

Operazioni sui Linguaggi

Operazioni sui Linguaggi

Proprietà

- $L \subseteq L^*$ (Monotonicità)
- Se $x \in L^*$ e $y \in L^*$,
 allora $xy \in L^*$ (Chiusura rispetto al concatenamento)
- $(L^*)^* = L^*$ (Idempotenza)
- $(L^R)^* = (L^*)^R$ (Commutatività della stella e della riflessione)

Casi Particolari

- $\emptyset^* = \{\epsilon\}$
- $\{\epsilon\}^* = \{\epsilon\}$

Esercizio 1.7

Definiamo il **linguaggio degli identificatori**.

- Consideriamo due alfabeti $\Sigma_1 = \{A, B, C, \dots, Z\}$ e $\Sigma_2 = \{0, 1, 2, \dots, 9\}$, con l'alfabeto generale $\Sigma = \Sigma_1 \cup \Sigma_2$.
- Un identificatore è una stringa che inizia con una lettera alfabetica (in Σ_1) e contiene un numero arbitrario di lettere e numeri (ma non è vuota).
- $L = \dots$

Esercizio 1.7

Definiamo il **linguaggio degli identificatori**.

- Consideriamo due alfabeti $\Sigma_1 = \{A, B, C, \dots, Z\}$ e $\Sigma_2 = \{0, 1, 2, \dots, 9\}$, con l'alfabeto generale $\Sigma = \Sigma_1 \cup \Sigma_2$.
- Un identificatore è una stringa che inizia con una lettera alfabetica (in Σ_1) e contiene un numero arbitrario di lettere e numeri (ma non è vuota).
- $L = \Sigma_1 \bullet (\Sigma_1 \cup \Sigma_2)^*$
- Se vogliamo limitare a 30 caratteri?

Esercizio 1.7

Definiamo il **linguaggio degli identificatori**.

- Consideriamo due alfabeti $\Sigma_1 = \{A, B, C, \dots, Z\}$ e $\Sigma_2 = \{0, 1, 2, \dots, 9\}$, con l'alfabeto generale $\Sigma = \Sigma_1 \cap \Sigma_2$.
- Un identificatore è una stringa che inizia con una lettera alfabetica (in Σ_1) e contiene un numero arbitrario di lettere e numeri (ma non è vuota).
- $L = \Sigma_1 \bullet (\Sigma_1 \cup \Sigma_2)^*$
- Se vogliamo limitare a 30 caratteri:

$$L = \Sigma_1 \bullet (\Sigma_1 \cup \Sigma_2 \cup \{\epsilon\})^{29}$$

Operatore CROCE**Chiusura Non Riflessiva al Concatenamento**

$$L^+ = \bigcup_{h=1}^{\infty} L^h = L^1 \cup L^2 \cup \dots$$

Proprietà

- $L^* = L^+ \cup \{\epsilon\}$
- $L^+ = L \bullet L^* = L^* \bullet L$

La stringa vuota ϵ compare in L^+ solo se è già in L .

Esercizio 1.8

Si consideri $L = \{\epsilon, aa\}$.

Che linguaggio è L^+ ?

Esercizio 1.8

Si consideri $L = \{\epsilon, aa\}$.

Che linguaggio è L^+ ?

$$L^+ = \{\epsilon, a^2, a^4, \dots\} = \{a^{2n} \mid \text{con } n \geq 0\}.$$

Operazioni sui Linguaggi

Operazioni sui Linguaggi

Operatori Insiemistici

- $L_1 \cup L_2$ Unione
- $L_1 \cap L_2$ Intersezione
- $L_1 - L_2$ Differenza

Confronti

- $L_1 \subseteq L_2$ Inclusione
- $L_1 \subset L_2$ Inclusione propria
- $L_1 = L_2$ Uguaglianza

Complemento

Dato l'alfabeto Σ ,

$$\neg L = \Sigma^* - L = \{x \mid x \in \Sigma^* \text{ AND } x \notin L\}.$$

Esercizio 1.9

Si consideri $L = \{a^{2n} \mid n \geq 0\}$, con $\Sigma = \{a\}$.

$$\neg L = \dots$$

Esercizio 1.9

Si consideri $L = \{a^{2n} \mid n \geq 0\}$, con $\Sigma = \{a\}$.
 $\neg L = \{a^{2n+1} \mid n \geq 0\}$

Esercizio 1.10

Si consideri $\Sigma_1 = \{a, b, c\}$ e $\Sigma_2 = \{a, b\}$.
 $(\Sigma_1)^* - (\Sigma_2)^* = \dots$

Operazioni sui Linguaggi

Esercizio 1.10

Si consideri $\Sigma_1 = \{a, b, c\}$ e $\Sigma_2 = \{a, b\}$.
 $(\Sigma_1)^* - (\Sigma_2)^* = (\Sigma_1)^* \bullet \{c\} \bullet (\Sigma_1)^*$
 insieme delle stringhe contenenti almeno un c .

Operazioni sui Linguaggi

Sostituzione

Si considerino due alfabeti Σ e Δ e i linguaggi $L \subseteq \Sigma^*$ (**Linguaggio Sorgente**) $L' \subseteq \Delta^*$ (**Linguaggio Pozzo**).
La sostituzione di L' al posto di un carattere $b \in \Sigma$ nella stringa $x = a_1a_2\dots a_h$ produce il linguaggio di alfabeto $(\Sigma - \{b\}) \cup \Delta$, così definita:

$$\begin{aligned}\phi_{b \rightarrow L'}(x) &= \\ &= \{y \mid x = a_1a_2\dots a_n \in L \text{ AND } y = c_1c_2\dots c_n \text{ AND } \\ &\quad \forall 1 \leq i \leq n (\text{IF } a_i \neq b \text{ THEN } c_i = a_i \text{ ELSE } c_i \in L')\}\end{aligned}$$

Se b e L' sono sottintesi, si indica $\phi(x)$.

Per l'intero linguaggio L , $\phi(L) = \cup_{(x \in L)}(\phi(x))$.

Esempio

Si considerino $\Sigma = \{a, b, c\}$ e $\Delta = \{x, y\}$.

$L = \{a, b, ac, cb\}$ e $L' = \{x, y, xy\}$.

- $\phi_{c \rightarrow L'}(ac) = \{ax, ay, axy\}$
- $\phi_{c \rightarrow L'}(cb) = \{xb, yb, xyb\}$
- $\phi(L) = \{a, b, ax, ay, axy, xb, yb, xyb\}$

Linguaggi Formali e Compilatori

Argomento 02:

Espressioni e Linguaggi Regolari

Prof. Giuseppe Psaila

Laurea Magistrale in Ingegneria Informatica
Università di Berga

Espressioni Regolarii

- Le **ESPRESSIONI REGOLARI** sono definite sfruttando gli operatori di **Unione**, **Concatenamento** e **Stella**.
- Dispongono di un meccanismo efficiente di riconoscimento: gli **AUTOMI A STATI FINITI**.

Espressioni Regolarii

Definizione: Linguaggio Regolare

- Un linguaggio di alfabeto $\Sigma = \{a_1, a_2, \dots, a_n\}$ è detto **REGOLARE** se può essere espresso mediante le operazioni di **Concatenamento**, **Unione** e **Stella** applicate un numero **Finito di Volte** ai linguaggi unitari $\{a_1\}, \{a_2\}, \dots, \{a_n\}$ e al linguaggio vuoto \emptyset .

Definizione: Espressione Regolare

- Un'**ESPRESSIONE REGOLARE** (abbreviata come e.r.) r

è una stringa costruita con i caratteri dell'alfabeto Σ , con i meta-simboli $\{\bullet, \cup, *, \emptyset\}$ e con le parentesi tonde.

Si suppone che i meta-simboli non facciano parte dell'alfabeto Σ .

Regole di Costruzione

- ① $r = \emptyset$
- ② $r = a$ dove $a \in \Sigma$
- ③ $r = (s \cup t)$ dove s e t sono e.r.
- ④ $r = (s \bullet t)$ dove s e t sono e.r.
o anche $r = (st)$
- ⑤ $r = (s)^*$ dove s è un'e.r.

Anche il simbolo ϵ può essere usato in una e.r., poiché vale l'identità $\epsilon = \emptyset^*$.

Precedenza degli operatori: $*$, \bullet , \cup .

Espressioni Regolari e Linguaggi

- Una espressione regolare **Definisce o Genera** un linguaggio.

Notazioni Comode

- $e^k = ee \dots e$ $k \geq 0$ volte
- $e^+ = ee^*$
- $[e]_k^h = e^k \cup e^{k+1} \cup \dots \cup e^h$
con $0 \leq k \leq h$
Ripetizione da k a h volte
- $[e] = [e]_0^1 = \epsilon \cup e$
Opzionalità

EsempioDato $\Sigma = \{0, 1\}$

- Struttura dei numeri binari di lunghezza arbitraria
 $e = (0 \cup 1)(0 \cup 1)^* = (0 \cup 1)^+$
- Struttura dei numeri binari di lunghezza fissa k
 $e = (0 \cup 1)^k$

Esercizio 2.1Dato $\Sigma = \{a, n\}$ (a indica una generica lettera alfabetica, n un generico carattere numerico),definire il **Linguaggio degli Identifieri**, dove un identificatore inizia con una lettera alfabetica e prosegue con una sequenza arbitraria di lettere e numeri.

- Di lunghezza non limitata

$$e = a(a \cup n)^*$$

- Di lunghezza massima k

$$e = a[a \cup n]_0^{k-1} \text{ oppure } e = a(a \cup n \cup \epsilon)^{k-1}$$

Sotto-Espressione (S.E.)

Una Sotto-Espressione (S.E.) di un'espressione regolare è così definita.

- e_k è una S.E. di $(e_1 \cup e_2 \cup \dots \cup e_k \cup \dots \cup e_j)$
- e_k è una S.E. di $(e_1 \bullet \dots \bullet e_k \bullet \dots \bullet e_j)$
- e è una S.E. di e^* , e^+ e di e^k
- ϵ è una S.E. di ogni espressione regolare

Per proseguire, definiamo il concetto di **Relazione di Implicazione**.

Relazione di Implicazione \Rightarrow

$e' \Rightarrow e''$ (e' implica e'') se le due espressioni si possono fattorizzare come

$$e' = \alpha\beta\gamma \quad \text{e} \quad e'' = \alpha\delta\gamma$$

dove α, β e γ sono S.E. di e'

e tra β e δ vale una delle seguenti relazioni.

β	δ
$(e_1 \cup \dots \cup e_k \cup \dots \cup e_h)$	e_k con $1 \leq k \leq h$
e^*	$ee \dots e$ $k \geq 0$ volte
e^+	$ee \dots e$ $k \geq 1$ volte
e^n	$ee \dots e$ n volte

- L'implicazione può essere **Applicata Transitivamente**

$e_0 \xrightarrow{n} e_n$ si dice che e_0 implica e_n in un numero n di passi, cioè

$$e_0 \Rightarrow e_1 \Rightarrow \dots \Rightarrow e_{n-1} \Rightarrow e_n$$

Se $n > 1$, l'Implicazione è detta **NON IMMEDIATA** o **TRANSITIVA**.

- Le scritture $e_0 \xrightarrow{*} e_n$ e $e_0 \xrightarrow{\neq} e_n$ indicano che e_n è ottenuta da e_0 in un numero di passi $n \geq 0$ (risp. $n > 0$).
Se $n = 0$, risulta $e_n = e_0$.

Linguaggio Generato da un'Espressione Regolare r

$$L(r) = \{x \in \Sigma^* \mid r \xrightarrow{*} x\}$$

con x **Priva di Meta-Simboli**.

Un linguaggio è detto **REGOLARE** se è **Generato da un'Espressione Regolare**.

Due espressioni regolari sono **EQUIVALENTI** se **Generano lo Stesso Linguaggio**.

Implicazione Sinistra

Un'**Implicazione** è detta **SINISTRA**

se, data $e' \Rightarrow e''$, e' e e'' si fattorizzano come

$$e' = \alpha\beta\gamma \quad \text{e} \quad e'' = \alpha\delta\gamma$$

dove α è il **pù lungo prefisso** comune ad e' e e'' **privo di meta-simboli**.

Altri Operatori

- **Complemento**

 $\neg e$

$$L(\neg e) = \Sigma^* - L(e)$$

- **Intersezione**

 $e_1 \cap e_2$

$$L(e_1 \cap e_2) = L(e_1) \cap L(e_2)$$

Esempio

- $(a^* \cup b^+) \Rightarrow a^*$
 $(a^* \cup b^+) \Rightarrow b^+$
- $(a^* \cup b^+) \Rightarrow a^* \Rightarrow \epsilon$, quindi $(a^* \cup b^+) \xrightarrow{?} \epsilon$
 generalizzabile in $(a^* \cup b^+) \xrightarrow{+} \epsilon$
- $(a^* \cup b^+) \Rightarrow b^+ \Rightarrow bb$, quindi $(a^* \cup b^+) \xrightarrow{?} bb$
 generalizzabile come $(a^* \cup b^+) \xrightarrow{+} bb$

Esempio

- $(ab)^* \Rightarrow abab$
- $(ab \cup c) \Rightarrow ab$
- $a(ba \cup c)^*d \Rightarrow ad$
- $a(ba \cup c)^*d \Rightarrow a(ba \cup c)(ba \cup c)d$
- $a^*(b \cup c \cup d)f^+ \Rightarrow aa(b \cup c \cup d)f^+$
- $a^*(b \cup c \cup d)f^+ \Rightarrow a^*cf^+ \Rightarrow aaacf^+ \Rightarrow aaacf$

Sia Θ un operatore che, applicato ad uno o due linguaggi, ne produce un altro.

Una famiglia di linguaggi si dice **CHIUSA** rispetto ad un operatore Θ se il linguaggio risultante appartiene alla stessa famiglia dei linguaggi cui Θ viene applicato.

- La famiglia dei **Linguaggi Regolari** è **Chiusa** rispetto agli operatori **Concatenamento**, **Unione** e **Stella**.

Proprietà

La famiglia dei Linguaggi Regolari è la **PIÙ PICCOLA FAMIGLIA** di linguaggi che:

- ① contiene **Tutti i Linguaggi Finiti**;
- ② è **Chiusa** rispetto a **Concatenamento. Unione e Stella.**

Inoltre, la famiglia dei Linguaggi Regolari è chiusa rispetto a **Complemento, Intersezione e Ripetizione**.

Automa a Stati Finiti Deterministico

$$M = (Q, \Sigma, \delta, q_0, F)$$

- Q : Insieme degli Stati
- Σ : alfabeto di ingresso
- δ : Funzione di Transizione
 $\delta : Q \times \Sigma \rightarrow Q$
- $q_0 \in Q$ Stato Iniziale
- $F \subseteq Q$: Stati Finali

Automi a Stati Finiti

Funzione di Transizione Transitiva

$$\delta^*(q, ya) = \delta(\delta^*(q, y), a)$$

$$\text{con } \delta^*(q, \epsilon) = q.$$

Una stringa $x \in L(r)$ (dove r è un'espressione regolare riconosciuta dall'automa) se

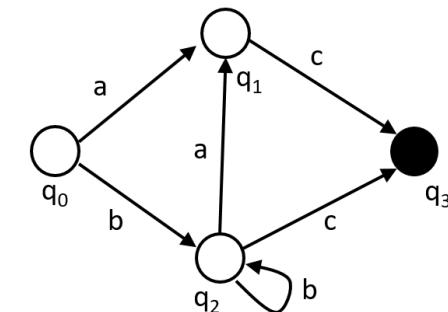
$$\delta^*(q_0, x) = q \text{ con } q \in F.$$

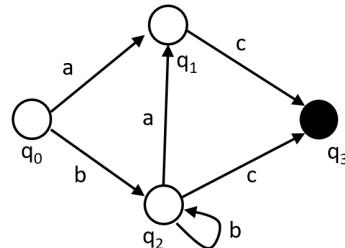
Automi a Stati Finiti

$$\Sigma = \{a, b, c\}$$

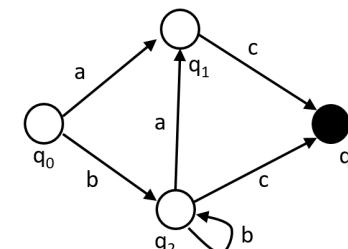
$$Q = \{q_0, q_1, q_2, q_3\}$$

$$F = \{q_3\}$$



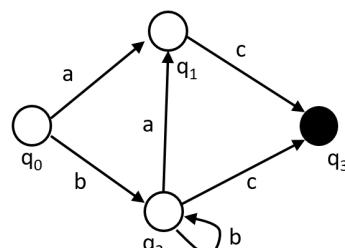
Esecuzione Corretta

Input	Stato	Decisione
<i>bbac</i>	q_0	Leggi
<i>bac</i>	q_2	Leggi
<i>ac</i>	q_2	Leggi
<i>c</i>	q_1	Leggi
	q_3	OK

Esecuzione Scorretta

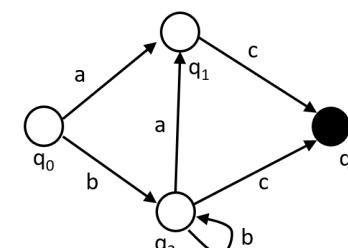
Input	Stato	Decisione
<i>acc</i>	q_0	Leggi
<i>cc</i>	q_1	Leggi
<i>c</i>	q_3	Errore

La stringa in Input **non è vuota**

Esecuzione Scorretta

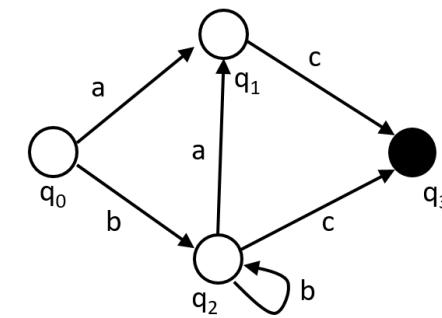
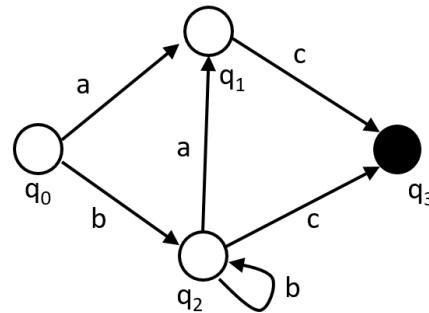
Input	Stato	Decisione
<i>abc</i>	q_0	Leggi
<i>bc</i>	q_1	Errore

Non esistono mosse uscenti da q_1 con b

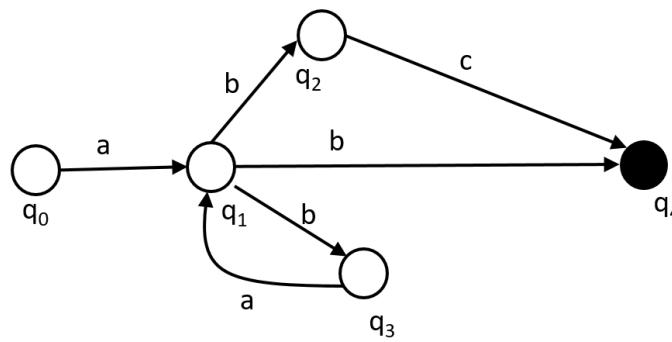
Esecuzione Scorretta

Input	Stato	Decisione
<i>ba</i>	q_0	Leggi
<i>a</i>	q_2	Leggi
	q_1	Errore

La stringa in Input è finita, ma l'Automa **non è in uno stato finale**

Esercizio 2.2: Espressione Regolare?**Esercizio 2.2: Espressione Regolare?**

$$\begin{aligned}
 r_1 &= (b^+ (c \cup ac)) \cup ac \\
 r_2 &= ((b^+(\epsilon \cup a)) \cup a)c & r_5 &= (b^+[a] \cup a)c \\
 r_3 &= [b^+][a]c & r_4 &= b^+[a]c
 \end{aligned}$$

Automa Non Deterministico**Automa a Stati Finiti Non Deterministico senza ϵ -Mosse**

$$M = (Q, \Sigma, \delta, q_o, F)$$

- Q : Insieme degli Stati
- Σ : alfabeto di ingresso
- δ : Funzione di Transizione Non-Deterministica
 $\delta : Q \times \Sigma \rightarrow (2^Q - \{\emptyset\})$
- $q_o \in Q$ Stato Iniziale
- $F \subseteq Q$: Stati Finali

Insieme delle Parti

Dato un insieme Q , l'insieme delle sue parti 2^Q è

I'Insieme di Tutti i Suoi SottoInsiemi

Si noti che $|2^Q| = 2^{|Q|}$

Esempio

Se $Q = \{a, b, c\}$

$$2^Q = \{\{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}, \emptyset\}$$

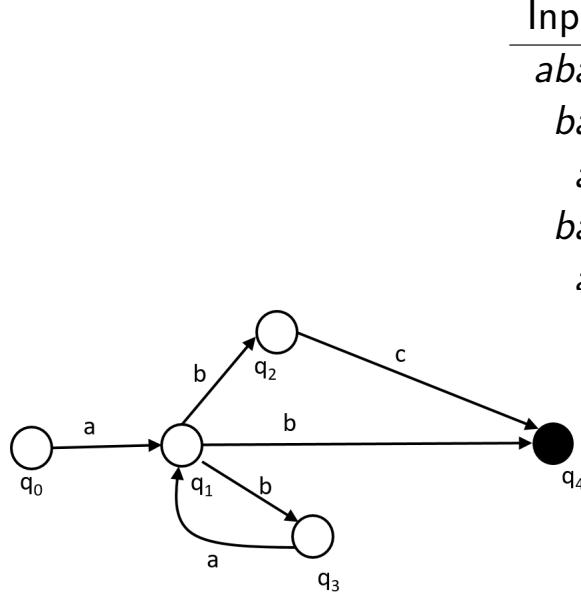
Funzione di Transizione Transitiva Non Deterministica senza ϵ -Mosse

$$\delta^*(q, ya) = \{p \mid \exists r \in \delta^*(q, y) \text{ AND } p \in \delta(r, a)\}$$

con $\delta^*(q, \epsilon) = \{q\}$.

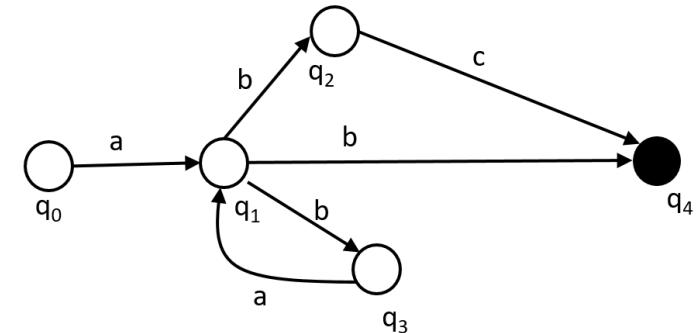
Una stringa $x \in L(r)$ (dove r è un'espressione regolare riconosciuta dall'automa) se
 $\exists q \in F$ tale che $q \in \delta^*(q_0, x)$.

Automi a Stati Finiti

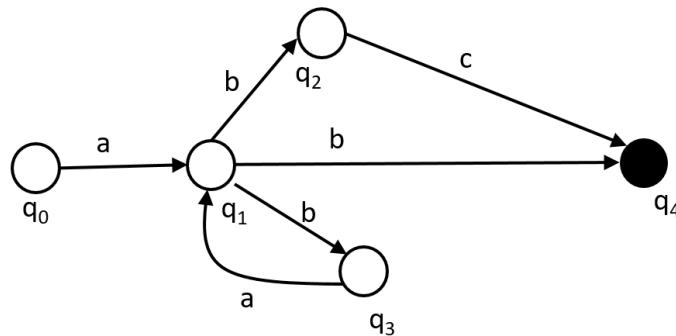
Automa Non Deterministico

Input	Stato	Decisione
abab	q_0	Leggi
bab	q_1	Scegli q_2
ab	q_2	BackTrack
bab	q_1	Scegli q_3
ab	q_3	Leggi
b	q_1	Scegli q_4
	q_4	OK

Automi a Stati Finiti

Esercizio 2.3: Espressione Regolare?

Esercizio 2.3: Espressione Regolare?

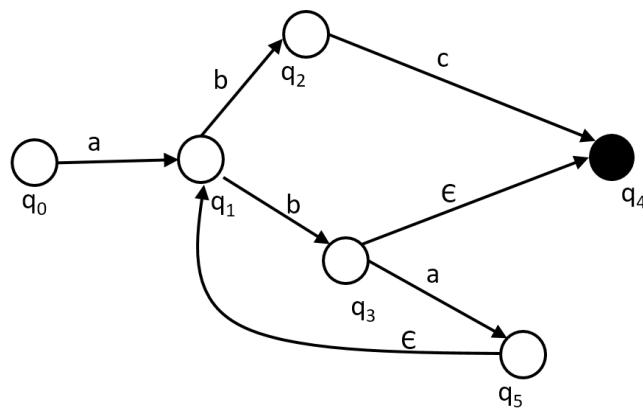


$$r_1 = a(bc \cup (b(ab)^*[c]))$$

$$r_2 = a((ba)^*b[c])$$

In opposizione, le altre mosse vengono dette **NON SPONTANEE**.

Automa Non Deterministico con Epsilon Mosse



Automa a Stati Finiti Non Deterministico con ϵ -Mosse

$$M = (Q, \Sigma, \delta, q_o, F)$$

- Q : Insieme degli Stati
- Σ : alfabeto di ingresso
- δ : Funzione di Transizione Non-Deterministica
 $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow (2^Q - \{\emptyset\})$
- $q_0 \in Q$ Stato Iniziale
- $F \subseteq Q$: Stati Finali

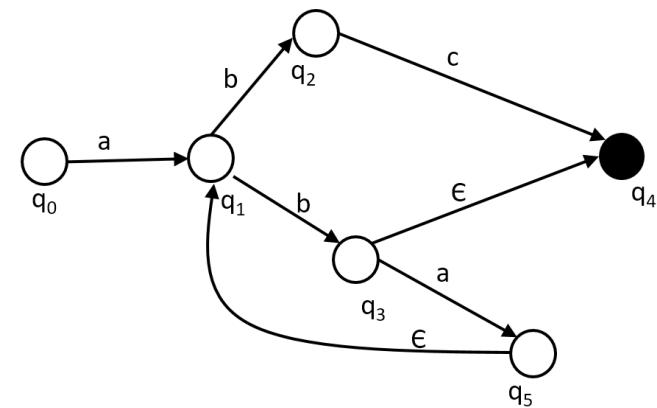
Funzione di Transizione Transitiva Non Deterministica con ϵ -Mosse

$$\delta^*(q, ya) = \{p \mid \exists r \in \delta^*(q, y) \text{ AND } p \in \delta(r, a)\}$$

con $\delta^*(q, \epsilon) = \{q\}$ se $\delta(q, \epsilon)$ non è definita, dove $a \in \Sigma \cup \{\epsilon\}$.

Una stringa $x \in L(r)$ (dove r è un'espressione regolare riconosciuta dall'automa) se $\exists q \in F$ tale che $q \in \delta^*(q_0, x)$.

Automa Non Deterministico con Essilon Mosse



$$r_1 = a(bc \cup (b(ab)^*[c]))$$

Esercizio 2.4: Trasformiamo l'Espressione Regolare

$$r_1 = a(bc \cup (b(ab)^*[c]))$$

$$r_2 =$$

Esercizio 2.4: Trasformiamo l'Espressione Regolare

$$r_1 = a(bc \cup (b(ab)^*[c]))$$

$$r_2 = a(b(c \cup (ab)^*[c]))$$

$$r_3 = a(b(\epsilon \cup (ab)^*)[c])$$

$$r_4 = ab(ab)^*[c]$$

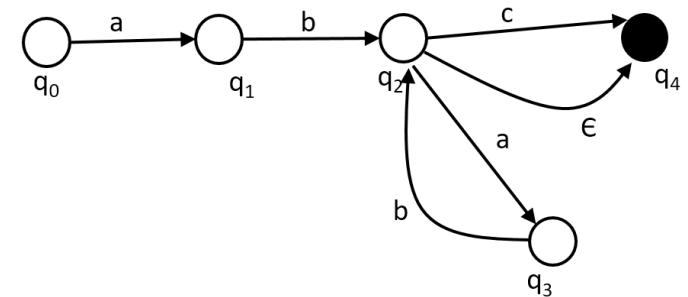
Esercizio 2.5**Automa Non Deterministico con Epsilon Mosse**

$$r_4 = ab(ab)^*[c]$$

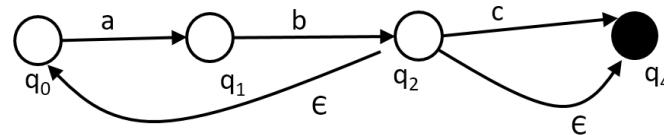
?

Esercizio 2.5**Automa Non Deterministico con Epsilon Mosse**

$$r_4 = ab(ab)^*[c]$$

**Automa Non Deterministico con Epsilon Mosse**
 $r_4 = ab(ab)^*[c]$ è equivalente a

$$r_5 = (ab)^+[c]$$

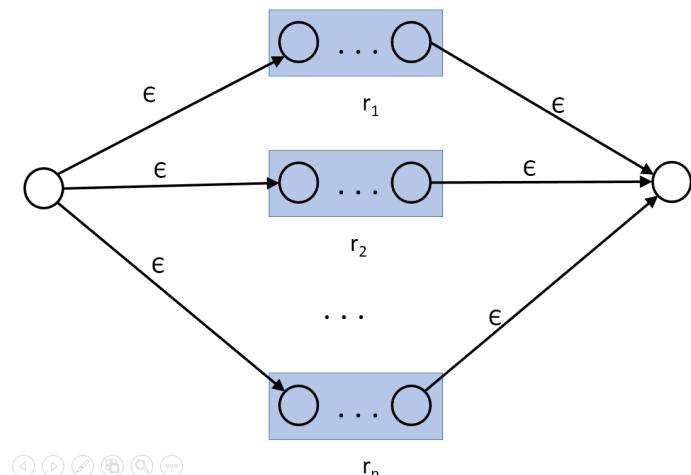
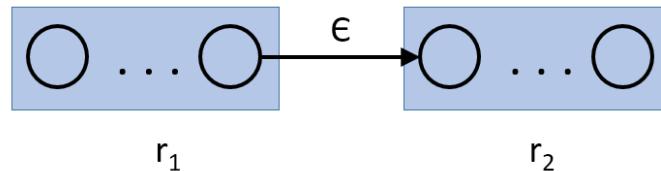
**Utilità delle EPsiilon-Mosse**

- Le ϵ -mosse servono per costruire automi
- a partire da espressioni regolari di qualsiasi complessità
- Perché consentono di comporre liberamente gli automi derivati dalle S.E.

Composizione: Alternativa
 $r = (r_1 \cup r_2 \cup \dots \cup r_n)$

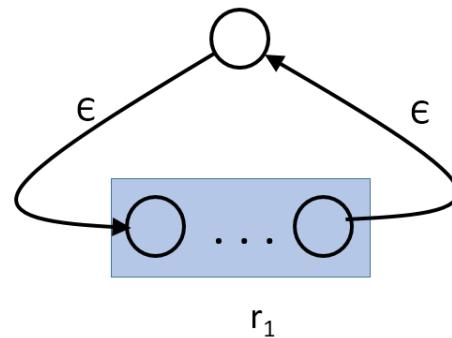
Composizione: Concatenamento

$$r = r_1 \bullet r_2$$



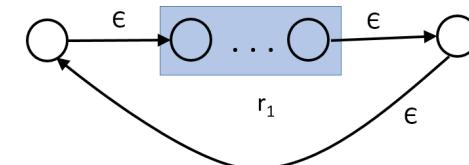
Composizione: Stella

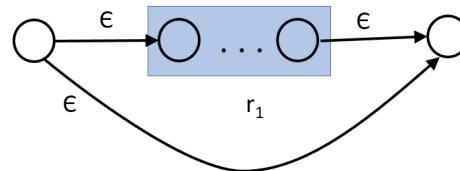
$$r = (r_1)^*$$



Composizione: Croce

$$r = (r_1)^+$$



Composizione: Opzionalità $r = [r_1]$ **Linguaggio dei Numeri in Base 10**

$$\Sigma = \{d, '+', '-\}, ", "\}$$

$$i = ['+' \cup '-'] d^+$$

$$v = ['+' \cup '-'] (d^+, , d^+)$$

$$r = (i \cup v)$$

Semplifichiamo l'espressione

$$r_s =$$

Linguaggio dei Numeri in Base 10

$$\Sigma = \{d, '+', '-\}, ", "\}$$

$$i = ['+' \cup '-'] d^+$$

$$v = ['+' \cup '-'] (d^+, , d^+)$$

$$r = (i \cup v)$$

Semplifichiamo l'espressione

$$r_1 = (['+' \cup '-'] d^+ \cup ['+' \cup '-'] (d^+, , d^+))$$

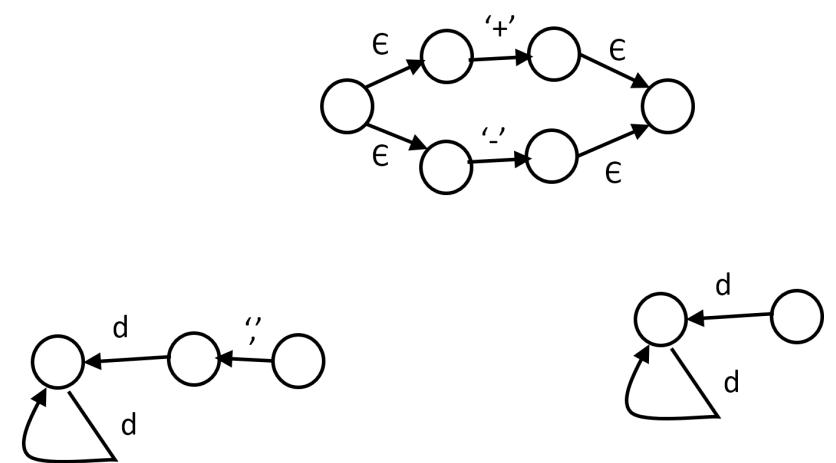
$$r_2 = ['+' \cup '-'] (d^+ \cup (d^+, , d^+))$$

$$r_3 = ['+' \cup '-'] d^+ (\epsilon \cup , , d^+)$$

$$r_s = ['+' \cup '-'] d^+ [, , d^+]$$

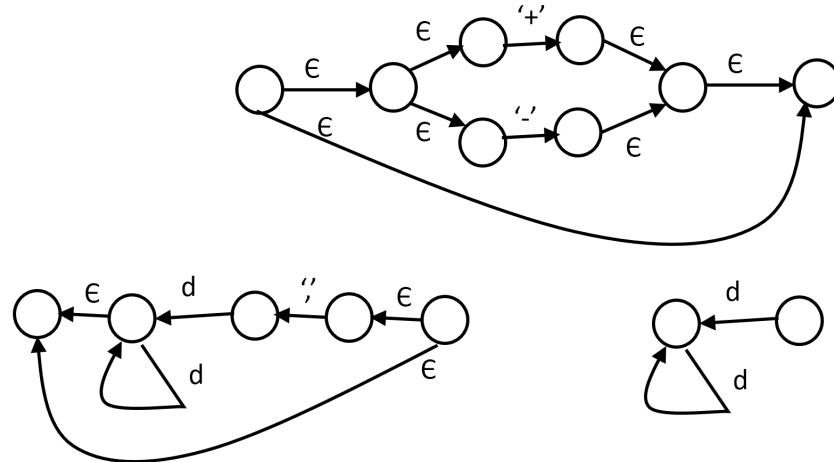
Componiamo l'Automa per il Linguaggio dei Numeri in Base 10

$$r_s = ['+' \cup '-'] d^+ [, , d^+]$$

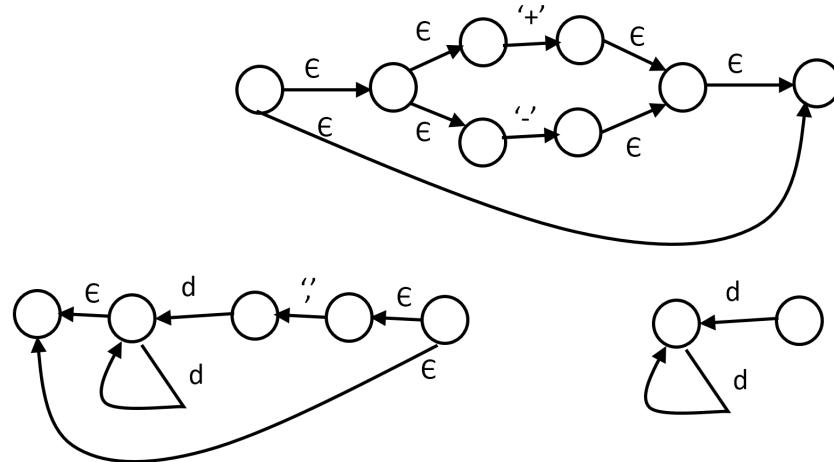


Componiamo l'Automa per il Linguaggio dei Numeri in Base 10

$$r_s = [\text{'} + \text{' } \cup \text{'} - \text{' }] d^+ [\text{' , } , d^+]$$



$$r_s = [\text{'} + \text{' } \cup \text{'} - \text{' }] d^+ [\text{' , } , d^+]$$



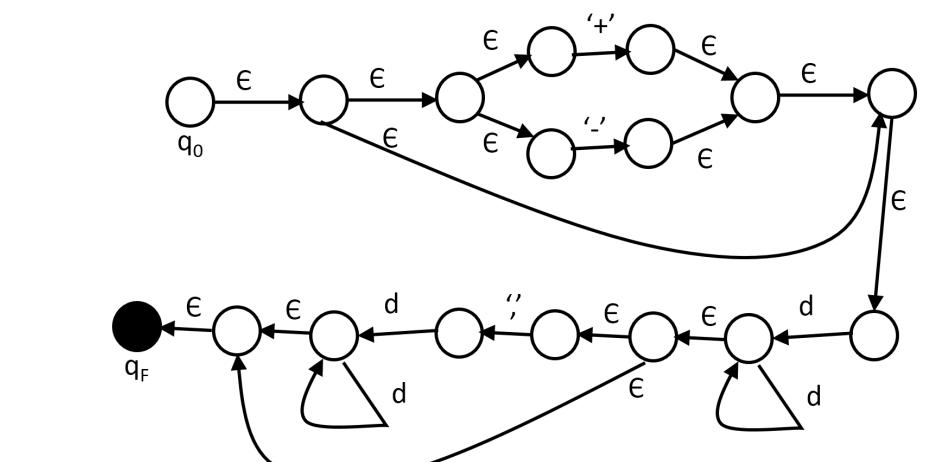
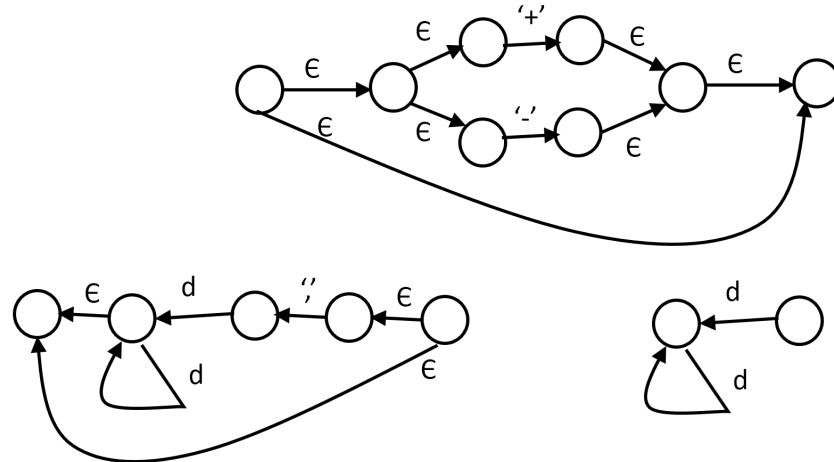
Complemento

Dato un nautoma finito **Deterministico** M che riconosce il linguaggio L , derivare l'automa deterministico \overline{M} che riconosce il linguaggio $\neg L$.

Esiste un algoritmo che deriva l'automa a stati finiti deterministico \overline{M} dall'automa a stati finiti deterministico M .

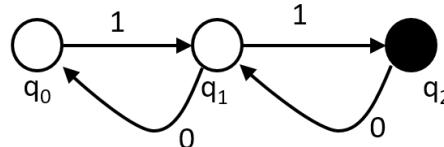
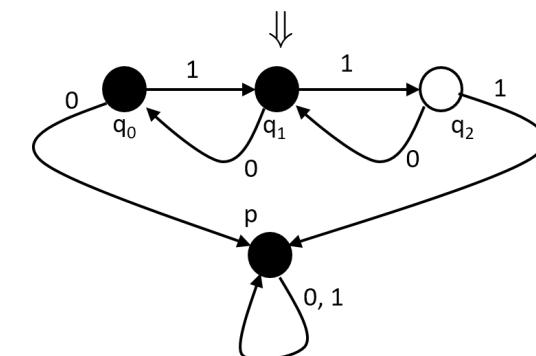
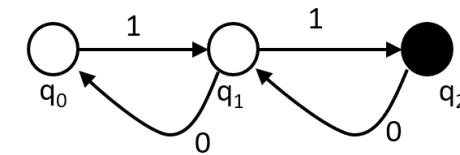
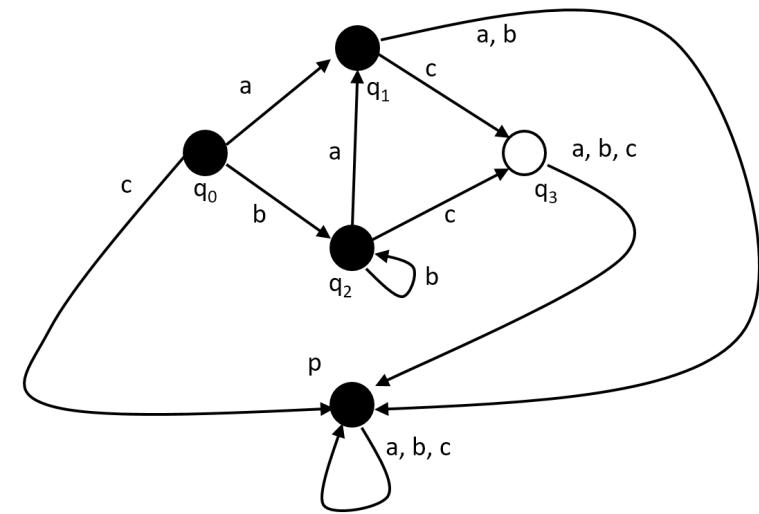
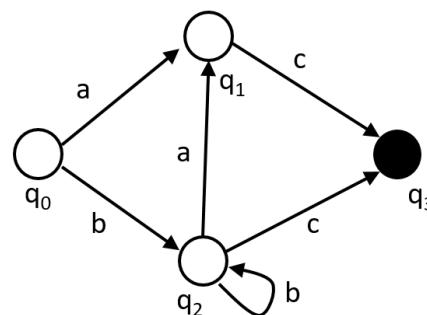
Componiamo l'Automa per il Linguaggio dei Numeri in Base 10

$$r_s = [\text{'} + \text{' } \cup \text{'} - \text{' }] d^+ [\text{' , } , d^+]$$



Algoritmo Complemento

- Sia $M(Q, \Sigma, \delta, q_0, F)$ l'automa deterministico per L .
- Sia $\overline{M} = (\overline{Q}, \Sigma, \overline{\delta}, q_0, \overline{F})$ la'utoma per $\neg L$ da calcolare.
- **(1)** Aggiungere agli stati Q uno stato **pozzo** p
 $\overline{Q} = Q \cup \{p\}$.
- **(2)** Per ogni $q \in Q$ e per ogni $a \in \Sigma$,
 $\overline{\delta}(q, a) = \delta(q, a)$ se $\delta(q, a)$ è definita
 $\overline{\delta}(q, a) = p$ altrimenti.
- **(3)** Per ogni $a \in \Sigma$, $\overline{\delta}(p, a) = p$.
- **(4)** $\overline{F} = (Q - F) \cup \{p\}$.

Alfabeto $\Sigma = \{0, 1\}$ **Calcolare l'Automa Complemento di**Alfabeto $\Sigma = \{0, 1\}$ **Automa Complemento****Calcolare l'Automa Complemento di**

Algoritmo per Eliminare le ϵ -mosse

Dato l'automa $M(Q, \Sigma, \delta, q_0, F)$ con ϵ -mosse, vogliamo ottenere l'automa $M'(Q', \Sigma, \delta', q_0, F')$ senza ϵ -mosse

- **Passo 1**

Inserire lo stato q_0 in Q' e in N (insieme dei nuovi stati)

Algoritmo per Eliminare le ϵ -mosse

- **Passo 2**

Impostiamo l'insieme $N' = \emptyset$

Per ogni stato $q \in N$,

copiare da δ tutte le mosse non spontanee che escono da q , cioè $\delta(q, a) = \bar{q}$, creando $\delta'(q, a) = \bar{q}$, aggiungendo \bar{q} in Q' e in N' , se non è già presente in Q' .

Algoritmo per Eliminare le ϵ -mosse

- **Passo 3**

Per ogni stato $q \in N$,

cercare tutte le transizioni transitive $q \xrightarrow{*} \bar{q}$ che contengono una sola mossa non spontanea preceduta e/o seguita da mosse spontanee,

(cioè $q \xrightarrow{\epsilon} \dots \xrightarrow{\epsilon} q_i \xrightarrow{a} \bar{q}$,

$q \xrightarrow{a} q_i \xrightarrow{\epsilon} \dots \xrightarrow{\epsilon} \bar{q}$,

$q \xrightarrow{\epsilon} \dots \xrightarrow{\epsilon} q_i \xrightarrow{a} q_j \xrightarrow{\epsilon} \dots \xrightarrow{\epsilon} \bar{q}$).

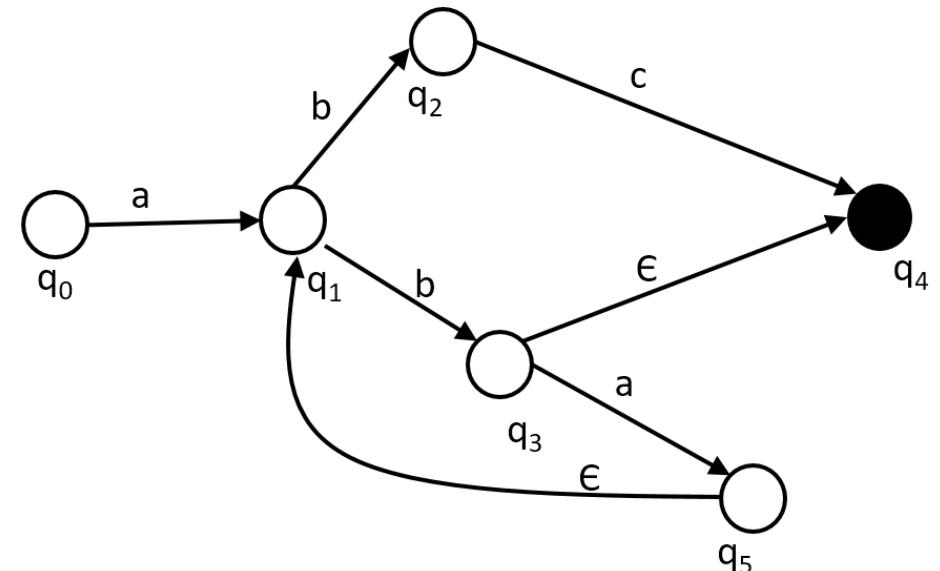
Creare la corrispondente $\delta'(q, a) = \bar{q}$, aggiungendo \bar{q} in Q' e in N' , se non è già presente in Q' .

ATTENZIONE: gli stati \bar{q} da considerare per i pattern

- $q \xrightarrow{a} q_i \xrightarrow{\epsilon} \dots \xrightarrow{\epsilon} \bar{q}$
- $q \xrightarrow{\epsilon} \dots \xrightarrow{\epsilon} q_i \xrightarrow{a} q_j \xrightarrow{\epsilon} \dots \xrightarrow{\epsilon} \bar{q}$

sono tutti e solo gli stati \bar{q} dai quali non escono mosse oppure esce almeno una mossa non spontanea

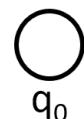
(non ci si ferma sugli stati nel mezzo di catene di ϵ dai quali non escono mosse non spontanee, perché introdurrebbero inutile ridondanza).

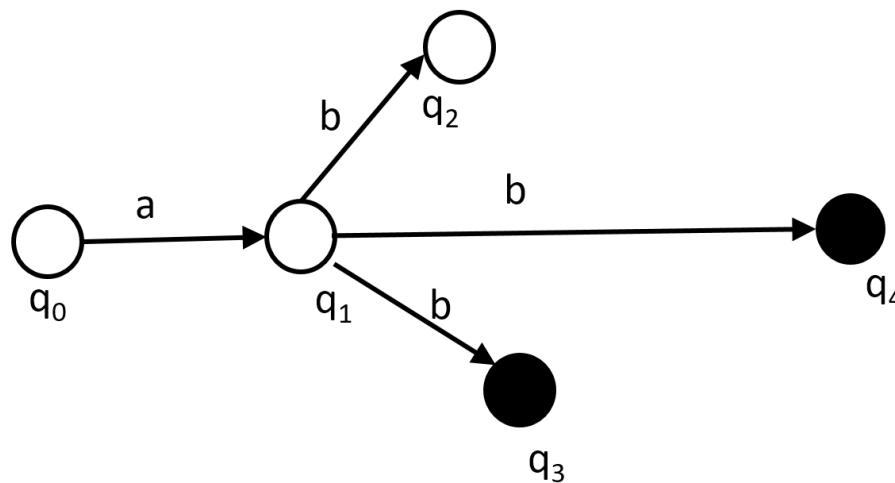
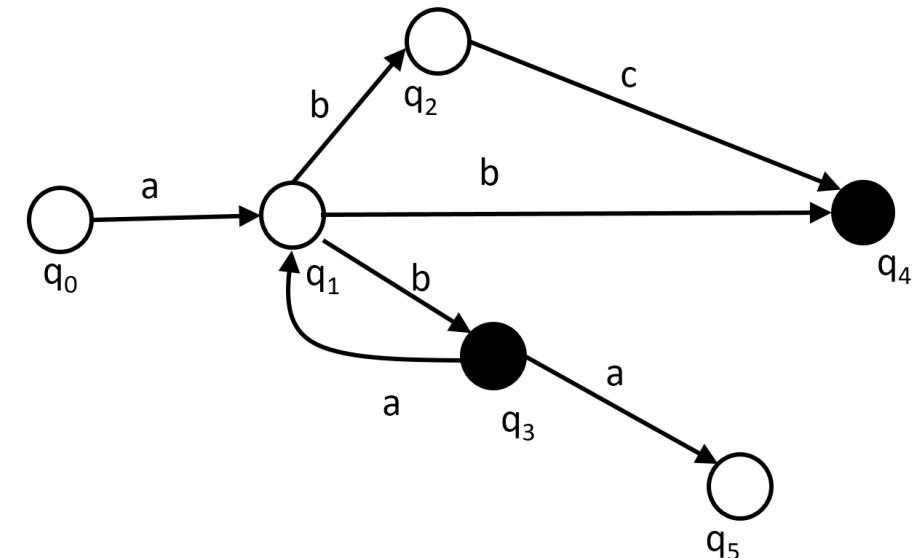
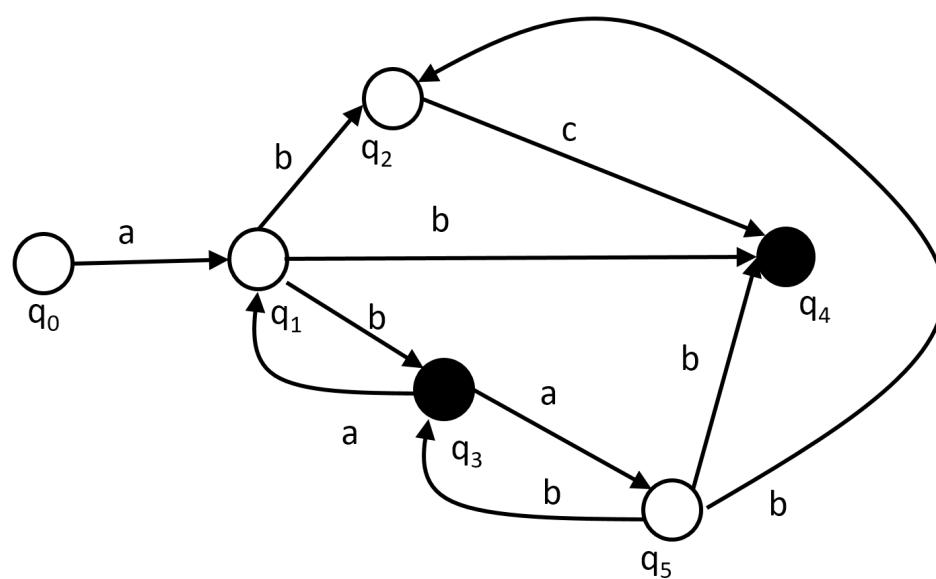
Eliminazione delle ϵ -Mosse: M **Algoritmo per Eliminare le ϵ -mosse****• Passo 4**

Per ogni stato $q \in N$,
se esiste un percorso che va da q ad uno stato \bar{q}
composto solo di ϵ -mosse, con $\bar{q} \in F$,
inserire q in F' (stato finale di M').

• Passo 5

Se $N' \neq \emptyset$, $N = N'$ e ripartire dal Passo 2,
altrimenti terminare

Eliminazione delle ϵ -Mosse: M' 

Eliminazione delle ϵ -Mosse: M' **Eliminazione delle ϵ -Mosse: M'** **Eliminazione delle ϵ -Mosse: M'** **Algoritmo per Eliminare il Non-Determinismo**

Dato l'automa $M(Q, \Sigma, \delta, q_0, F)$ Non-Deterministico senza ϵ -mosse, vogliamo ottenere l'automa $M'(Q', \Sigma, \delta', q_0, F')$ Deterministico

• Passo 1

Inserire lo stato q_0 in Q' e in N (insieme dei nuovi stati)

Algoritmo per Eliminare il Non-Determinismo**• Passo 2**

Impostiamo l'insieme $N' = \emptyset$

Per ogni stato $q \in N$ che compare anche in Q ,
per ogni simbolo $a \in \Sigma$ per cui

$\delta(q, a) = \{q_1, q_2, \dots, q_n\}$ non vuoto

- se $n > 1$, creare uno stato **collettivo** $[q_1, q_2, \dots, q_n]$ e aggiungere la mossa $\delta'(q, a) = [q_1, q_2, \dots, q_n]$; se $[q_1, q_2, \dots, q_n]$ non già in Q' , inserirlo in Q' e in N' ;
- se $n = 1$, lo stato collettivo diventa uno stato semplice e lo si aggiunge a Q' e a N' se non è già in Q' .

Algoritmo per Eliminare il Non-Determinismo**• Passo 3**

Per ogni stato collettivo $[q_1, q_2, \dots, q_n] \in N$,
per ogni stato $q_i \in [q_1, q_2, \dots, q_n]$ e ogni simbolo $a \in \Sigma$

calcolare il nuovo stato collettivo

$[\delta(q_1, a) \cup \delta(q_2, a) \cup \dots \cup \delta(q_n, a)]$

- aggiungere questo stato in Q' e in N' se non è già in Q' ;

- se il nuovo stato collettivo contiene un solo stato, diventa uno stato semplice.

- Definire $\delta'([q_1, q_2, \dots, q_n], a) =$
 $[\delta(q_1, a) \cup \delta(q_2, a) \cup \dots \cup \delta(q_n, a)]$

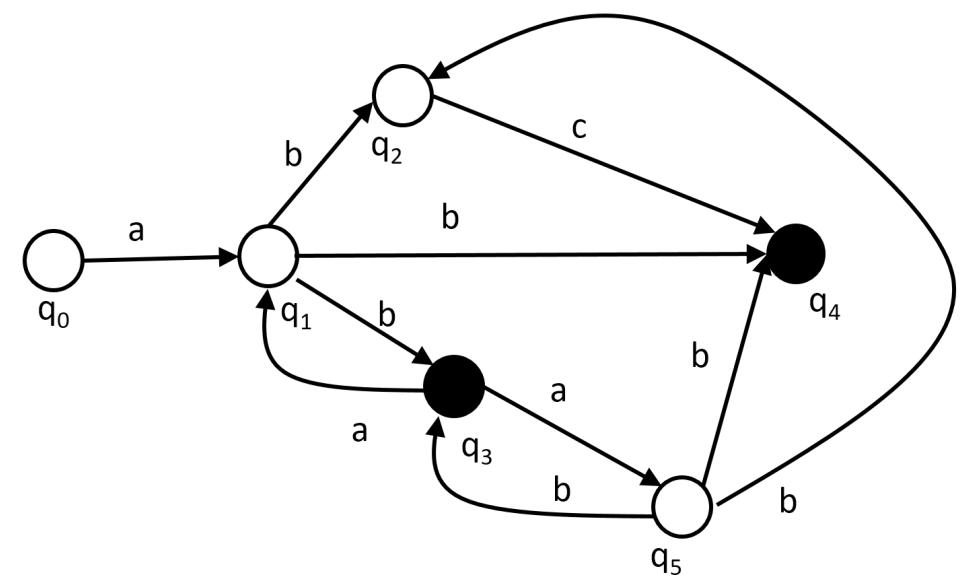
Algoritmo per Eliminare il Non-Determinismo**• Passo 4**

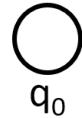
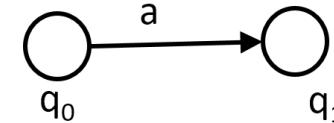
Per ogni stato $q = [q_1, q_2, \dots, q_n] \in N$ (anche con $n = 1$,

se almeno uno stato $q_i \in q$ è finale in M (cioè $q_i \in F$), inserire q negli stati finali di M' , cioè $q \in F'$.

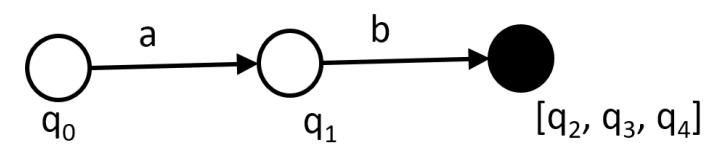
• Passo 5

Se $N' \neq \emptyset$, $N = N'$ e ripartire dal Passo 2,
altrimenti terminare

Eliminazione Non-Determinismo: M 

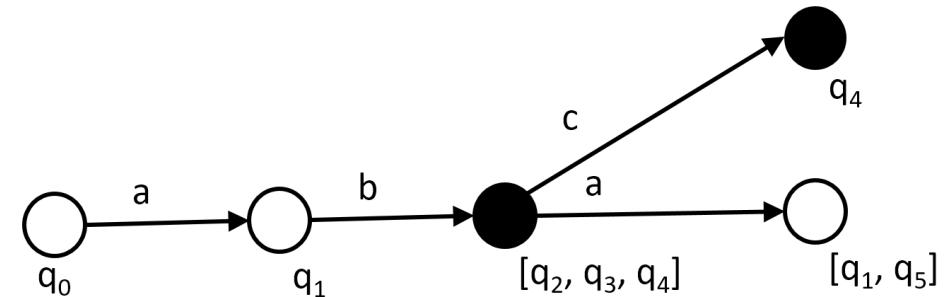
Eliminazione Non-Determinismo: M' **Eliminazione Non-Determinismo: M'** **Eliminazione Non-Determinismo: M'**

$$\frac{q_1 \xrightarrow{a} \emptyset}{\begin{array}{l} q_1 \xrightarrow{b} \{q_2, q_3, q_4\} \\ q_1 \xrightarrow{c} \emptyset \end{array}}$$

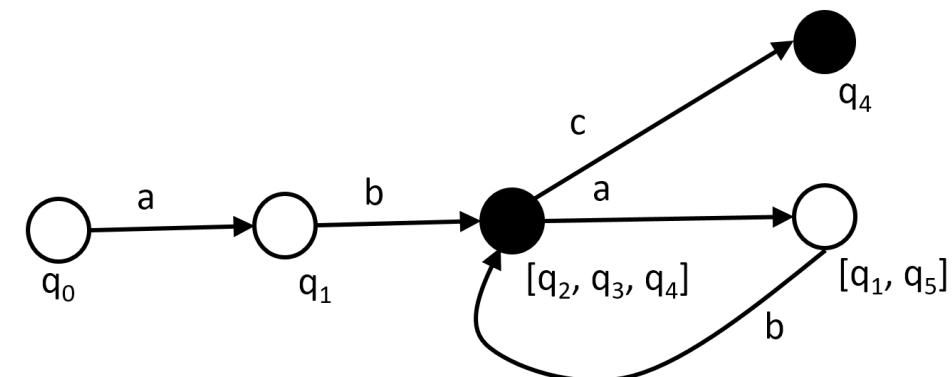
Eliminazione Non-Determinismo: M' 

Eliminazione Non-Determinismo: M' **Eliminazione Non-Determinismo: M'**

$q_2 \xrightarrow{a} \emptyset$	$q_2 \xrightarrow{b} \emptyset$	$q_2 \xrightarrow{c} \{q_4\}$
$q_3 \xrightarrow{a} \{q_1, q_5\}$	$q_3 \xrightarrow{b} \emptyset$	$q_3 \xrightarrow{c} \emptyset$
$q_4 \xrightarrow{a} \emptyset$	$q_4 \xrightarrow{b} \emptyset$	$q_4 \xrightarrow{c} \emptyset$

**Eliminazione Non-Determinismo: M'** **Eliminazione Non-Determinismo: M'**

$q_1 \xrightarrow{a} \emptyset$	$q_1 \xrightarrow{b} \{q_2, q_3, q_4\}$	$q_1 \xrightarrow{c} \emptyset$
$q_5 \xrightarrow{a} \emptyset$	$q_5 \xrightarrow{b} \{q_2, q_3, q_4\}$	$q_5 \xrightarrow{c} \emptyset$



Linguaggi Formali e Compilatori

Argomento 03:

Grammatiche BNF

Prof. Giuseppe Psaila

Laurea Magistrale in Ingegneria Informatica
Università di Berga

- Le espressioni regolari hanno una significativa limitazione:
non sono in grado di gestire gli ANNIDAMENTI e le STRUTTURE PARENTETICHE
- Per questo, occorre rivolgersi a famiglie di linguaggi più ampie
- Queste famiglie sono descritte dalle **Grammatiche BNF** (Backus-Naur Form):
Grammatiche Generative Libere dal Contesto (Non Contestuali)

Grammatiche BNF

Grammatiche BNF

Definizione: Grammatica BNF

Una Grammatica BNF è descritta da una tupla

$G(V, \Sigma, P, S)$

- V : alfabeto **Non-Terminale** (insieme dei simboli **Non-Terminali**)
- Σ : alfabeto **Terminale** (insieme dei simboli **Terminali**)
- P : insieme delle **Regole di Produzione** (vedi dopo)
- $S \in V$: l'**Assioma**

Regola di Produzione

- Una regola $p \in P$ è una **coppia ordinata** (X, α)
- dove $X \in V$
- e $\alpha \in (V \cup \Sigma)^*$
- La regola (X, α) viene scritta più frequentemente come

$$X \rightarrow \alpha$$
- X è la **Parte Sinistra** della regola
 α è la **Parte Destra** della regola

Esempio: Numeri binari che iniziano con 1

- $\Sigma = \{0, 1\}$
- $V = \{S, N\}$
- Regole di Produzione

$$S \rightarrow 1 \ N$$

$$N \rightarrow 1 \ N$$

$$N \rightarrow 0 \ N$$

$$N \rightarrow \epsilon$$

- Assioma: S

Esempio: Linguaggio degli Identifieri

- $\Sigma = \{a, n, _\}$
- $V = \{S, L\}$
- Regole di Produzione

$$S \rightarrow a \ L$$

$$L \rightarrow a \ L$$

$$L \rightarrow n \ L$$

$$L \rightarrow _ \ L$$

$$L \rightarrow \epsilon$$
- Assioma: S

Esempio: Linguaggio delle Liste Annidate

- $\Sigma = \{n, ", , (,)\} \quad V = \{S, L, F, N, E\}$ Axiom: S
- $S \rightarrow L$
- $L \rightarrow (F)$
- $F \rightarrow \epsilon$
- $F \rightarrow E \ N$
- $N \rightarrow \epsilon$
- $N \rightarrow " , " \ E \ N$
- $E \rightarrow n$
- $E \rightarrow L$
- Esempio: $(1, (2, 3), ((4)))$

Derivazioni

- Consideriamo due stringhe $\beta, \gamma \in (V \cup \Sigma)^*$
- Si dice che β **diviene** γ (o che γ **deriva da** β) per la grammatica G (scritto $\beta \xrightarrow{G} \gamma$ o, se G è sottintesa, $\beta \rightarrow \gamma$)
- se le stringhe β e γ si fattorizzano come

$$\beta = \eta A \delta \quad \text{e} \quad \gamma = \eta \alpha \delta$$
- e $A \rightarrow \alpha \in P$ è una regola in G .
- Si dice anche che α **si riduce** ad A .

Derivazioni

- Una **catena di derivazioni** di lunghezza $n \geq 0$ è
 $\beta_0 \rightarrow \beta_1 \rightarrow \dots \rightarrow \beta_n$
 e si scrive
 $\beta_0 \xrightarrow{n} \beta_n$
 (per $n = 0$, ogni stringa deriva da sé stessa)
- Si usano anche le scritture:
 $\beta_0 \xrightarrow{*} \beta_n \quad \text{lunghezza } n \geq 0$
 $\beta_0 \xrightarrow{+} \beta_n \quad \text{lunghezza } n > 0$

Linguaggio Generato

- Il **Linguaggio Generato da G Partendo dal non-terminale A** è
 $L_A(G) = \{x \in \Sigma^* \mid A \xrightarrow{+} x\}$
- Se si considera l'assioma S , il linguaggio generato da G è
 $L(G) = L_S(G) = \{x \in \Sigma^* \mid S \xrightarrow{+} x\}$

Esercizio 3.01: Linguaggio degli Identifieri

- $\Sigma = \{a, n, _\}$ $V = \{S, L\}$ Assioma: S
- Regole di Produzione

$$\begin{aligned} S &\rightarrow a \ L \\ L &\rightarrow a \ L \\ L &\rightarrow n \ L \\ L &\rightarrow _ \ L \\ L &\rightarrow \epsilon \end{aligned}$$
- Derivazione per aan_n ?

Esercizio 3.01: Linguaggio degli Identifieri

- Derivazione per aan_n ?
- $$\begin{aligned} S &\rightarrow aL \rightarrow aaL \rightarrow aanL \rightarrow aan_L \rightarrow aan_nL \rightarrow \\ &\quad aan_n\epsilon \end{aligned}$$

Esercizio 3.02: Linguaggio delle Liste Annidate

- $\Sigma = \{n, ", (,)\}$ $V = \{S, L, F, N, E\}$ Assioma: S
- $S \rightarrow L$
 $L \rightarrow (F)$
 $F \rightarrow \epsilon$
 $F \rightarrow E\ N$
 $N \rightarrow \epsilon$
 $N \rightarrow ",\ E\ N$
 $E \rightarrow n$
 $E \rightarrow L$
- Derivazione per $(1, (2, 3), ((4)))$?

Esercizio 3.02: Linguaggio delle Liste Annidate

- Derivazione per $(1, (2, 3), ((4)))$?
- $S \rightarrow L \rightarrow (F) \rightarrow (EN) \rightarrow (E, EN) \rightarrow (E, E, EN)$
 $\rightarrow (E, E, E\epsilon) \rightarrow (n, E, E) \rightarrow (n, L, E) \rightarrow (n, (F), E)$
 $\rightarrow (n, (EN), E) \rightarrow (n, (E, EN), E)$
 $\rightarrow (n, (E, E\epsilon), E) \rightarrow (n, (E, E), L)$
 $\rightarrow (n, (E, E), (F)) \rightarrow (n, (E, E), (EN))$
 $\rightarrow (n, (E, E), (E\epsilon)) \rightarrow (n, (E, E), (L))$
 $\rightarrow (n, (E, E), ((F))) \rightarrow (n, (E, E), ((EN)))$
 $\rightarrow (n, (E, E), ((E\epsilon))) \rightarrow (n, (n, E), ((E)))$
 $\rightarrow (n, (n, n), ((E))) \rightarrow (n, (n, n), ((n)))$

Forma e Forma di Frase

- Una **FORMA** generata da G partendo da un non-terminale $A \in V$ è una **STRINGA** $\alpha \in (V \cup \Sigma)^*$ tale che $A \xrightarrow{*} \alpha$
- Se A è l'assioma, α è detta una **FORMA DI FRASE**
- Una **FRASE** di $L(G)$ è una **Forma di Frase** che **NON Contiene Simboli Non-Terminali**

Grammatiche Erronee

Una Grammatica G è **PULITA** o **RIDOTTA** se valgono le seguenti condizioni.

- ① Ogni Non-Terminale A è **raggiungibile dall'assioma**, cioè se esiste una derivazione
 $S \xrightarrow{*} \alpha A \beta$
- ② Ogni Non-Terminale A è **DEFINITO**, cioè genera un linguaggio **NON VUOTO**, $L_A(G) \neq \emptyset$
 (si noti che se $L_A(G) = \{\epsilon\}$, allora A è definito)

Ricorsione

- Una derivazione a $n \geq 1$ passi $A \xrightarrow{n} xAy$ è detta **RICORSIVA**
(Immediatamente Ricorsiva se $n = 1$);
di conseguenza, il non-terminale A è detto **RICORSIVO**.
- Se x (rispettivamente y) è **VUOTA**, la ricorsione è detta **RICORSIONE SINISTRA** (risp. **RICORSIONE DESTRA**).

Ricorsione e INFINITEZZA DEL LINGUAGGIO

Proprietà

Condizione **Necessaria e Sufficiente** perché il linguaggio $L(G)$ sia **INFINITO**, dove G è una grammatica **Pulita** (in particolare, priva di derivazioni circolari), è che G permetti delle **Derivazioni Ricorsive**.

Esercizio 3.03

- $\Sigma = \{a, b, c\}$
- $V = \{S, B, C\}$
- $S \rightarrow a B C$
 $B \rightarrow a b$
 $B \rightarrow C a$
 $C \rightarrow c$
- $L(G)$ è finito o infinito?

Esercizio 30.3

- $\Sigma = \{a, b, c\}$
- $V = \{S, B, C\}$
- $S \rightarrow a B C$
 $B \rightarrow a b$
 $B \rightarrow C a$
 $C \rightarrow c$
- $L(G)$ è finito o infinito?
- È **FINITO**, perché non sono possibili derivazioni ricorsive

Esercizio 3.04: Linguaggio delle Liste Annidate

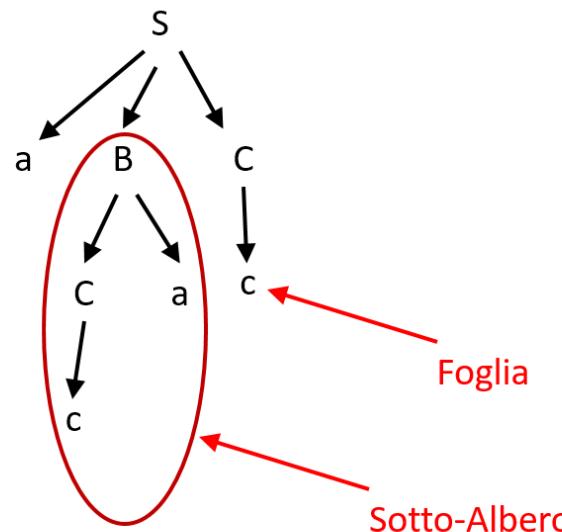
- $\Sigma = \{n, ", (,)\}$ $V = \{S, L, F, N, E\}$ Assioma: S
- $S \rightarrow L$
- $L \rightarrow (F)$
- $F \rightarrow \epsilon$
- $F \rightarrow E\ N$
- $N \rightarrow \epsilon$
- $N \rightarrow ", "\ E\ N$
- $E \rightarrow n$
- $E \rightarrow L$
- $L(G)$ è FINITO o INFINITO?

Albero

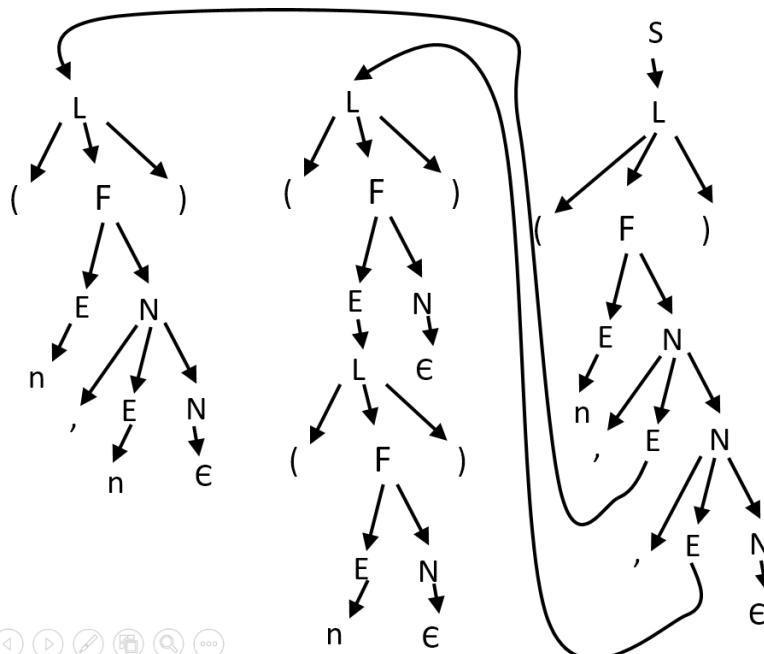
- Un **ALBERO** è un grafo **Orientato privo di Cicli**
tale che per ogni coppia di **Nodi** esiste un solo cammino che li congiunge.
- Un arco $N_1 \rightarrow N_2$ definisce la relazione **Padre-Figlio**.
- Il nodo **Privo di Padre** è detto **RADICE**.

Albero Sintattico

- Le regole di produzione possono essere viste come delle relazioni padre-figlio
- Una derivazione è ottenuta partendo dall'assioma, applicando le regole di produzione
- Ogni volta che un Non-Terminale viene espanso con la parte destra di una regola di produzione, viene implicitamente applicata la relazione padre-figlio
- Rappresentando la derivazione in questi termini, si ottiene un albero, detto **Albero Sintattico**

Albero Sintattico**Esercizio 3.05: Linguaggio delle Liste Annidate**

- Disegnare l'albero sintattico per $(1, (2, 3), ((4)))$?

**Derivazioni Sinistre e Destre**

- Una derivazione $\beta_0 \rightarrow \beta_1 \rightarrow \beta_2 \rightarrow \dots \rightarrow \beta_n$
- dove $\beta_i = \eta_i A_i \delta_i$
e $\beta_{i+1} = \eta_i \alpha_i \delta_i$
- è detta **CANONICA DESTRA**
(risp. **CANONICA SINISTRA**)
se per ogni $0 \leq i \leq n-1$ si ha che $\delta_i \in \Sigma^*$
(risp. $\eta_i \in \Sigma^*$).

Esercizio 3.06:

- $S \rightarrow a B C$
 $B \rightarrow a b$
 $B \rightarrow C a$
 $C \rightarrow c$
- Derivazione Destra per $S \xrightarrow{+} acac$
 $S \rightarrow \dots$
- Derivazione Sinistra per $S \xrightarrow{+} acac$
 $S \rightarrow \dots$

Esercizio 3.06:

- $S \rightarrow a B C$
 $B \rightarrow a b$
 $B \rightarrow C a$
 $C \rightarrow c$
- Derivazione Destra per $S \xrightarrow{+} acac$
 $S \rightarrow aBC \rightarrow aBc \rightarrow aCac \rightarrow acac$
- Derivazione Sinistra per $S \xrightarrow{+} acac$
 $S \rightarrow aBC \rightarrow aCaC \rightarrow acaC \rightarrow acac$

Grammatiche Lineari

- Una grammatica è detta **LINEARE** se la parte destra di ogni regola di produzione contiene al più **Un Solo Non-Terminale**.
- Se il Non-Terminale in questione è il simbolo più a **DESTRA** (risp. più a **SINISTRA**), la grammatica è detta **LINEARE DESTRA** (risp. **LINEARE SINISTRA**).

Esercizio 3.07: Linguaggio degli Identifieri

- $\Sigma = \{a, n, _\}$
- $V = \{S, L\}$
- Regole di Produzione
 $S \rightarrow a L$
 $L \rightarrow a L$
 $L \rightarrow n L$
 $L \rightarrow _ L$
 $L \rightarrow \epsilon$
- Assioma: S
- Questa è Lineare Destra?

Esercizio 30.7: Linguaggio degli Identifieri

- $\Sigma = \{a, n, _\}$
- $V = \{S, L\}$
- Regole di Produzione

$$S \rightarrow a \ L$$

$$L \rightarrow a \ L$$

$$L \rightarrow n \ L$$

$$L \rightarrow _ \ L$$

$$L \rightarrow \epsilon$$

- Assioma: S
- Questa è Lineare Destra? Sì

Grammatiche Lineari

I **Linguaggi Regolari** sono descritti da grammatiche **Lineari Destre** o **Lineari Sinistre**.

Esercizio 3.08: Linguaggio degli Identifieri, descritto da una grammatica lineare sinistra**Esercizio 30.8:** Linguaggio degli Identifieri

- $\Sigma = \{a, n, _\}$
 - $V = \{S, L\}$
 - Regole di Produzione
- $$S \rightarrow L$$
- $$L \rightarrow L \ a$$
- $$L \rightarrow L \ n$$
- $$L \rightarrow L \ _$$
- $$L \rightarrow a$$
- Assioma: S
 - Questa è Lineare Sinistra

Ambiguità

- Data una grammatica G , una frase x del linguaggio $L(G)$ è **Ambigua** se è generata da G con due alberi sintattici differenti.
- La grammatica G è detta **AMBIGUA** se almeno una delle frasi da essa generate è ambigua.

Esempio: Espressioni Matematiche

- $\Sigma = \{n, +, -, *, /, (,)\}$
- $V = \{S, E\}$
- Regole di Produzione

$$S \rightarrow E$$

$$E \rightarrow E + E$$

$$E \rightarrow E - E$$

$$E \rightarrow E * E$$

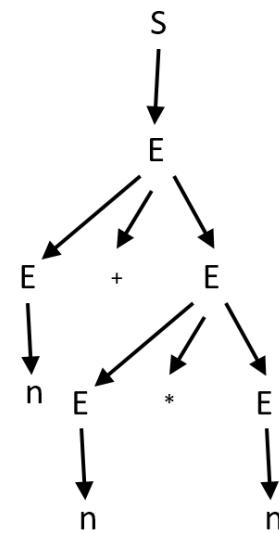
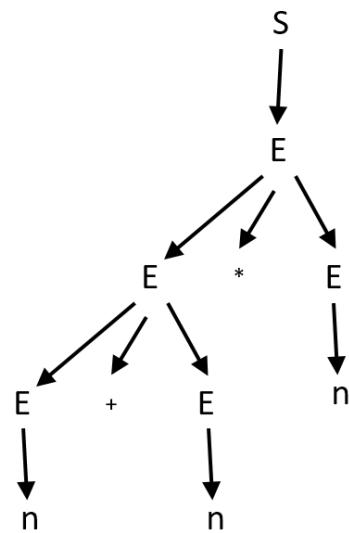
$$E \rightarrow E/E$$

$$E \rightarrow (E)$$

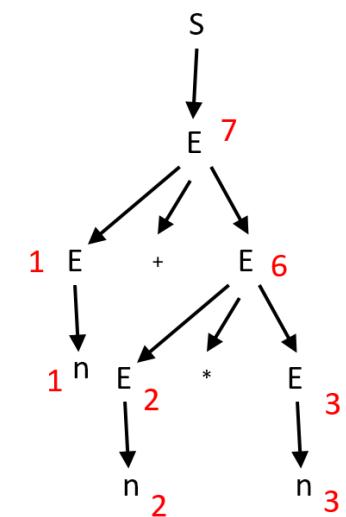
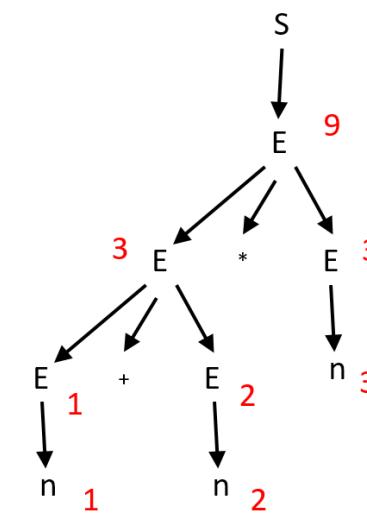
$$E \rightarrow n$$

Assioma: S

Albero Sintattico

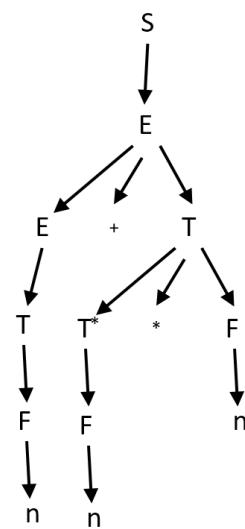
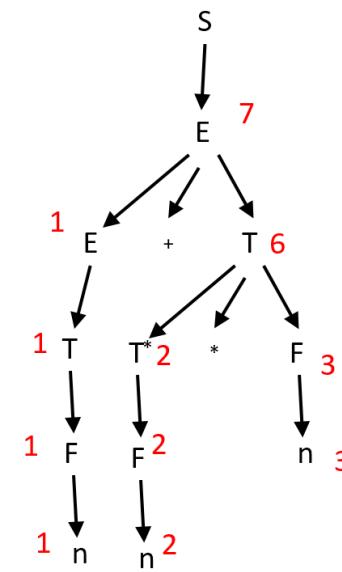


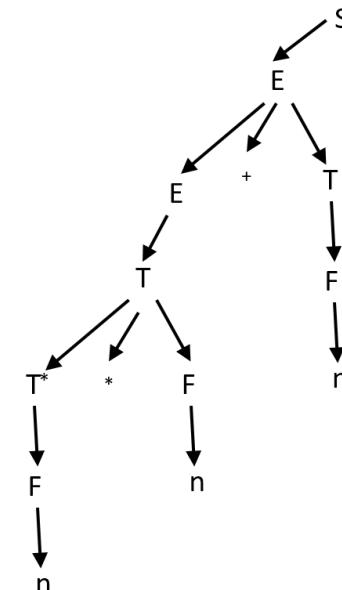
Albero Sintattico



Esercizio 3.09: Espressioni Matematiche

- $\Sigma = \{n, +, -, *, /, (,)\}$ $V = \{S, E, T, F\}$
- $S \rightarrow E$
 - $E \rightarrow E + T$
 - $E \rightarrow E - T$
 - $E \rightarrow T$
 - $T \rightarrow T * F$
 - $T \rightarrow T / F$
 - $T \rightarrow F$
 - $F \rightarrow (E)$
 - $F \rightarrow n$
- Assioma: S

Albero Sintattico**Albero Sintattico**

Albero Sintattico**Esercizio 3.10: Espressioni Matematiche**

- Che succede con $n * n + n$?

Linguaggi Formali e Compilatori
Argomento 04:
Parsing Discendente

Prof. Giuseppe Psaila

Laurea Magistrale in Ingegneria Informatica
Università di Berga

Parsing Discendente

- Ricostruisce le derivazioni **Canoniche Sinistre**
- Denominata **LL(1)**
Left-to-Right
Left-most
Look-ahead 1
- Look-ahead = prospezione
Quanti simboli terminali occorre guardare in avanti per decidere

Insieme degli Inizi di una stringa

- L'Insieme degli Inizi di una stringa $\alpha \in (V \cup \Sigma)^*$ è
 $Ini(\alpha) = \{a \in \Sigma \mid \alpha \xrightarrow{*} a\beta\}$
- $Ini(\epsilon) = \emptyset$

Annnullabilità di una stringa

- Una stringa $\alpha \in (V \cup \Sigma)^*$ è **Annnullabile** se da essa **Deriva la stringa** ϵ
- $Annulabile(\alpha) = \text{Vero se } \alpha \xrightarrow{*} \epsilon$
 $Annulabile(\alpha) = \text{Falso altrimenti}$

Insieme dei Seguiti di un simbolo Non-Terminale

- L'Insieme dei Seguiti di un Non-Terminale $A \in V$ è
 $Seg(A) = \{a \in \Sigma \cup \{\downarrow\} \mid S \xrightarrow{*} \alpha A a \beta\}$
- Implicitamente, la stringa da analizzare è terminata dal simbolo di **Fine Stringa** \downarrow
 Ma non può essere generata da S , quindi dobbiamo pensarla come $S \downarrow$
- Di conseguenza $\downarrow \in Seg(S)$ e $\downarrow \in Seg(A)$ se $S \xrightarrow{+} \alpha A$

Insieme Guida di una Regola di Produzione

- Data $A \rightarrow \alpha$
- $Gui(A \rightarrow \alpha) = Ini(\alpha)$ se $\neg Annulabile(\alpha)$
- $Gui(A \rightarrow \alpha) = Ini(\alpha) \cup Seg(A)$ se $Annulabile(\alpha)$

Come caso particolare, $Gui(A \rightarrow \epsilon) = Seg(A)$

Non-Terminali e Grammatica LL(1)

- Un **Non-Terminale A** è LL(1) se, **Per Ogni Coppia di Regole di Produzione con la stessa parte sinistra**

$$A \rightarrow \alpha_1$$

$$A \rightarrow \alpha_2$$

risulta $Gui(A \rightarrow \alpha_1) \cap Gui(A \rightarrow \alpha_2) = \emptyset$

- Una **Grammatica G** è LL(1) se **Tutti i Suoi Non-Terminali** sono LL(1)

Esempio: Numeri binari che iniziano con 1

- $\Sigma = \{0, 1\}$ $V = \{S, N\}$

Regole	Ini	Gui
$S \rightarrow 1 N$	{1}	{1}
$N \rightarrow 1 N$	{1}	{1}
$N \rightarrow 0 N$	{0}	{0}
$N \rightarrow \epsilon$	\emptyset	{ \swarrow }

- Non-Term.

Seg	
S	{ \swarrow }
N	{ \swarrow }

- La grammatica è LL(1)

Calcolo degli Insiemi degli Inizi di una stringa α

- $a \in Ini(\alpha)$ se $\alpha = a\beta$
- $a \in Ini(\alpha)$ se $\alpha = A\beta$ e $a \in Ini(\gamma)$, con $A \rightarrow \gamma$
- $a \in Ini(\alpha)$ se $\alpha = A\beta$ e $Annulabile(A)$ e $a \in Ini(\beta)$ (con $\beta \in (V \cup \Sigma)^*$)
- $Ini(\epsilon) = \emptyset$

Calcolo dell'Insieme dei Seguiti di un Non-Terminale A

- $\swarrow \in Seg(S)$
- $a \in Seg(A)$ se $B \rightarrow \alpha A \beta$ e $a \in Ini(\beta)$
- $a \in Seg(A)$ se $B \rightarrow \alpha A$, con $B \neq A$, e $a \in Seg(B)$
- $a \in Seg(A)$ se $B \rightarrow \alpha A \beta$, con $B \neq A$, $Annulabile(\beta)$ e $a \in Seg(B)$

Si intendono $\alpha \in (V \cup \Sigma)^*$ e $Seg(A) \subseteq \Sigma \cup \{\swarrow\}$

Esempio:

- $\Sigma = \{a, b, c\}$
- $V = \{S, A\}$
- Regole di Produzione
 $S \rightarrow A \ b$
 $S \rightarrow c \ S$
 $A \rightarrow a \ A$
 $A \rightarrow \epsilon$
- Assioma: S

$Annulabile(A) = Vero$	$Seg(A) = \{b\}$
$Annulabile(S) = Falso$	$Seg(S) = \{\checkmark\}$
• Equazioni Insiemistiche	
$Ini(Ab) = Ini(A) \cup Ini(b)$	$Ini(Ab) = Ini(A) \cup \{b\}$
$Ini(A) = Ini(aA) \cup \emptyset$	$Ini(A) = \{a\}$
$Ini(cS) = \{c\}$	$Ini(cS) = \{c\}$
$Ini(Ab) = \{a\} \cup \{b\} = \{a, b\}$	
$Ini(A) = \{a\}$	
$Ini(cS) = \{c\}$	

- $Gui(S \rightarrow Ab) = Ini(Ab) = \{a, b\}$
- $Gui(S \rightarrow cS) = Ini(cS) = \{c\}$
- $Gui(A \rightarrow aA) = Ini(aA) = \{a\}$
- $Gui(A \rightarrow \epsilon) = Seg(A) = \{b\}$
- Sia S che A sono LL(1)

Parser a Discesa Ricorsiva - Pseudo-codice

Program DiscesaRicorsiva

Var x : TESTO

cc : Terminale

Function Pross(): Terminale

Begin

*Cambia il valore di cc,
leggendo il nuovo terminale da x*

End Function

```
Procedure S()
```

```
Begin
```

```
  If cc ∈ {a, b} Then      //S → Ab
```

```
    call A()
```

```
    If cc ≠ 'b' Then Error
```

```
    cc := Pross()
```

```
    Return
```

```
End If
```

```
  If cc ∈ {c} Then      //S → cS
```

```
    If cc ≠ 'c' Then Error
```

```
    cc := Pross()
```

```
    call S()
```

```
    Return
```

```
End If
```

```
Error
```

```
End Procedure
```

```
Procedure A()
```

```
Begin
```

```
  If cc ∈ {a} Then      //A → aA
```

```
    If cc ≠ 'a' Then Error
```

```
    cc := Pross()
```

```
    call A()
```

```
    Return
```

```
End If
```

```
  If cc ∈ {b} Then      //A → ε
```

```
    Return
```

```
End If
```

```
Error
```

```
End Procedure
```

```
Begin          // Main Program
```

```
  cc := Pross()
```

```
  Call S()
```

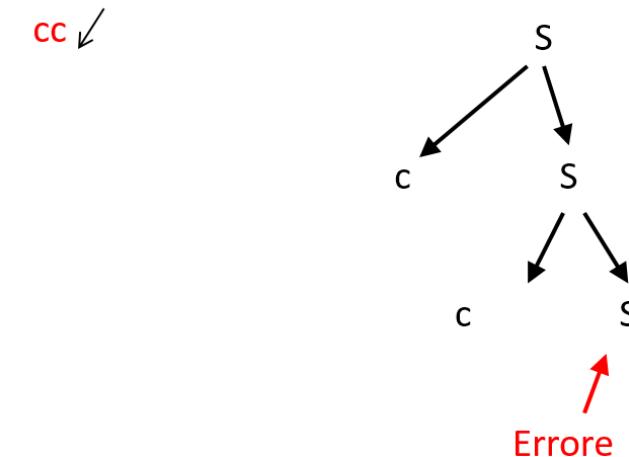
```
  If cc ≠ '↙' Then Error
```

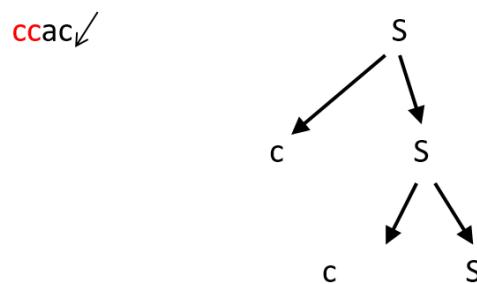
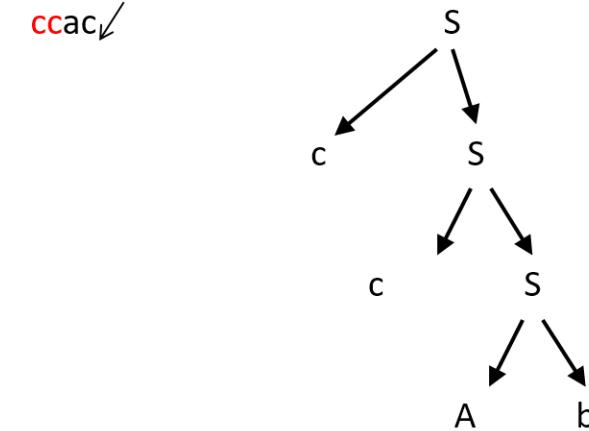
```
End Program    // Main Program
```

cc ↘

S

Esempio di Parsing: cc ↘

Esempio di Parsing: $cc \swarrow$ **Esempio di Parsing:** $cc \swarrow$ **Esempio di Parsing:** $ccac \swarrow$ **Esempio di Parsing:** $ccac \swarrow$ 

Esempio di Parsing: $ccac \swarrow$ **Esempio di Parsing:** $ccac \swarrow$ 

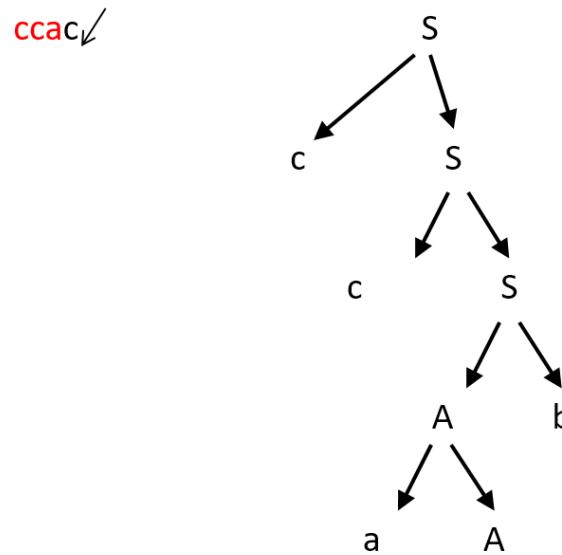
Prof. Giuseppe Psaila (UniBG) Linguaggi Formali e Compilatori Argomento

23 / 50

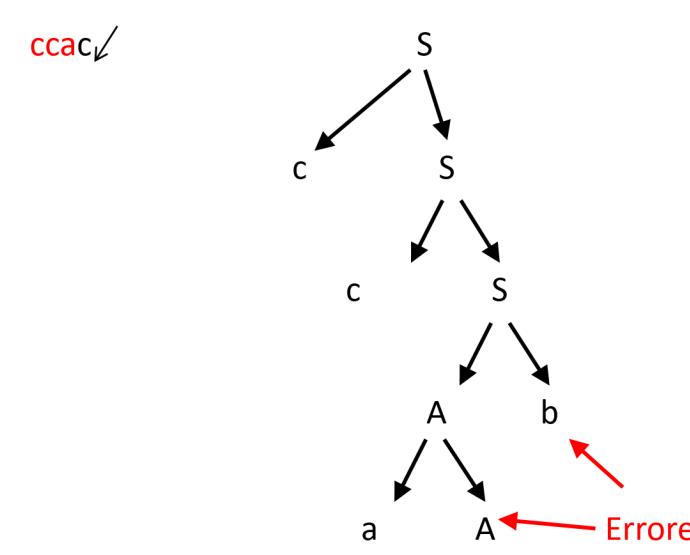
Prof. Giuseppe Psaila (UniBG)

Linguaggi Formali e Compilatori Argomento

24 / 50

Esempio di Parsing: $ccac \swarrow$ 

Prof. Giuseppe Psaila (UniBG) Linguaggi Formali e Compilatori Argomento



Esercizio 4.01: Palindromi con CentroEsempio: *acbaqabca*

- $\Sigma = \{a, b, c, q\}$ $V = \{S, T\}$

- Regole

<i>Ini</i>	<i>Gui</i>
------------	------------

$S \rightarrow T$	
$T \rightarrow a T a$	
$T \rightarrow b T b$	
$T \rightarrow c T c$	
$T \rightarrow q$	

- Non-Term.

<i>Seg</i>

S	
T	

Esercizio 4.01: Palindromi con Centro
È LL(1)

- $\Sigma = \{a, b, c, q\}$ $V = \{S, T\}$

- Regole

<i>Ini</i>	<i>Gui</i>
------------	------------

$S \rightarrow T$	$\{a, b, c, q\}$	$\{a, b, c, q\}$
$T \rightarrow a T a$	$\{a\}$	$\{a\}$
$T \rightarrow b T b$	$\{b\}$	$\{b\}$
$T \rightarrow c T c$	$\{c\}$	$\{c\}$
$T \rightarrow q$	$\{q\}$	$\{q\}$

- Non-Term.

<i>Seg</i>

S	$\{\swarrow\}$
T	$\{a, b, c, \swarrow\}$

Parsing Discendente

Esercizio 4.02: Palindromi senza CentroEsempio: *acbaabca*

- $\Sigma = \{a, b, c\}$ $V = \{S, T\}$

- Regole

<i>Ini</i>	<i>Gui</i>
------------	------------

$S \rightarrow T$	
$T \rightarrow a T a$	
$T \rightarrow b T b$	
$T \rightarrow c T c$	
$T \rightarrow \epsilon$	

- Non-Term.

<i>Seg</i>

S	
T	

Parsing Discendente

Esercizio 4.02: Palindromi senza Centro

NON È LL(1)

- $\Sigma = \{a, b, c\}$ $V = \{S, T\}$

- Regole

<i>Ini</i>	<i>Gui</i>
------------	------------

$S \rightarrow T$	$\{a, b, c\}$	$\{a, b, c, \swarrow\}$
$T \rightarrow a T a$	$\{a\}$	$\{a\}$
$T \rightarrow b T b$	$\{b\}$	$\{b\}$
$T \rightarrow c T c$	$\{c\}$	$\{c\}$
$T \rightarrow \epsilon$	\emptyset	$\{a, b, c, \swarrow\}$

- Non-Term.

<i>Seg</i>

S	$\{\swarrow\}$
T	$\{a, b, c, \swarrow\}$

Tecnica di Calcolo

- Grammatica
- $\Sigma = \{a, b, c\}$ $V = \{S, A, B, C\}$
- Regole

$$\overline{S \rightarrow A \ B}$$

$$A \rightarrow a \ A$$

$$A \rightarrow \epsilon$$

$$B \rightarrow B \ b$$

$$B \rightarrow C$$

$$C \rightarrow c$$

Equazioni Insiemistiche per gli Insiemi degli Inizi

- $Ini(S \rightarrow AB) = Ini(A) \cup Ini(B)$
- $Ini(A \rightarrow aA) = \{a\}$
- $Ini(A) = Ini(A \rightarrow aA) \cup \emptyset = Ini(aA)$
- $Ini(B \rightarrow Bb) = Ini(B)$
- $Ini(B \rightarrow C) = Ini(C)$
- $Ini(B) = Ini(B \rightarrow Bb) \cup Ini(B \rightarrow C) =$
 $= Ini(Bb) \cup Ini(C) = Ini(B) \cup Ini(C)$
- $Ini(C \rightarrow c) = \{c\}$
- $Ini(C) = Ini(C \rightarrow c) = \{c\}$

Come risolvere il sistema di equazioni insiemistiche?

Tecnica iterativa a Punto Fisso

Tecnica a Punto Fisso

- **Passo 0:** per ogni produzione $A_i \rightarrow \alpha_i$, si definisce $Ini^0(A_i \rightarrow \alpha_i) = Ini^0(\alpha_i)$ insieme degli **Inizi Immediati** (ottenibili direttamente da α_i); inoltre, $Ini^0(A) = \cup Ini^0(A_i \rightarrow \alpha_i)$ (per ogni $A \in \Sigma$)
- Al passo $j \geq 1$, si calcolano le parti sinistre delle equazioni insiemistiche $Ini^j(\alpha_i)$ usando, a destra, le versioni degli insiemi degli inizi $Ini^{j-1}(A)$ calcolati al passo $j - 1$.
- Se almeno per una produzione $A_i \rightarrow \alpha_i$ risulta $Ini^j(A_i \rightarrow \alpha_i) \neq Ini^{j-1}(A_i \rightarrow \alpha_i)$, si procede, altrimenti ci si ferma (punto fisso raggiunto).

Equazioni Insiemistiche per gli Insiemi degli Inizi

- $Ini^0(S \rightarrow AB) = \emptyset$
- $Ini^0(A \rightarrow aA) = \{a\}$
- $Ini^0(A) = Ini^0(A \rightarrow aA) \cup Ini(A \rightarrow \epsilon) = \{a\} \cup \emptyset = \{a\}$
- $Ini^0(B \rightarrow Bb) = \emptyset$, $Ini^0(B \rightarrow C) = \emptyset$
- $Ini^0(B) = Ini^0(B \rightarrow Bb) \cup Ini^0(B \rightarrow C) = \emptyset \cup \emptyset = \emptyset$
- $Ini^0(C \rightarrow c) = \{c\}$, $Ini^0(C) = Ini^0(C \rightarrow c) = \{c\}$
- $Ini^j(S \rightarrow AB) = Ini^{j-1}(A) \cup Ini^{j-1}(B)$
- $Ini^j(A \rightarrow aA) = \{a\}$
- $Ini^j(A) = Ini^j(A \rightarrow aA) \cup \emptyset$
- $Ini^j(B \rightarrow Bb) = Ini^{j-1}(B)$, $Ini^j(B \rightarrow C) = Ini^{j-1}(C)$
- $Ini^j(B) = Ini^j(B \rightarrow Bb) \cup Ini^j(B \rightarrow C)$
- $Ini^j(C \rightarrow c) = \{c\}$, $Ini^j(C) = Ini^j(C \rightarrow c) = \{c\}$

Calcolo Insieme degli Inizi**Calcolo Insieme degli Inizi**

Regole	0	1	2	3
$S \rightarrow A B$	\emptyset	$\{a\}$	$\{a, c\}$	$\{a, c\}$
$A \rightarrow a A$	$\{a\}$	$\{a\}$	$\{a\}$	$\{a\}$
$A \rightarrow \epsilon$	\emptyset	\emptyset	\emptyset	\emptyset
$B \rightarrow B b$	\emptyset	\emptyset	$\{c\}$	$\{c\}$
$B \rightarrow C$	\emptyset	$\{c\}$	$\{c\}$	$\{c\}$
$C \rightarrow c$	$\{c\}$	$\{c\}$	$\{c\}$	$\{c\}$

Regole	0
$S \rightarrow A B$	\emptyset
$A \rightarrow a A$	$\{a\}$
$A \rightarrow \epsilon$	\emptyset
$B \rightarrow B b$	\emptyset
$B \rightarrow C$	\emptyset
$C \rightarrow c$	$\{c\}$

Calcolo Insieme degli Inizi

Regole	0
$S \rightarrow A B$	\emptyset
$A \rightarrow a A$	$\{a\}$
$A \rightarrow \epsilon$	\emptyset
$B \rightarrow B b$	\emptyset
$B \rightarrow C$	\emptyset
$C \rightarrow c$	$\{c\}$

Calcolo Insieme degli Inizi

Regole	0	1
$S \rightarrow A B$	\emptyset	$\{a\}$
$A \rightarrow a A$	$\{a\}$	$\{a\}$
$A \rightarrow \epsilon$	\emptyset	\emptyset
$B \rightarrow B b$	\emptyset	\emptyset
$B \rightarrow C$	\emptyset	$\{c\}$
$C \rightarrow c$	$\{c\}$	$\{c\}$

Calcolo Insieme degli Inizi

Regole	0	1	2
$S \rightarrow A B$	\emptyset	$\{a\}$	$\{a, c\}$
$A \rightarrow a A$	$\{a\}$	$\{a\}$	$\{a\}$
$A \rightarrow \epsilon$	\emptyset	\emptyset	\emptyset
$B \rightarrow B b$	\emptyset	\emptyset	$\{c\}$
$B \rightarrow C$	\emptyset	$\{c\}$	$\{c\}$
$C \rightarrow c$	$\{c\}$	$\{c\}$	$\{c\}$

Calcolo Insieme degli Inizi

Regole	0	1	2	3
$S \rightarrow A B$	\emptyset	$\{a\}$	$\{a, c\}$	$\{a, c\}$
$A \rightarrow a A$	$\{a\}$	$\{a\}$	$\{a\}$	$\{a\}$
$A \rightarrow \epsilon$	\emptyset	\emptyset	\emptyset	\emptyset
$B \rightarrow B b$	\emptyset	\emptyset	$\{c\}$	$\{c\}$
$B \rightarrow C$	\emptyset	$\{c\}$	$\{c\}$	$\{c\}$
$C \rightarrow c$	$\{c\}$	$\{c\}$	$\{c\}$	$\{c\}$

Parsing Discendente

Parsing Discendente

Equazioni Insiemistiche per gli Insiemi dei Seguiti

- $\text{Seg}(S) = \{\swarrow\}$
- $\text{Seg}(A) = \text{Ini}(B) = \{c\}$
- $\text{Seg}(B) = \{b\} \cup \text{Seg}(S)$
- $\text{Seg}(C) = \text{Seg}(B)$

Risolviamo anche questo sistema con la Tecnica iterativa
a **Punto Fisso**

Equazioni Insiemistiche per gli Insiemi dei Seguiti

- $\text{Seg}^0(S) = \{\swarrow\}$
- $\text{Seg}^0(A) = \{c\}$
- $\text{Seg}^0(B) = \{b\}$
- $\text{Seg}^0(C) = \emptyset$
- $\text{Seg}^j(S) = \{\swarrow\}$
- $\text{Seg}^j(A) = \{c\}$
- $\text{Seg}^j(B) = \{b\} \cup \text{Seg}^{j-1}(S)$
- $\text{Seg}^j(C) = \text{Seg}^{j-1}(B)$

Grammatica

Regole

$$S \rightarrow A \ B$$

$$A \rightarrow a \ A$$

$$A \rightarrow \epsilon$$

$$B \rightarrow B \ b$$

$$B \rightarrow C$$

$$C \rightarrow c$$

Calcolo Insieme dei Seguiti

Non-Term.	0	1	2	3
S	$\{\swarrow\}$	$\{\swarrow\}$	$\{\swarrow\}$	$\{\swarrow\}$
A	$\{c\}$	$\{c\}$	$\{c\}$	$\{c\}$
B	$\{b\}$	$\{b, \swarrow\}$	$\{b, \swarrow\}$	$\{b, \swarrow\}$
C	\emptyset	$\{b\}$	$\{b, \swarrow\}$	$\{b, \swarrow\}$

Calcolo Insieme dei Seguiti

Non-Term.	0
S	$\{\swarrow\}$
A	$\{c\}$
B	$\{b\}$
C	\emptyset

Calcolo Insieme dei Seguiti

Non-Term.	0
S	$\{\swarrow\}$
A	$\{c\}$
B	$\{b\}$
C	\emptyset

Calcolo Insieme dei Seguiti

Non-Term.	0	1
S	$\{\swarrow\}$	$\{\swarrow\}$
A	$\{c\}$	$\{c\}$
B	$\{b\}$	$\{b, \swarrow\}$
C	\emptyset	$\{b\}$

Calcolo Insieme dei Seguiti

Non-Term.	0	1	2
S	$\{\swarrow\}$	$\{\swarrow\}$	$\{\swarrow\}$
A	$\{c\}$	$\{c\}$	$\{c\}$
B	$\{b\}$	$\{b, \swarrow\}$	$\{b, \swarrow\}$
C	\emptyset	$\{b\}$	$\{b, \swarrow\}$

Calcolo Insieme dei Seguiti

Non-Term.	0	1	2	3
S	$\{\swarrow\}$	$\{\swarrow\}$	$\{\swarrow\}$	$\{\swarrow\}$
A	$\{c\}$	$\{c\}$	$\{c\}$	$\{c\}$
B	$\{b\}$	$\{b, \swarrow\}$	$\{b, \swarrow\}$	$\{b, \swarrow\}$
C	\emptyset	$\{b\}$	$\{b, \swarrow\}$	$\{b, \swarrow\}$

Insiemi Guida

Regole	Ini	Gui	
$S \rightarrow A B$	$\{a, c\}$	$\{a, c\}$	
$A \rightarrow a A$	$\{a\}$	$\{a\}$	
$A \rightarrow \epsilon$	\emptyset	$\{c\}$	
$B \rightarrow B b$	$\{c\}$	$\{c\}$	Conflitto
$B \rightarrow C$	$\{c\}$	$\{c\}$	Conflitto
$C \rightarrow c$	$\{c\}$	$\{c\}$	

Linguaggi Formali e Compilatori

Argomento 05:

Parsing Ascendente

Prof. Giuseppe Psaila

Laurea Magistrale in Ingegneria Informatica
Università di Berga

- Ricostruisce le derivazioni **Canoniche Destre**
- Denominata **LR(0)**
Left-to-Right
Right-most
Look-ahead 0
- Cioè senza look-ahead, si consuma il simbolo e si fa qualche cosa
(ma senza look-ahead risulta limitata, quindi poi lo introdurremo)

Parsing Ascendente

Parsing Ascendente

Parti Destre e Automi

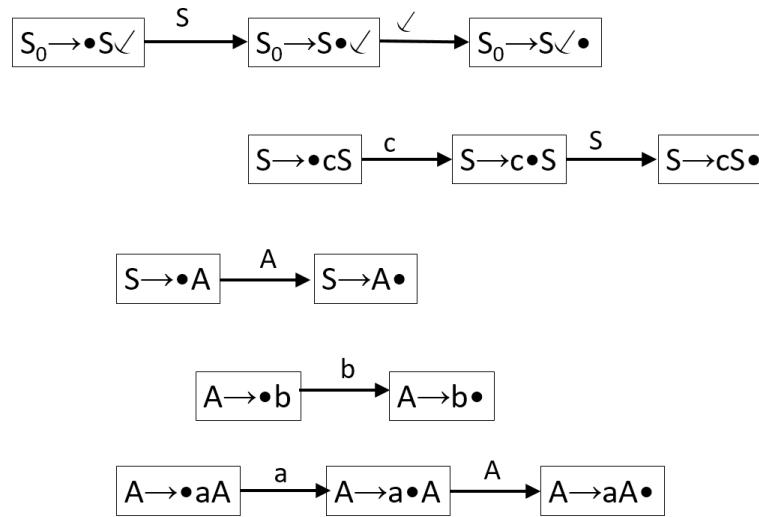
- La parte destra di una regola di produzione $A \rightarrow \alpha$ è una stringa basata sull'alfabeto ($V \cup \Sigma$)
- Un semplice automa a stati finiti con alfabeto ($V \cup \Sigma$) è in grado di riconoscerla
- Possiamo immaginare di avere tanti micro-automi, uno per ogni regola

Parti Destre e Automi

Consideriamo la grammatica seguente, con $\Sigma = \{a, b, c\}$ e $V = \{S, A\}$ (S_0 è il **Super-Assioma**)

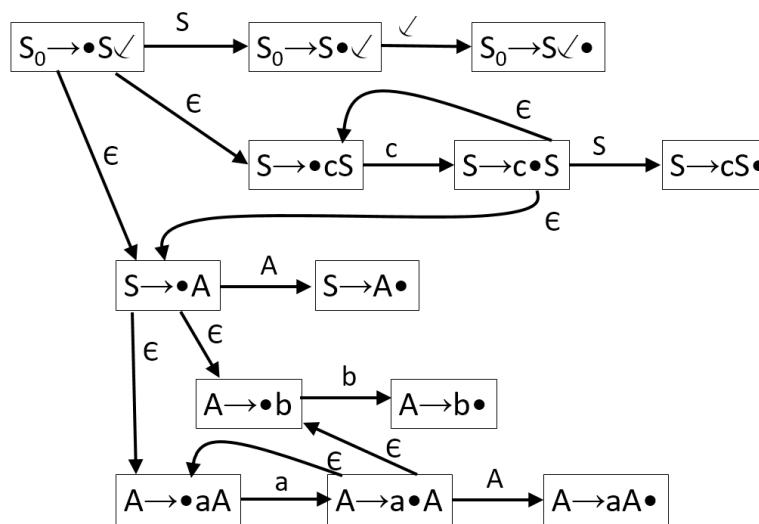
- $S_0 \rightarrow S \swarrow$
 $S \rightarrow A$
 $S \rightarrow c S$
 $A \rightarrow a A$
 $A \rightarrow b$

Parti Destre e Automi



- Il simbolo \bullet indica la **testina di lettura**, quindi quando un simbolo viene consumato, si sposta a **destra**
- Ma che cosa vuol dire **Consumare un Non-Terminale?**
- Vuol dire **Aver Riconosciuto il Sotto-Albero in Esso Radicato**
- IDEA: mettiamo delle ϵ -mosse di raccordo, ma così facendo l'automa diventa non-deterministico
- Inoltre, occorre associare una pila per gestire gli annidamenti e i punti di decisione

Parti Destre e Automi



- Quando una regola di produzione viene riconosciuta
- sul dispositivo di ingresso viene messo il simbolo Non-Terminale padre della produzione
- quindi si deve tornare indietro allo stato dal quale è stata fatta la mossa ϵ .
- Come fare a eliminare il non-determinismo?
Raccogliendo in un unico stato tutte le regole che espandono un **non-terminale marcato** da \bullet

Costruzione Automa LR(0)

- Data la grammatica $G(V, \Sigma, P, S_0)$
(con $S_0 \rightarrow S \in P$ e S_0 **Non Ricorsivo**)
- Uno stato s è un **Insieme di Candidate**

Costruzione Automa LR(0): Candidate

- Una candidata c ha la forma
 $N \rightarrow \alpha \bullet \beta$
 con $\alpha, \beta \in (V \cup \Sigma)^*$
 e $N \rightarrow \alpha\beta \in P$

Costruzione Automa LR(0): Candidate di Spostamento/Riduzione

- Candidata di **SPOSTAMENTO**:
 Una candidata $N \rightarrow \alpha \bullet \beta$ è detta di **Spostamento** se $|\beta| > 0$
- Candidata di **RIDUZIONE**:
 Una candidata $N \rightarrow \alpha \bullet \beta$ è detta di **Riduzione** se $|\beta| = 0$

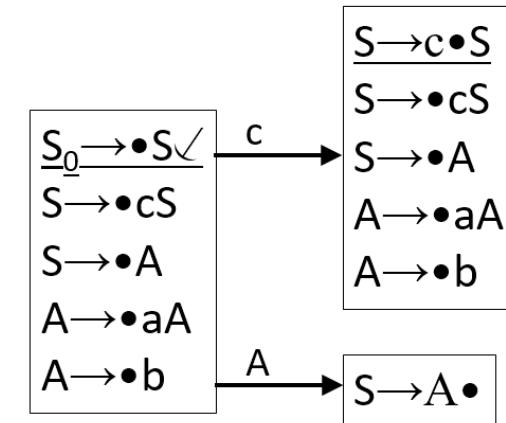
Costruzione Automa LR(0): Candidate Core/di Completamento

- Candidata **CORE**:
 Una candidata $N \rightarrow \alpha \bullet \beta$ è detta **CORE** se $|\alpha| \neq 0$ o se è $S_0 \rightarrow \bullet S$
- Candidata di **COMPLETAMENTO**:
 Una candidata $N \rightarrow \alpha \bullet \beta$ è detta di **COMPLETAMENTO** se $|\alpha| = 0$ e se vi è una candidata $\bar{c} = M \rightarrow \bar{\alpha} \bullet N\bar{\beta}$ nello stesso stato s

Costruzione Automa LR(0): Mosse Uscenti

- Dati due stati s_1 e s_2 , nell'automa esiste una transizione $s_1 \xrightarrow{A} s_2$ (con $A \in (V \cup \Sigma)$) se
- in s_1 vi è un insieme di candidate $m(A) = \{c_A_1, c_A_2, \dots, c_A_n\} \neq \emptyset$, con $c_A_i : N_i \rightarrow \alpha_i \bullet A \beta_i$
- e in s_2 vi è un insieme $\overline{m}(A) = \{\overline{c_A}_1, \overline{c_A}_2, \dots, \overline{c_A}_n\}$ tale che per ogni candidata $c_A_i \in m(A)$, esiste la corrispondente candidata $\overline{c_A}_i$ tale che $\overline{c_A}_i : N_i \rightarrow \alpha_i A \bullet \beta_i$

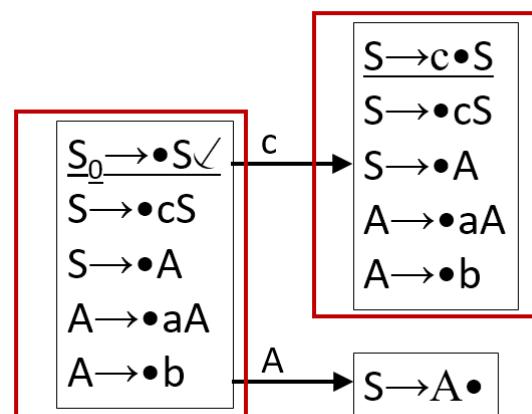
Alcuni Stati



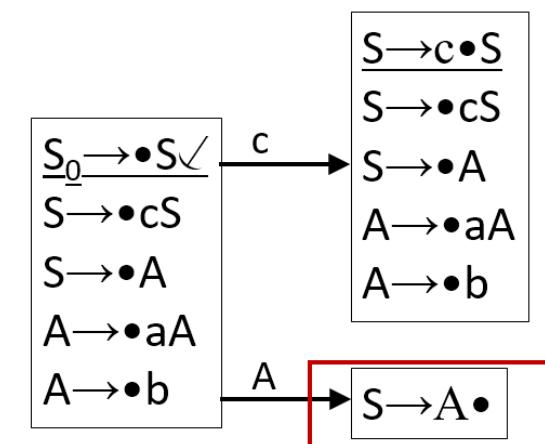
Parsing Ascendente

Parsing Ascendente

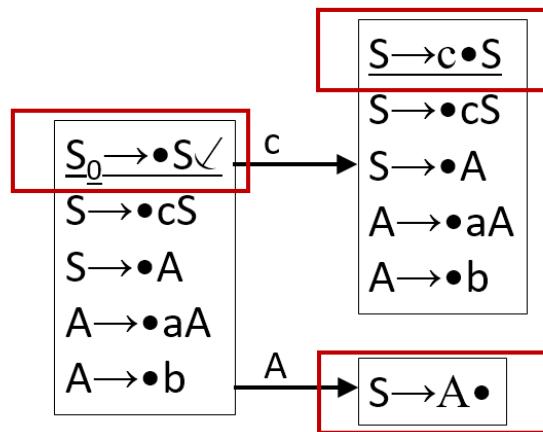
Candidate di Spostamento



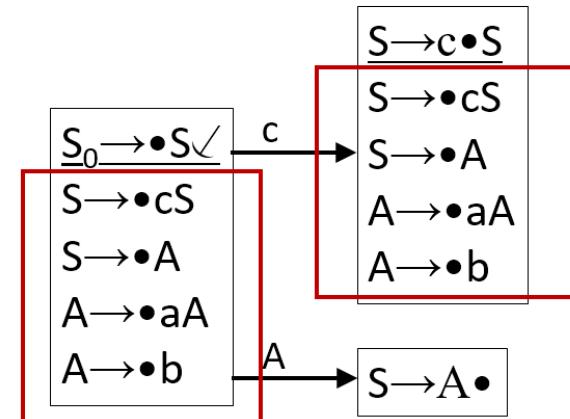
Candidate di Riduzione



Candidate Core



Candidate di Completamento



Parsing Ascendente

Parsing Ascendente

Procedimento Costruttivo

• Passo 1

Lo stato iniziale I_1 contiene la candidata core

$$S_0 \rightarrow •S \swarrow$$

• Passo 2

Per ogni stato s , per ogni candidata $M \rightarrow \alpha \bullet N\beta$ con $N \in V$

si aggiungono in s le candidate di **completamento**

$N \rightarrow \bullet\alpha_i$, per ogni regola di produzione $N \rightarrow \alpha_i \in P$.

Si continua ad aggiungere candidate di completamento fino a che tutti i simboli non-terminali marcati con \bullet sono stati completati

• Passo 3

Per ogni stato s , si determinano le mosse uscenti.

Per ogni gruppo di candidate $m(A) = \{c_A\}$, con $c_A : N_i \rightarrow \alpha_i \bullet A\beta_i$ (e con $A \in (V \cup \Sigma)$), cioè le candidate con lo stesso simbolo marcato da \bullet ,

si crea (se non è già presente) uno stato \bar{s} le cui candidata core sono esattamente $\bar{m}(A) = \{\overline{c_A}\}$ (con $|m(A)| = |\bar{m}(A)|$),

tale che $\overline{c_A} : N_i \rightarrow \alpha_i \bullet A\beta_i$ deriva da $c_A : N_i \rightarrow \alpha_i \bullet A\beta_i$.

Si aggiunge la transizione $s \xrightarrow{A} \bar{s}$.

• Per ogni nuovo stato s , si ripete dal **Passo 2**.

Conflitti

- Si dice che uno stato s presenta un **Conflitto SPOSTAMENTO/RIDUZIONE** se contiene sia una candidata di spostamento che una candidata di riduzione.
- Si dice che uno stato s presenta un **Conflitto RIDUZIONE/RIDUZIONE** se contiene due diverse candidate di riduzione

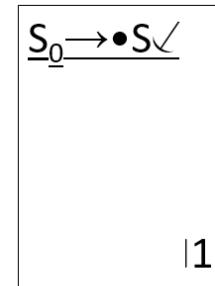
Automa LR(0)

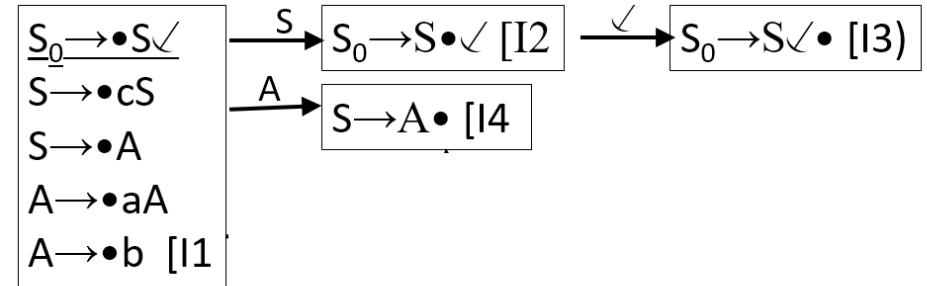
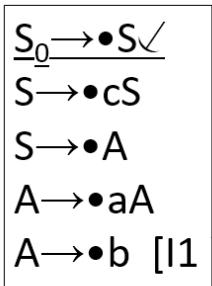
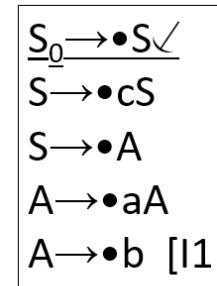
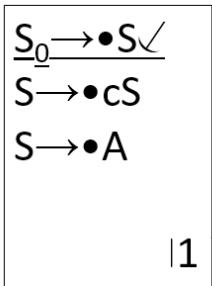
- Uno stato s è detto **LR(0)** se non contiene né conflitti SPOSTAMENTO/RIDUZIONE né conflitti RIDUZIONE/RIDUZIONE.
- Un Automa è detto **LR(0)** se tutti i suoi stati sono LR(0).
Conseguentemente, anche la grammatica è detta LR(0).

Automa LR(0) per la grammatica

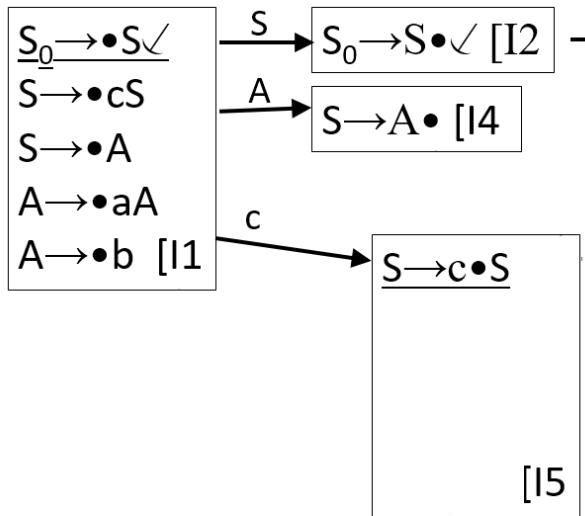
$\Sigma = \{a, b, c\}$ e $V = \{S, A\}$ (S_0 è il **Super-Assioma**)

- $S_0 \rightarrow S \downarrow$
- $S \rightarrow A$
- $S \rightarrow c$
- $S \rightarrow S$
- $A \rightarrow a$
- $A \rightarrow A$
- $A \rightarrow b$

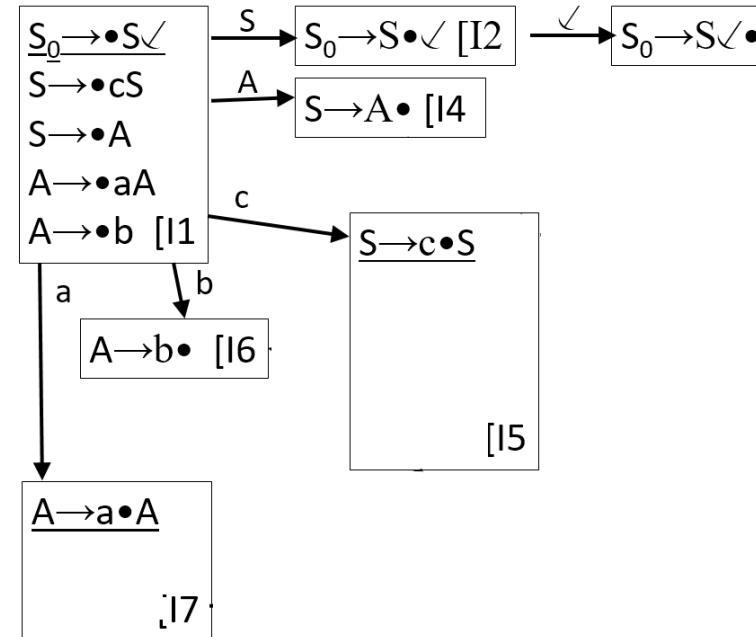




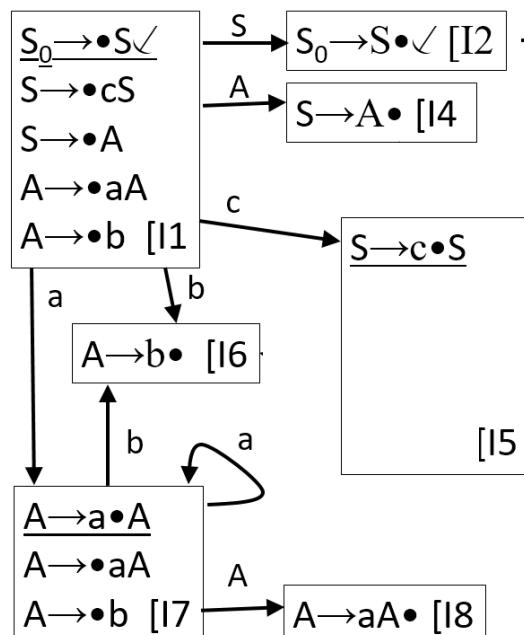
Parsing Ascendente



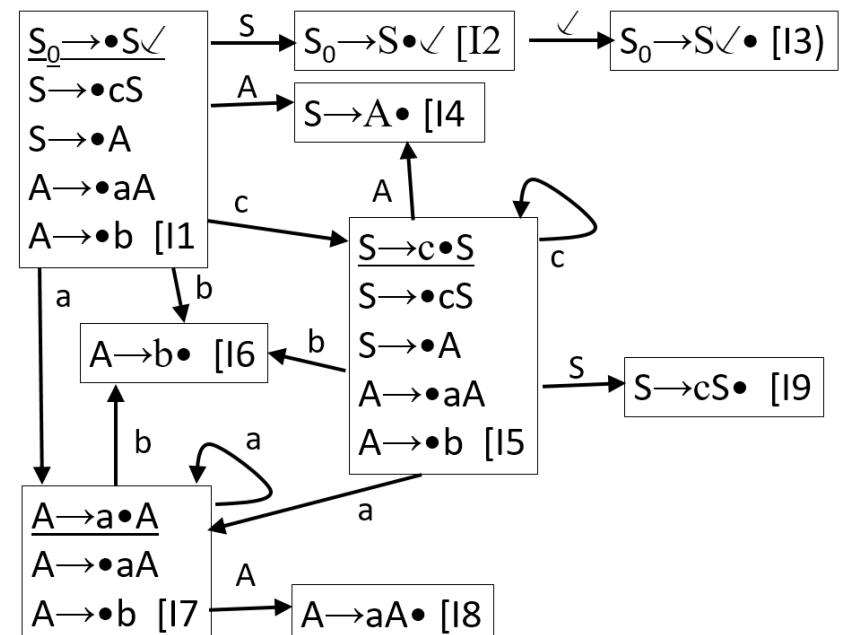
Parsing Ascendente



Parsing Ascendente



Parsing Ascendente



Funzionamento dell'Automa**• Passo 1**

Inizializzare la pila con lo stato iniziale I_1

• Passo 2

Con uno stato s in cima alla pila,
consumare il simbolo c dal dispositivo di ingresso e
metterlo in cima alla pila.

• Passo 3

Con una coppia (s, c) in cima alla pila, dove s è uno
stato e $c \in (V \cup \Sigma)$,
se la transizione $s \xrightarrow{c} s'$ NON è definita, segnalare
errore,
altrimenti mettere s' in cima alla pila

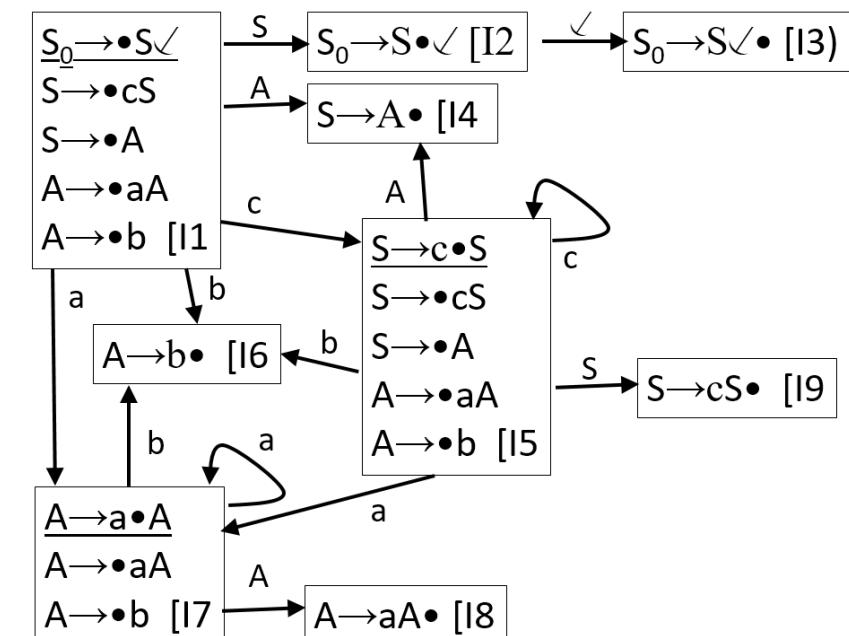
Funzionamento dell'Automa**• Passo 4**

Se lo stato s in cima alla pila può ridurre la regola
 $N \rightarrow \alpha$ (dalla candidata $N \rightarrow \alpha \bullet$)
rimuovere dalla pila $n = |\alpha|$ coppie (c, s)
e impilare N (parte sinistra della regola ridotta).

- Se la pila contiene solo $[I_1, S_0]$, la stringa è stata riconosciuta
altrimenti ripetere dal **Passo 2**

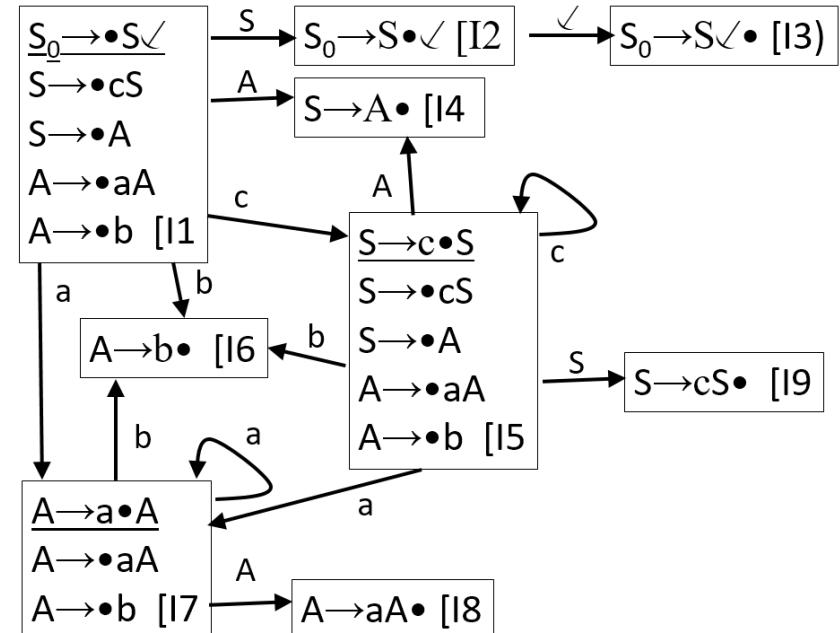
Esempio: stringa $cab \swarrow$

Stringa	Pila	Azione
$cab \swarrow$	I_1	



Esempio: stringa $cab \swarrow$

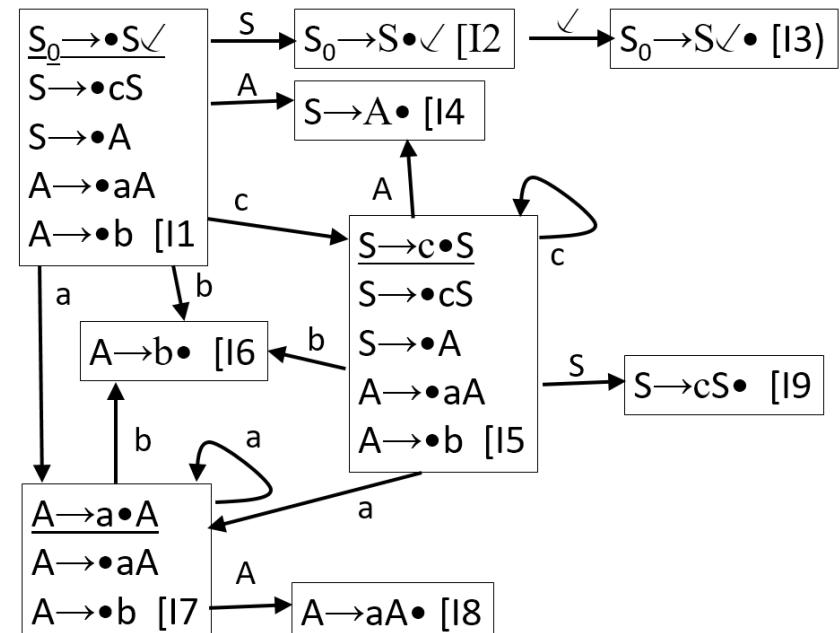
Stringa	Pila	Azione
$cab \swarrow$	I_1	
$ab \swarrow$	$I_1 c I_5$	



Prof. Giuseppe Psaila (UniBG) Linguaggi Formali e Compilatori Argomento 37 / 89 Prof. Giuseppe Psaila (UniBG) Linguaggi Formali e Compilatori Argomento 38 / 89

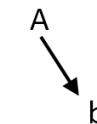
Esempio: stringa $cab \swarrow$

Stringa	Pila	Azione
$cab \swarrow$	I_1	
$ab \swarrow$	$I_1 c I_5$	
$b \swarrow$	$I_1 c I_5 a I_7$	



Esempio: stringa $cab \swarrow$

Stringa	Pila	Azione
$cab \swarrow$	l_1	
$ab \swarrow$	l_1cl_5	
$b \swarrow$	$l_1cl_5al_7$	
\swarrow	$l_1cl_5al_7bl_6$	Riduzione $A \rightarrow b$

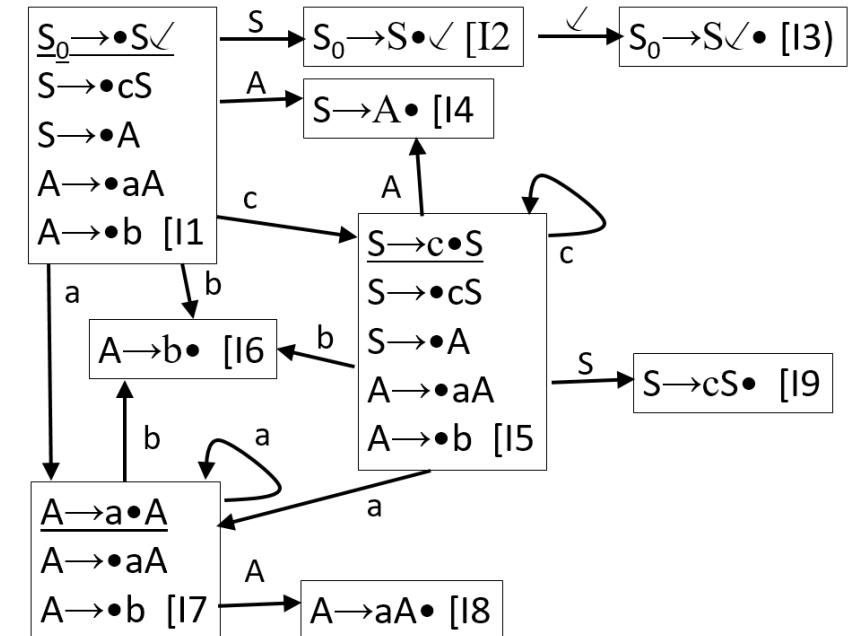


Parsing Ascendente

Parsing Ascendente

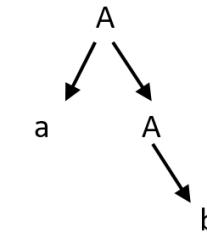
Esempio: stringa $cab \swarrow$

Stringa	Pila	Azione
$cab \swarrow$	l_1	
$ab \swarrow$	l_1cl_5	
$b \swarrow$	$l_1cl_5al_7$	
\swarrow	$l_1cl_5al_7bl_6$	Riduzione $A \rightarrow b$
\swarrow	$l_1cl_5al_7A$	



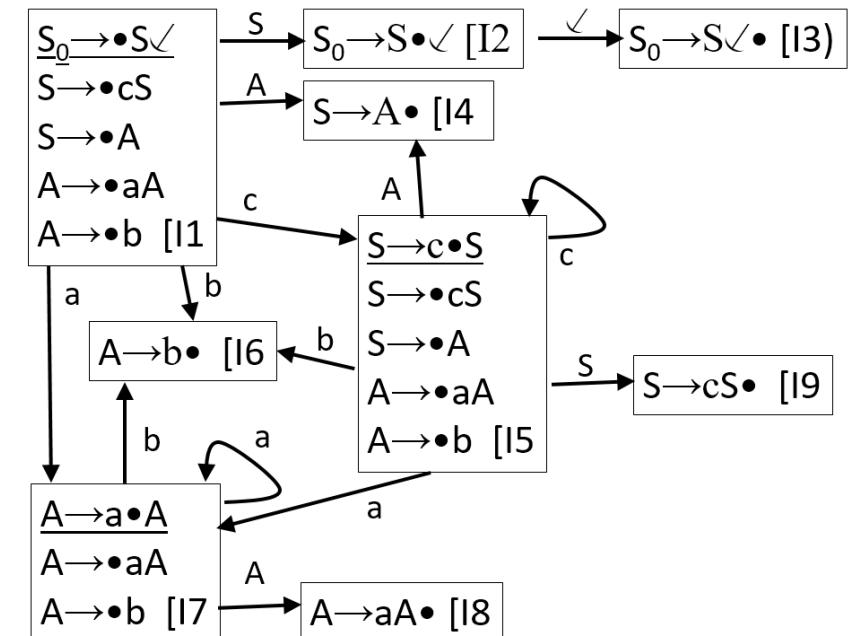
Esempio: stringa $cab \swarrow$

Stringa	Pila	Azione
$cab \swarrow$	I_1	
$ab \swarrow$	I_1cl_5	
$b \swarrow$	$I_1cl_5al_7$	
\swarrow	$I_1cl_5al_7bl_6$	Riduzione $A \rightarrow b$
\swarrow	$I_1cl_5al_7Al_8$	Riduzione $A \rightarrow aA$



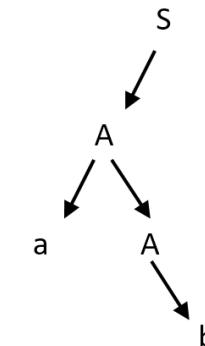
Esempio: stringa $cab \swarrow$

Stringa	Pila	Azione
$cab \swarrow$	I_1	
$ab \swarrow$	I_1cl_5	
$b \swarrow$	$I_1cl_5al_7$	
\swarrow	$I_1cl_5al_7bl_6$	Riduzione $A \rightarrow b$
\swarrow	$I_1cl_5al_7Al_8$	Riduzione $A \rightarrow aA$
\swarrow	I_1cl_5A	



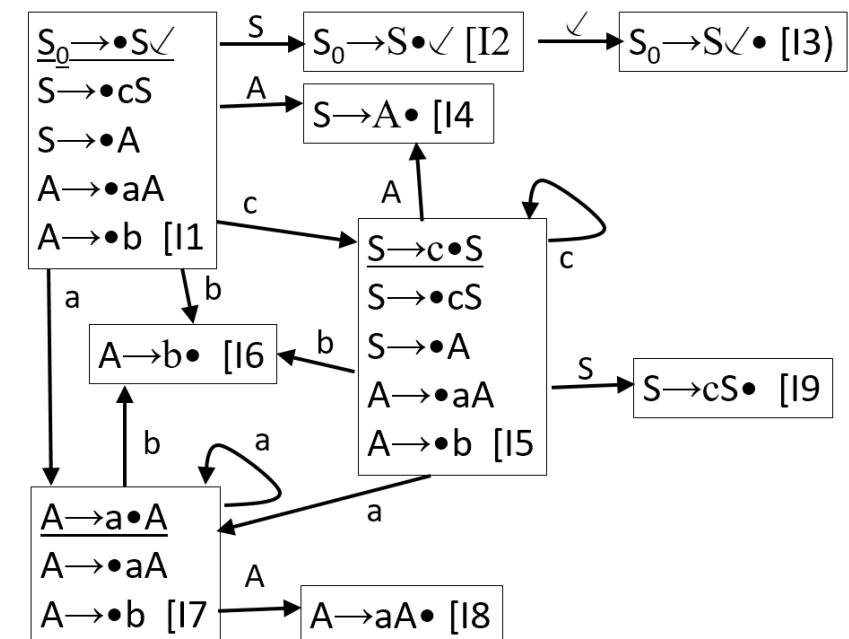
Esempio: stringa $cab \swarrow$

Stringa	Pila	Azione
$cab \swarrow$	I_1	
$ab \swarrow$	I_1cl_5	
$b \swarrow$	$I_1cl_5al_7$	
\swarrow	$I_1cl_5al_7bl_6$	Riduzione $A \rightarrow b$
\swarrow	$I_1cl_5al_7Al_8$	Riduzione $A \rightarrow aA$
\swarrow	$I_1cl_5Al_4$	Riduzione $S \rightarrow A$



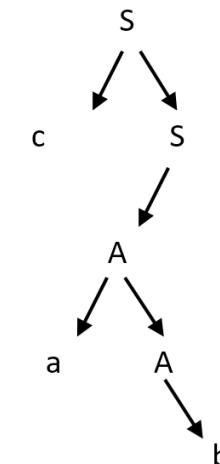
Esempio: stringa $cab \swarrow$

Stringa	Pila	Azione
$cab \swarrow$	I_1	
$ab \swarrow$	I_1cl_5	
$b \swarrow$	$I_1cl_5al_7$	
\swarrow	$I_1cl_5al_7bl_6$	Riduzione $A \rightarrow b$
\swarrow	$I_1cl_5al_7Al_8$	Riduzione $A \rightarrow aA$
\swarrow	$I_1cl_5Al_4$	Riduzione $S \rightarrow A$
\swarrow	I_1cl_5S	



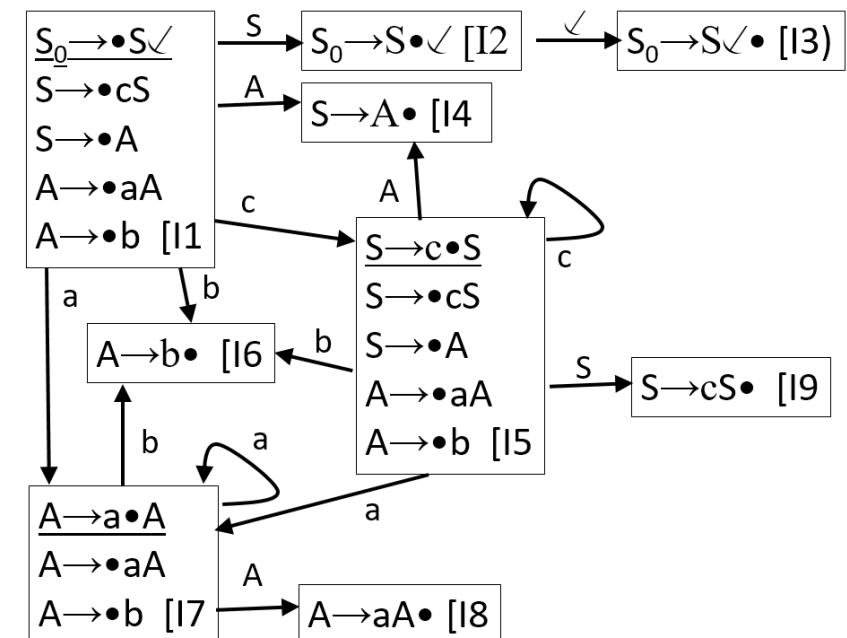
Esempio: stringa $cab \swarrow$

Stringa	Pila	Azione
$cab \swarrow$	I_1	
$ab \swarrow$	I_1cl_5	
$b \swarrow$	$I_1cl_5al_7$	
\swarrow	$I_1cl_5al_7bl_6$	Riduzione $A \rightarrow b$
\swarrow	$I_1cl_5al_7Al_8$	Riduzione $A \rightarrow aA$
\swarrow	$I_1cl_5Al_4$	Riduzione $S \rightarrow A$
\swarrow	$I_1cl_5Sl_9$	Riduzione $S \rightarrow cS$



Esempio: stringa $cab \swarrow$

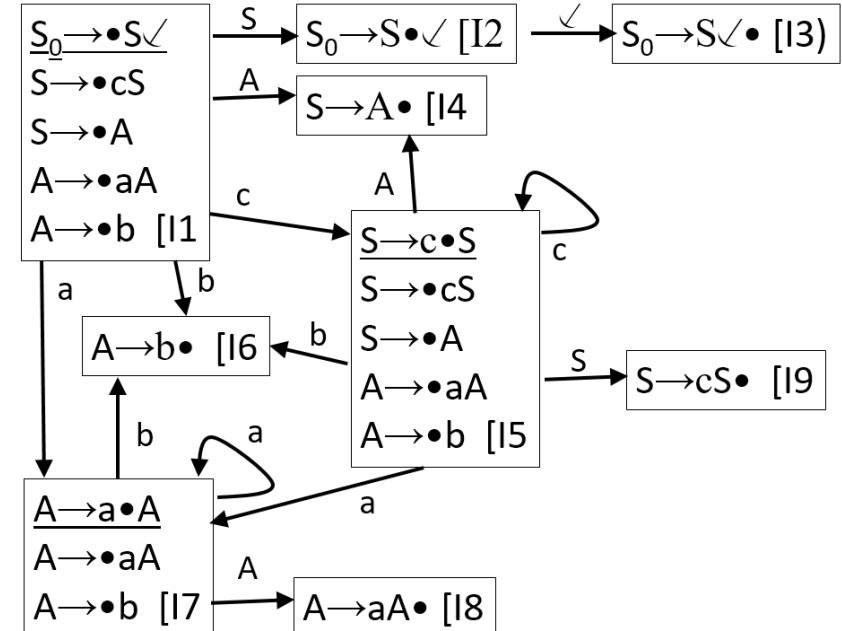
Stringa	Pila	Azione
$cab \swarrow$	I_1	
$ab \swarrow$	I_1cl_5	
$b \swarrow$	$I_1cl_5al_7$	
\swarrow	$I_1cl_5al_7bl_6$	Riduzione $A \rightarrow b$
\swarrow	$I_1cl_5al_7Al_8$	Riduzione $A \rightarrow aA$
\swarrow	$I_1cl_5Al_4$	Riduzione $S \rightarrow A$
\swarrow	$I_1cl_5Sl_9$	Riduzione $S \rightarrow cS$
\swarrow	I_1S	



Parsing Ascendente

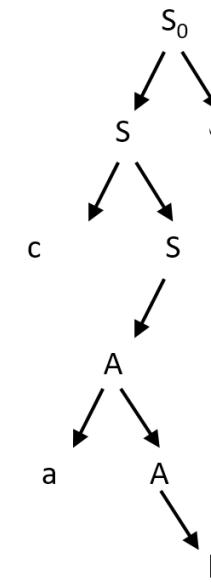
Esempio: stringa *cab* ↴

Stringa	Pila	Azione
cab	I_1	
ab	I_1cl_5	
b	$I_1cl_5al_7$	
	$I_1cl_5al_7bl_6$	Riduzione $A \rightarrow b$
	$I_1cl_5al_7Al_8$	Riduzione $A \rightarrow aA$
	$I_1cl_5Al_4$	Riduzione $S \rightarrow A$
	$I_1cl_5Sl_9$	Riduzione $S \rightarrow cS$
	I_1Sl_2	



Parsing Ascendente

Esempio: stringa cab	Stringa	Pila	Azione
	cab	l_1	
	ab	l_1cl_5	
	b	$l_1cl_5al_7$	
		$l_1cl_5al_7bl_6$	Riduzione $A \rightarrow b$
		$l_1cl_5al_7Al_8$	Riduzione $A \rightarrow aA$
		$l_1cl_5Al_4$	Riduzione $S \rightarrow A$
		$l_1cl_5Sl_9$	Riduzione $S \rightarrow cS$
		l_1Sl_2	
		l_1Sl_2	Riduzione $S_0 \rightarrow S$



Esempio: stringa $cab \swarrow$

Stringa	Pila	Azione
$cab \swarrow$	I_1	
$ab \swarrow$	$I_1 c I_5$	
$b \swarrow$	$I_1 c I_5 a I_7$	
\swarrow	$I_1 c I_5 a I_7 b I_6$	Riduzione $A \rightarrow b$
\swarrow	$I_1 c I_5 a I_7 A I_8$	Riduzione $A \rightarrow aA$
\swarrow	$I_1 c I_5 A I_4$	Riduzione $S \rightarrow A$
\swarrow	$I_1 c I_5 S I_9$	Riduzione $S \rightarrow cS$
\swarrow	$I_1 S I_2$	
$I_1 S I_2 \swarrow I_3$	$I_1 S I_2$	Riduzione $S_0 \rightarrow S \swarrow$
$I_1 S_0$		OK

Esercizio

- $\Sigma = \{a, b, x, y, \swarrow\}$, $V = \{S_0, S, A, B\}$
- $S_0 \rightarrow S \swarrow$
 $S \rightarrow A B y$
 $A \rightarrow a A$
 $A \rightarrow x$
 $B \rightarrow B b$
 $B \rightarrow y$
- È LR(0)?

Insiemi di Prospezione LALR(1)

- Come potenziare la tecnica LR(0)?
- introducendo gli **Insiemi di Prospezione (Look-Ahead Set)** per le candidate di riduzione che creano un conflitto

Insiemi di Prospezione LALR(1)

- Dato l'automa di tipo LR(0), ma **NON LR(0)**,
- negli stati non LR(0) si associano alle **candidate di riduzione** gli insiemi di prospezione $LA(s, N \rightarrow \alpha \bullet)$ o, se lo stato s è sottinteso, $LA(N \rightarrow \alpha \bullet)$.
- $LA(s, N \rightarrow \alpha \bullet) = \{a \in \Sigma \mid \exists S_o \xrightarrow{*} \gamma N a z \rightarrow \gamma \alpha a z \text{ AND } \delta^*(I_1, \gamma \alpha) = s\}$
derivazione destra

Condizioni LALR(1)

- Per ogni stato s contenente $LA(s, N \rightarrow \alpha \bullet)$
- Per ogni coppia di candidate
 $N_1 \rightarrow \alpha_1 \bullet$ e $N_2 \rightarrow \alpha_2 \bullet$
 deve essere $LA(s, N_1 \rightarrow \alpha_1 \bullet) \cap LA(s, N_2 \rightarrow \alpha_2 \bullet) = \emptyset$
- Per ogni candidata di riduzione $N \rightarrow \alpha \bullet$
 deve essere
 $LA(s, N \rightarrow \alpha \bullet) \cap \{b \in \Sigma \mid \delta(s, b) \text{ è definita}\} = \emptyset$
- Se queste due condizioni sono verificate, lo stato s viene detto **ADEGUATO**.

Grammatica LALR(1)

- La grammatica G è LALR(1)
 se **Tutti gli Stati dell'Automa sono ADEGUATI**
 (uno stato LR(0) è, di per sé, adeguato).

Metodo Operativo Calcolo LA

In uno stato s , data una candidata $c : N \rightarrow \alpha \bullet \beta$

vogliamo calcolare $Seg_s(N \rightarrow \alpha \bullet \beta)$

- Se $|\alpha| > 0$, candidata core, i seguiti arrivano da stati esterni:

$$Seg_s(N \rightarrow \alpha \bullet \beta) = \cup_{s_j} Seg_{s_j}(N \rightarrow \bullet \alpha \beta)$$

per ogni stato s_j per cui $\delta^*(s_j, \alpha)$ è definita

- Se $|\alpha| = 0$, candidata di completamento, i seguiti arrivano da altre candidate nello stesso stato s :

$$Seg_s(N \rightarrow \bullet \beta) = \cup_i Seg_s(N, c_i)$$

per ogni candidata $c_i : M_i \rightarrow \alpha_i \bullet N \beta_i$

In particolare:

- $Seg_s(N, c_i) = Ini(\beta_i)$ se $\neg Annulabile(\beta_i)$
- $Seg_s(N, c_i) = Ini(\beta_i) \cup Seg_s(M_i \rightarrow \alpha_i \bullet N \beta_i)$ se $Annulabile(\beta_i)$

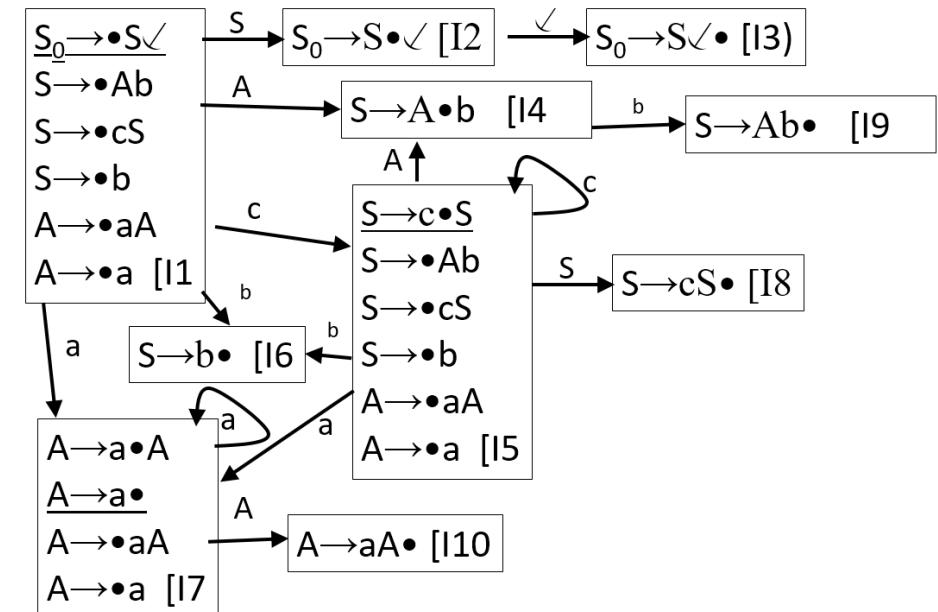
In uno stato s NON LR(0), data una candidata di riduzione $N \rightarrow \alpha \bullet$, si calcola

$$LA(N \rightarrow \alpha \bullet) = Seg_s(N \rightarrow \alpha \bullet)$$

La grammatica è LR(0)?

$\Sigma = \{a, b, c\}$, $V = \{S, A, \}$

- $S_0 \rightarrow S \downarrow$
- $S \rightarrow Ab$
- $S \rightarrow cS$
- $S \rightarrow b$
- $A \rightarrow aA$
- $A \rightarrow a$

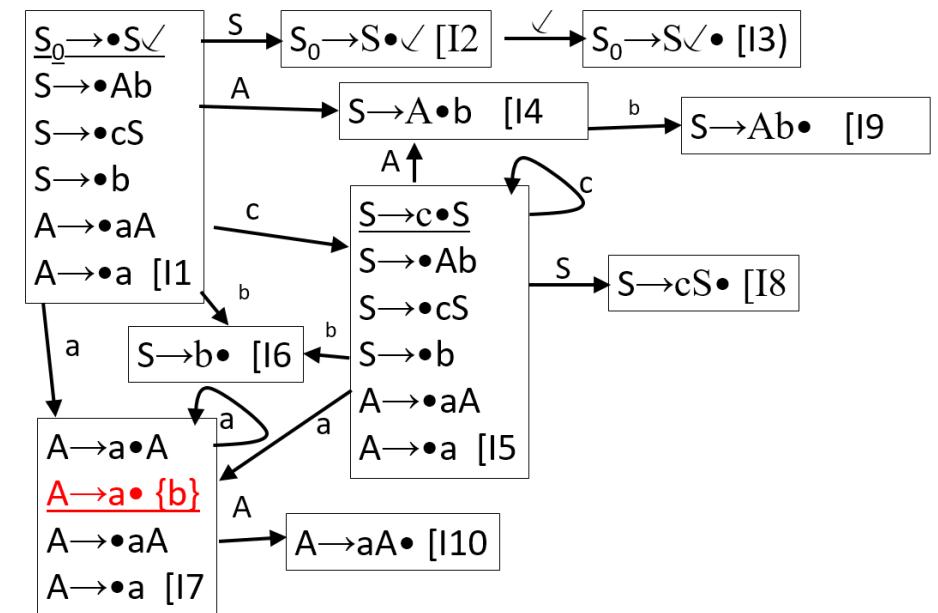


Parsing Ascendente

Parsing Ascendente

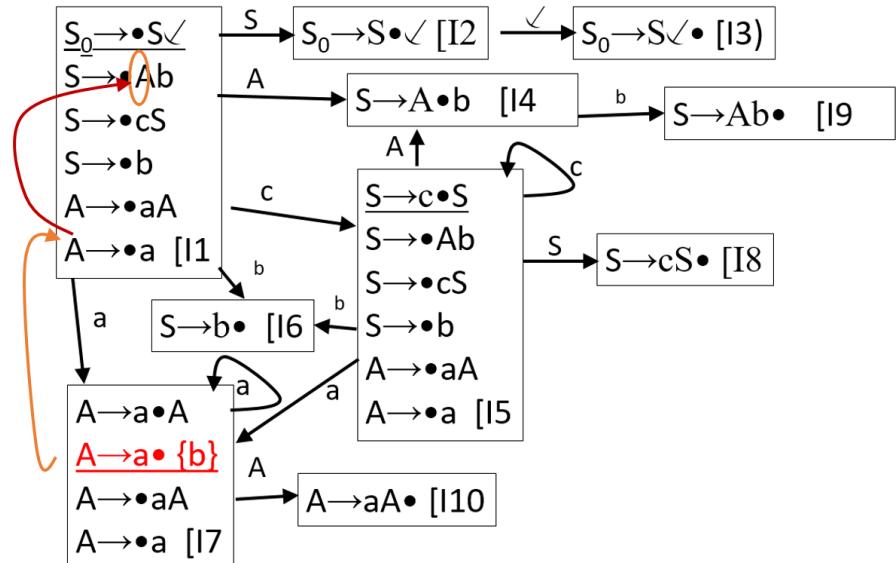
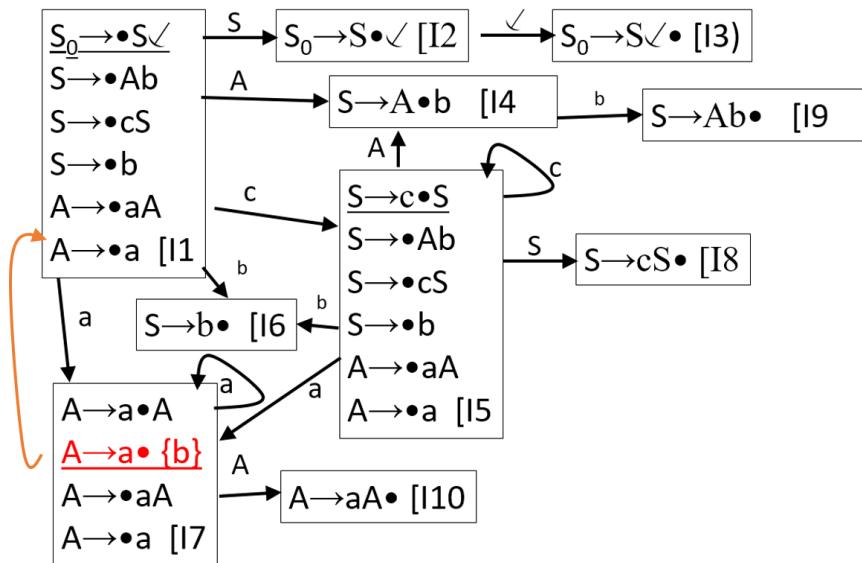
La grammatica è LR(0)?

- No, la grammatica NON è LR(0)
- Passiamo alla tecnica LALR(1): calcoliamo i look-ahead set per le candidate di riduzione



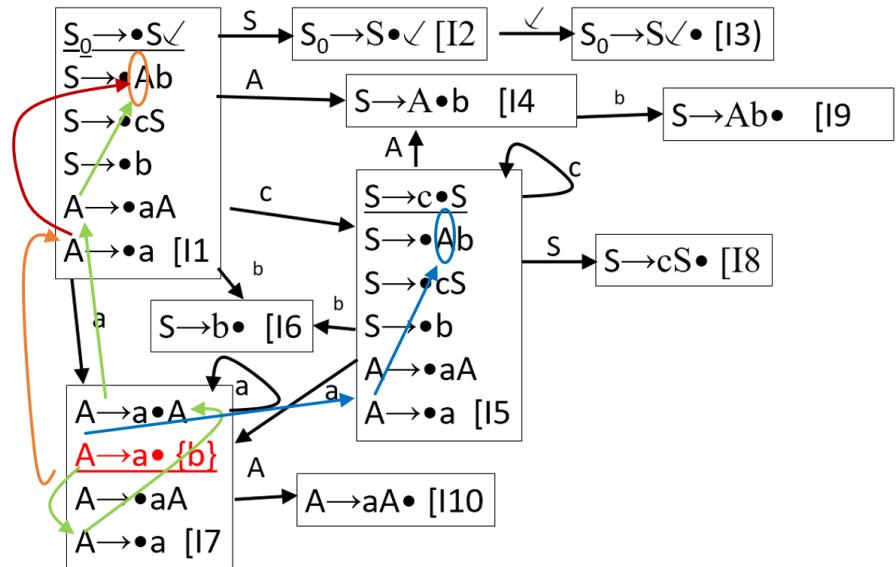
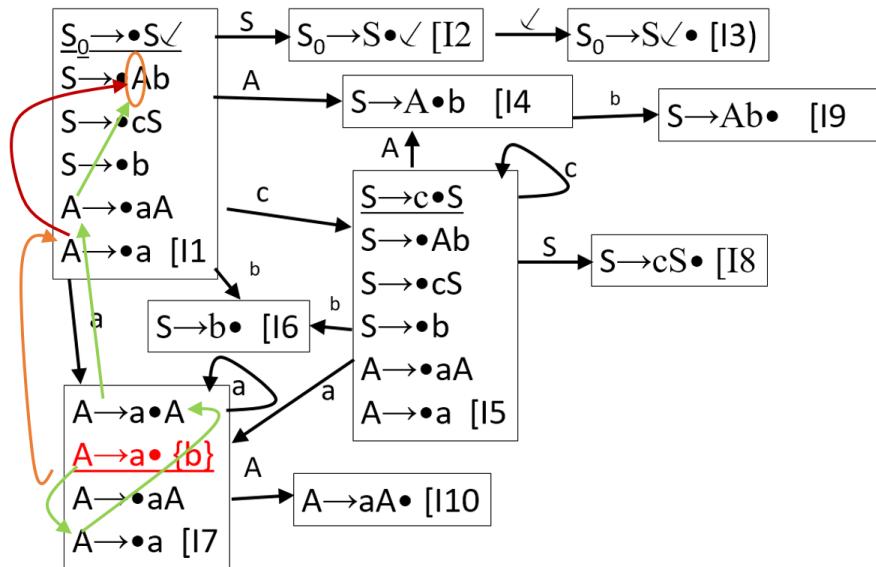
Parsing Ascendente

Parsing Ascendente



Parsing Ascendente

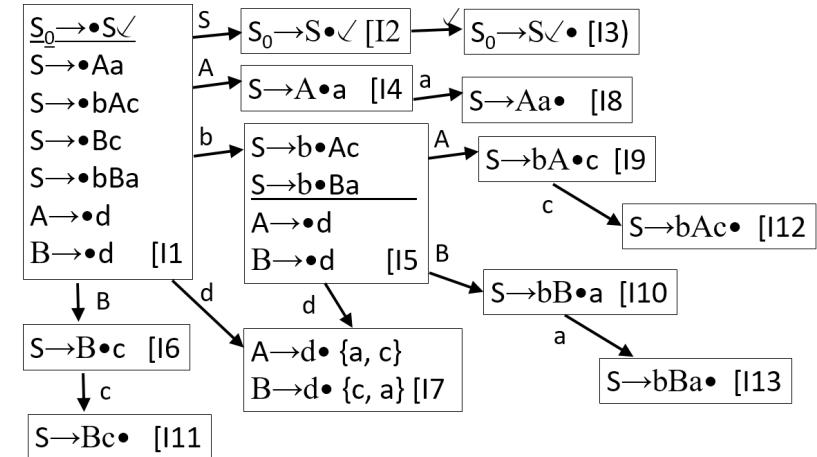
Parsing Ascendente



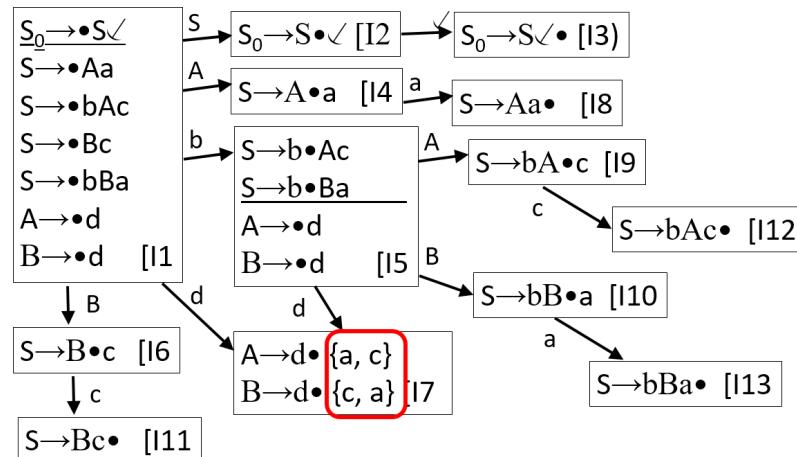
Superare i Limiti di LALR(1)

Consideriamo la grammatica seguente

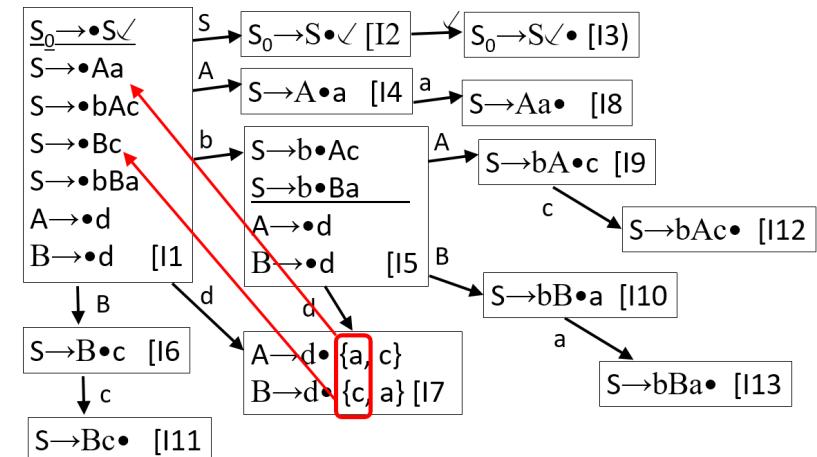
- $S_0 \rightarrow S \downarrow$
- $S \rightarrow A \ a$
- $S \rightarrow b \ A \ c$
- $S \rightarrow B \ c$
- $S \rightarrow b \ B \ a$
- $A \rightarrow d$
- $B \rightarrow d$

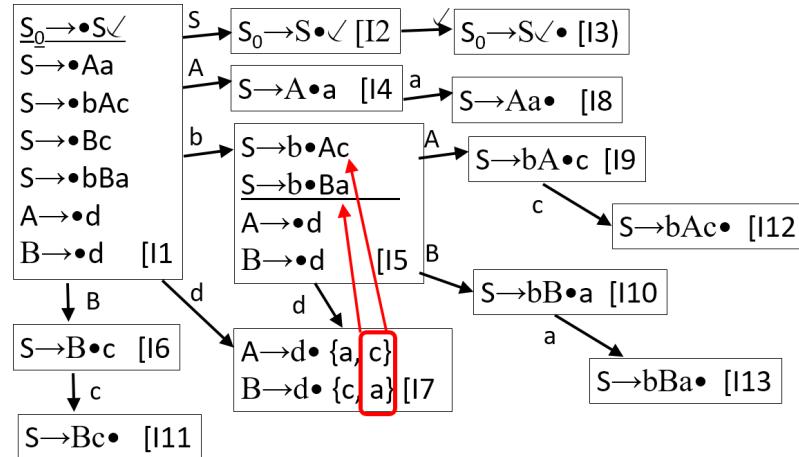


Parsing Ascendente



Parsing Ascendente





Parsing Ascendente

Automa LR(1)

- È la versione più potente del parsing ascendente con prospettiva 1
- Cerca di separare la provenienza dei look-ahead, per gestire separatamente i contesti dei sotto-alberi
- Come? Calcolando i look-ahead di ogni candidata fin dall'inizio.
- Ogni candidata ha **SEMPRE** associato un Look-ahead set.

Creazione Automa LR(1)

- Nello stato I_1 , la candidata core $S_0 \rightarrow \bullet S \checkmark$, ha $LA(I_1, S_0 \rightarrow \bullet S \checkmark) = \emptyset$
- In ogni stato s , nelle candidate di completamento $N \rightarrow \bullet \beta$, e le candidate $c_j : M_j \rightarrow \alpha_j \bullet N \beta_j$ dove $Seg(N, c_j) = Ini(\beta_j)$ se $\neg Annulabile(\beta_j)$ o $Seg(N, c_j) = Ini(\beta_j) \cup LA(s, M_j \rightarrow \alpha_j \bullet N \beta_j)$ $LA(s, N \rightarrow \bullet \beta) = \bigcup_{c_j} Seg(N, c_j)$

Creazione Automa LR(1)

- Le candidate core di uno stato diverso da I_1 mantengono il look-ahead set delle candidate di provenienza
- Quindi, se esiste $s \xrightarrow{X} \bar{s}$, (con $X \in (\Sigma \cup V)$), abbiamo $N \rightarrow \alpha \bullet X \beta \in s$ e $N \rightarrow \alpha X \bullet \beta \in \bar{s}$ con $LA(\bar{s}, N \rightarrow \alpha X \bullet \beta) = LA(s, N \rightarrow \alpha \bullet X \beta)$

Creazione Automa LR(1)

- I look-ahead set contribuiscono anche all'identità dello stato, quindi
 - se lo spostamento genera uno stato con le stesse candidate core di uno stato già esistente, ma con look-ahead set diversi,
 - si crea un nuovo stato

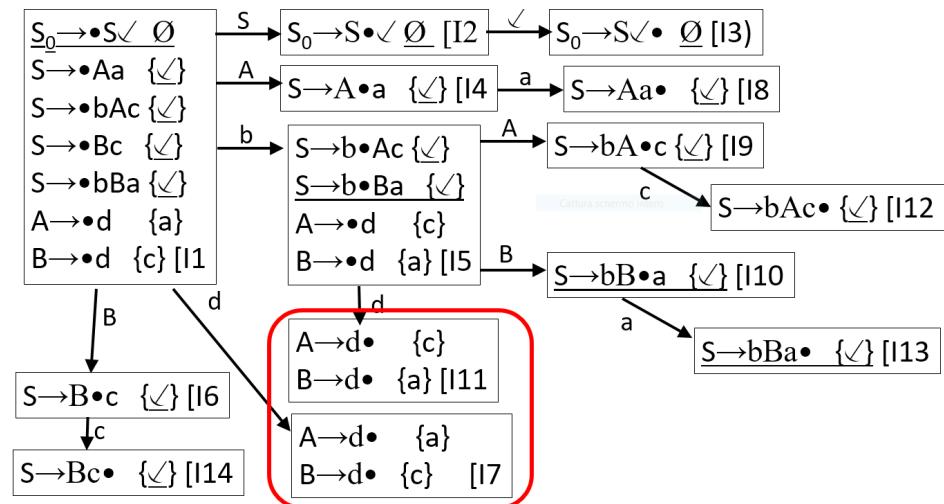
Condizioni LR(1)

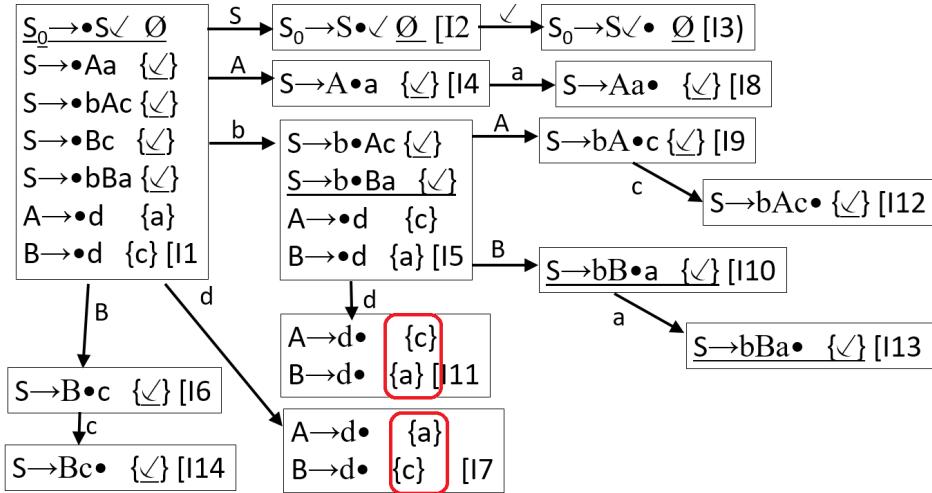
- Dato uno stato che presenta conflitti spostamento/riduzione o riduzione/riduzione,
- Le condizioni sono le stesse del caso LALR(1), cioè
 - i look-ahead set delle candidate di riduzione devono essere disgiunti tra di loro
 - e devono essere disgiunti dalle mosse uscenti

Automa LR(1)

Riconsideriamo la grammatica seguente

- $S_0 \rightarrow S \swarrow$
- $S \rightarrow A a$
- $S \rightarrow b A c$
- $S \rightarrow B c$
- $S \rightarrow b B a$
- $A \rightarrow d$
- $B \rightarrow d$





Linguaggi Formali e Compilatori

Argomento 06:

Semantica dei Linguaggi

Prof. Giuseppe Psaila

Laurea Magistrale in Ingegneria Informatica
Università di Berga

Semantica dei Linguaggi

- Le grammatiche BNF hanno la capacità di specificare strutture sintattiche complesse, che i parser deterministici riescono a riconoscere in molti casi
- Ma trattano i terminali solo in base alla loro categoria lessicale, non in base al loro valore
- In altre parole, non viene considerato il **Significato** delle frasi del linguaggio (**la semantica del linguaggio**)

Semantica dei Linguaggi

- Che cosa è il **Significato** di una frase?
- Dipende dal linguaggio
- Nelle espressioni matematiche, il significato è il **Valore** dell'espressione
- Nei linguaggi di programmazione procedurale, il significato è la versione in linguaggio macchina

- **Correttezza Semantica:** va oltre la correttezza **Sintattica**, perché per essere verificata occorre usare i valori dei terminali
- Nell'XML, la correttezza semantica è necessaria per verificare se i marcatori di apertura e chiusura corrispondenti hanno lo stesso nome
- Nei linguaggi di programmazione procedurale, la correttezza semantica è necessaria per gestire le definizioni delle variabili prima del loro uso e i tipi delle espressioni

- Per gestire gli aspetti semantici (la **Semantica del Linguaggio**)
- occorre introdurre un nuovo formalismo,
- che superi i limiti delle grammatiche BNF
- Questo formalismo prende il nome di **GRAMMATICHE AD ATTRIBUTI**

Approccio

- Non si buttano via le grammatiche BNF, perchè fanno egregiamente il loro lavoro
- Le grammatiche ad attributi vengono **Aggiunte** alle grammatiche BNF
- Infatti, le grammatiche BNF e gli alberi sintattici forniscono lo **Scheletro** sul quale fare la **Valutazione Semantica**

Attributo

- Un **Attributo** a è associato all'occorrenza di un **Non-Terminale** N
denota una proprietà dell'occorrenza di quel non-terminale
- Con $Attr(N)$ indichiamo l'**Insieme degli Attributi del Non-Terminale** N .
- Nell'albero sintattico, tutte le occorrenze N_i dello stesso Non-Terminale N hanno lo stesso insieme di attributi, cioè $Attr(N_i) = Attr(N)$.

Regola Semantica

- Si consideri una **Regola di Produzione**

$$p : X_0 \rightarrow X_1 X_2 \dots X_h$$

- Alla regola di produzione p si associa un insieme di **REGOLE SEMANTICHE** $R(p)$

- Una **REGOLA SEMANTICA** ha la forma

$$a \text{ of } X_i := f(a_1 \text{ of } X_{p1}, \dots, a_j \text{ of } X_{pj}, \dots, a_n \text{ of } X_{pn})$$

dove $i, pj \in \{0, 1, 2, \dots, h\}$

$$a \in Attr(X_i) \text{ e } a_j \in Attr(X_{pj})$$

- f è detta **Funzione Semantica**.

Regola Semantica

Notazione alternativa

- Data la regola di produzione $p : X_0 \rightarrow X_1 X_2 \dots X_h$

- La regola semantica

$$a \text{ of } X_i := f(a_1 \text{ of } X_{p1}, \dots, a_j \text{ of } X_{pj}, \dots, a_n \text{ of } X_{pn})$$

può essere scritta anche come

$$X[i].a := f(X[p1].a_1, \dots, X[pj].a_j, \dots, X[pn].a_{pn})$$

Attributi Ereditati

- Data la regola di produzione $p : X_0 \rightarrow X_1 X_2 \dots X_h$
 - e data la regola semantica
- $$a \text{ of } X_i := f(a_1 \text{ of } X_{p1}, \dots, a_j \text{ of } X_{pj}, \dots, a_n \text{ of } X_{pn})$$
- l'attributo a of X_i è detto **EREDITATO** se $i \geq 1$
(cioè l'attributo a appartiene ad un **figlio** della regola di produzione).

Attributi Sintetizzati

- Data la regola di produzione $p : X_0 \rightarrow X_1 X_2 \dots X_h$
 - e data la regola semantica
- $$a \text{ of } X_i := f(a_1 \text{ of } X_{p1}, \dots, a_j \text{ of } X_{pj}, \dots, a_n \text{ of } X_{pn})$$
- l'attributo a of X_i è detto **SINTETIZZATO** se $i = 0$
(cioè l'attributo a appartiene al **padre** della regola di produzione).

Attributi Ereditati e Sintetizzati

- Dato un Non-Terminale N , indichiamo con $Ere(N) = \{a_1, a_2, \dots\}$ l'insieme dei suoi attributi ereditati, cioè l'insieme degli attributi $a \in Attr(N)$ per cui esista almeno una regola semantica $a \text{ of } N_i := \dots$, con $i \geq 1$.
- Dato un Non-Terminale N , indichiamo con $Sin(N) = \{a_1, a_2, \dots\}$ l'insieme dei suoi attributi sintetizzati, cioè l'insieme degli attributi $a \in Attr(N)$ per cui esista almeno una regola semantica $a \text{ of } N_i := \dots$, con $i = 0$.

Grammatiche ad Attributi Ben Formate

- Per ogni Non-Terminale $N \in V$, $Ere(N) \cup Sin(N) = Attr(N)$
- Per ogni Non-Terminale figlio $X_i \in V$ (con $i \geq 1$), per ogni attributo $a \in Ere(X_i)$ deve esserci **una e una sola** regola semantica che lo definisce;
- Per ogni regola di produzione $p : X_0 \rightarrow X_1 X_2 \dots X_h$, per ogni attributo $a \in Sin(X_0)$ deve esserci **una e una sola** regola semantica che lo definisce.

Semantica dei Linguaggi

Attributi dei Terminali

- Dal punto di vista sintattico, non ha importanza conoscere l'effettiva stringa associata all'occorrenza di un simbolo terminale
- Ma dal punto di vista semantico, spesso è necessario (pensate ad XML o ai nomi delle variabili).
- Ogni simbolo terminale $t \in \Sigma$ ha un **Attributo Predefinito** denominato Value, che indica la stringa di testo associata all'occorrenza del terminale.
- Possiamo dire che $Sin(t) = \{Value\}$ e che $Ere(t) = \emptyset$.

Semantica dei Linguaggi

Esempio

- 1) $S \rightarrow L$
- 2) $L \rightarrow A L$
- 3) $L \rightarrow B L$
- 4) $L \rightarrow \epsilon$
- 5) $A \rightarrow a$
- 6) $B \rightarrow b$

Vogliamo scrivere una grammatica ad attributi che conta il numero di a e di b

Impostiamo gli Attributi

- $\text{Attr}(L) = \{\text{tota}, \text{totb}\}$
- $\text{Attr}(S) = \{\text{tota}, \text{totb}\}$
- $\text{Attr}(A) = \emptyset$
- $\text{Attr}(B) = \emptyset$

- 1) $S \rightarrow L$
 $S[0].\text{tota} := L[1].\text{tota}$
 $S[0].\text{totb} := L[1].\text{totb}$
- 2) $L \rightarrow A L$
 $L[0].\text{tota} := L[2].\text{tota} + 1$
 $L[0].\text{totb} := L[2].\text{totb}$
- 3) $L \rightarrow B L$
 $L[0].\text{tota} := L[2].\text{tota}$
 $L[0].\text{totb} := L[2].\text{totb} + 1$

- 4) $L \rightarrow \epsilon$
 $L[0].\text{tota} := 0$
 $L[0].\text{totb} := 0$
- 5) $A \rightarrow a$
- 6) $B \rightarrow b$

- Abbiamo usato la funzione semantica “+”, con la forma infissa
- Se vogliamo vederla in forma funzionale, possiamo introdurre la funzione Increase
- 2) $L \rightarrow A L$
 $L[0].\text{tota} := \text{Increase}(L[2].\text{tota})$
 $L[0].\text{totb} := L[2].\text{totb}$
- 3) $L \rightarrow B L$
 $L[0].\text{tota} := L[2].\text{tota}$
 $L[0].\text{totb} := \text{Increase}(L[2].\text{totb})$

- Alcune regole non usano alcuna funzione semantica
- 3) $L \rightarrow B L$
 $L[0].tota := L[2].tota$
- Sono dette **Regole di Propagazione**, perché propagano il valore di un attributo ad un altro attributo, senza alterarlo.

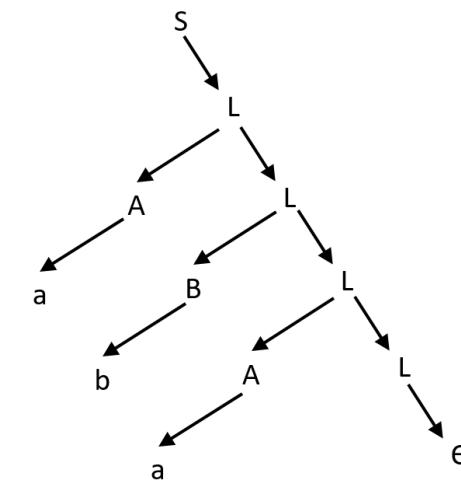
Dipendenza Funzionale

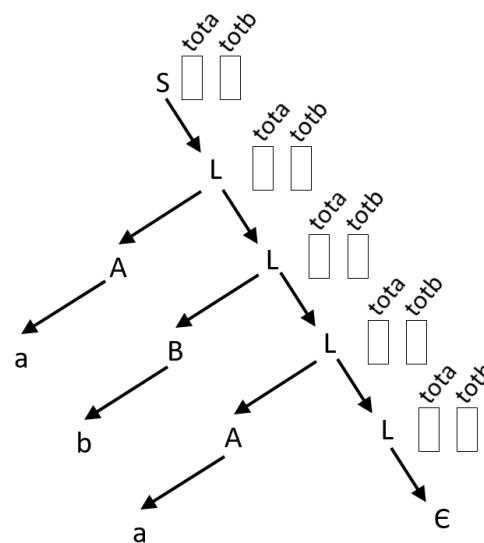
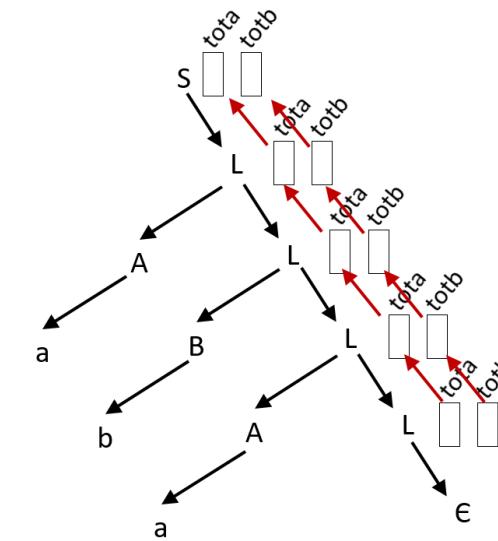
- Dato un albero sintattico e l'occorrenza di una regola di produzione che genera le occorrenze dei Non-Terminali e Terminali $p : X_0 \rightarrow X_1 X_2 \dots X_h$, si dice che a of X_i **DIPENDE** **FUNZIONALMENTE** da a_j of X_{pj} , indicato come a of $X_i \leftarrow a_j$ of X_{pj} (o anche come a_j of $X_{pj} \rightarrow a$ of X_i), se esiste una regola semantica associata a p tale che a of $X_i := f(\dots, a_j$ of $X_{pj}, \dots)$.
- a of $X_i \leftarrow a_j$ of X_{pj} viene detta **DIPENDENZA FUNZIONALE**.

Albero Sintattico Decorato

Albero Sintattico Decorato

- Data una stringa s e il corrispondente albero sintattico
- L'albero sintattico esteso con le **Occorrenze degli Attributi** e con le **Dipendenze Funzionali** indotte dalle regole semantiche viene detto **ALBERO DECORATO**



Albero Sintattico Decorato**Albero Sintattico Decorato**

Semantica dei Linguaggi

Valutazione Semantica

- Gli attributi presenti nell'albero sintattico decorato devono essere **Valutati**, cioè occorre valutare il loro valore, applicando le regole semantiche.
- Trascrivendo le regole semantiche, applicandole alle specifiche occorrenze degli attributi, otteniamo un **Sistema di EQUAZIONI SEMANTICHE**.
- Valutare gli attributi vuole dire **Risolvere il Sistema di Equazioni Semantiche**.

Semantica dei Linguaggi

Valutazione Semantica

- In termini pratici, le **Dipendenze Funzionali** forniscono la chiave per risolvere il sistema di equazioni semantiche
- Infatti, se calcoliamo l'**Ordinamento Topologico** delle occorrenze degli attributi sull'albero sintattico decorato in base alle dipendenze funzionali, otteniamo l'**Ordine** con cui le occorrenze degli attributi devono essere valutate

Valutazione Semantica

- Tuttavia, se il sistema di equazioni semantiche dà luogo a **Dipendenze Circolari**
 - il sistema di equazioni semantiche
NON HA ALCUNA SOLUZIONE
 - Condizione necessaria e sufficiente perché il sistema di equazioni semantiche abbia una e una sola soluzione è che la grammatica ad attributi sia ben formata e che le dipendenze funzionali non presentino circolarità

Valutazione Semantica

- Se si riesce a dimostrare che, per qualsiasi albero sintattico, il sistema di equazioni semantiche non presenta circolarità,
 - allora la valutazione semantica è **SEMPRE POSSIBILE**.
 - In tal caso, la Grammatica ad Attributi viene detta **Aciclica**.

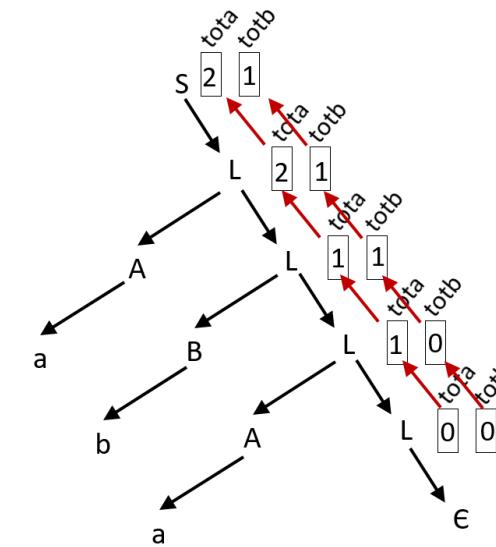
Semantica dei Linguaggi

Semantica dei Linguaggi

Albero Sintattico Decorato

Valutazione Semantica Base

- Dato l'albero sintattico decorato D
 - si calcola l'ordinamento topologico $O = \langle a_1, a_2, \dots \rangle$ (sequenza), in base alle dipendenze funzionali.
 - Ogni attributo $a_i \in O$ viene valutato applicando la corrispondente regola semantica, seguendo l'ordine con cui compare in O .



Osservazioni

- Gli attributi tota e totb sono sintetizzati.
Non ci sono attributi ereditati.
Questa grammatica viene detta
Puramente Sintetizzata.

Variante

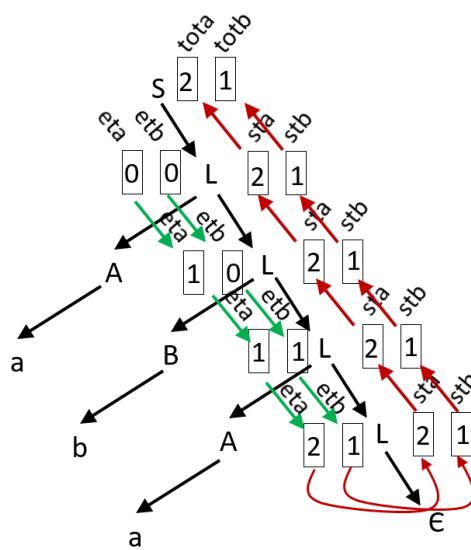
Adottiamo un approccio **Discendente**

- $\text{Attr}(L) = \{\text{eta,etb,sta,stb}\}$
- $\text{Attr}(S) = \{\text{tota,totb}\}$
- $\text{Attr}(A) = \emptyset$
- $\text{Attr}(B) = \emptyset$

- 1) $S \rightarrow L$
 $S[0].\text{tota} := L[1].\text{sta}$
 $S[0].\text{totb} := L[1].\text{stb}$
 $L[1].\text{eta} := 0$
 $L[1].\text{etb} := 0$
- 2) $L \rightarrow A L$
 $L[0].\text{sta} := L[2].\text{sta}$
 $L[0].\text{stb} := L[2].\text{stb}$
 $L[2].\text{eta} := L[0].\text{eta} + 1$
 $L[2].\text{etb} := L[0].\text{etb}$

- 3) $L \rightarrow B L$
 $L[0].\text{sta} := L[2].\text{sta}$
 $L[0].\text{stb} := L[2].\text{stb}$
 $L[2].\text{eta} := L[0].\text{eta}$
 $L[2].\text{etb} := L[0].\text{etb} + 1$
- 4) $L \rightarrow \epsilon$
 $L[0].\text{sta} := L[0].\text{eta}$
 $L[0].\text{stb} := L[0].\text{etb}$
- 5) $A \rightarrow a$
- 6) $B \rightarrow b$

Albero Sintattico Decorato



Semantica dei Linguaggi

Gestione delle Variabili in un Linguaggio di Programmazione

- 1) PR → LI
- 2) LI → I LI
- 3) LI → ε
- 4) I → VDEF
- 5) I → VUSE
- 6) VDEF → Def Name
- 7) VUSE → Name

Semantica dei Linguaggi

Gestione delle Variabili in un Linguaggio di Programmazione

- Il programma è una lista di istruzioni
- Un'istruzione è o una definizione di variabile o un uso di variabile
- Una definizione di variabile inizia con la parola chiave "DEF" (terminale Def) seguita dal nome (terminale Name)
- L'uso di una variabile è dato dal solo nome della variabile (terminale Name)

Controlli Semanticci

- Verificare che tutte le variabili usate siano state precedentemente definite
- Approccio: Dobbiamo costruire la **Tabella delle Variabili**
- che useremo per controllare che la variabile usata sia definita
- La condizione di errore è il risultato della valutazione (attributo dell'assioma).
- Gli errori devono essere segnalati nell'ordine naturale.

Attributi

- $Ere(PR) = \emptyset$
 $Sin(PR) = \{\text{err}\}$
- $Ere(LI) = \{\text{ev}\}$
 $Sin(LI) = \{\text{err}\}$
- $Ere(I) = \{\text{ev}\}$
 $Sin(I) = \{\text{err}, \text{ sv}\}$
- $Ere(VDEF) = \{\text{ev}\}$
 $Sin(VDEF) = \{\text{err}, \text{ sv}\}$
- $Ere(VUSE) = \{\text{ev}\}$
 $Sin(VUSE) = \{\text{err}\}$

Semantica dei Linguaggi

- 1) $PR \rightarrow LI$
 $LI[1].ev := \emptyset$
 $PR[0].err := LI[1].err$
- 2) $LI \rightarrow I$ LI
 $I[1].ev := LI[0].ev$
 $LI[2].ev := I[1].sv$
 $LI[0].err := I[1].err \text{ OR } LI[2].err$
- 3) $LI \rightarrow \epsilon$
 $LI[0].err := \text{False}$

Semantica dei Linguaggi

- 4) $I \rightarrow VDEF$
 $VDEF[1].ev := I[0].ev$
 $I[0].sv := VDEF[1].sv$
 $I[0].err := VDEF[1].err$
- 5) $I \rightarrow VUSE$
 $VUSE[1].ev := I[0].ev$
 $I[0].sv := I[0].ev$
 $I[0].err := VUSE[1].err$

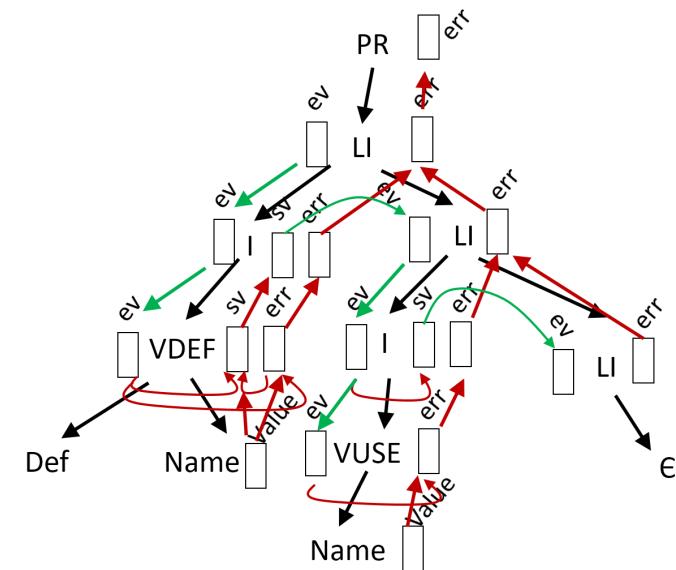
Albero Sintattico Decorato

- 6) VDEF → Def Name

```
VDEF[0].err :=  
    CheckDef(VDEF[0].ev, Name[2].Value)  
VDEF[0].sv :=  
    AddDef(VDEF[0].ev, Name[2].Value,  
           VDEF[0].err)
```

- 7) VUSE → Name

```
VUSE[0].err :=  
    CheckUse(VUSE[0].ev, Name[1].Value)
```

**Correttezza dei Documenti XML**

- 1) $S \rightarrow E$
- 2) $E \rightarrow o A e$
- 3) $E \rightarrow o A n C c$
- 4) $A \rightarrow a u s A$
- 5) $A \rightarrow \epsilon$
- 6) $C \rightarrow t C$
- 7) $C \rightarrow E C$
- 8) $C \rightarrow \epsilon$

Correttezza dei Documenti XML

- Vogliamo controllare che il nome del marcatore di chiusura (terminale n) sia lo stesso del corrispondente marcatore di apertura (terminale o).
- Vogliamo controllare che un elemento non abbia nomi di attributi duplicati.

Attributi

- $Ere(S) = \emptyset$
 $Sin(S) = \{\text{err}\}$
- $Ere(E) = \emptyset$
 $Sin(E) = \{\text{err}, \text{ bad}\}$
- $Ere(C) = \emptyset$
 $Sin(C) = \{\text{err}\}$
- $Ere(A) = \{\text{attrs}\}$
 $Sin(A) = \{\text{err}, \text{ dup}\}$

- 1) $S \rightarrow E$
 $S[0].\text{err} := E[1].\text{err}$
- 2) $E \rightarrow o A e$
 $E[0].\text{err} := A[2].\text{err}$
 $A[2].\text{attrs} := \emptyset$
 $E[0].\text{bad} := \text{False}$
- 3) $E \rightarrow o A n C c$
 $A[2].\text{attrs} := \emptyset$
 $E[0].\text{bad} :=$
CheckNames($o[1].\text{Value}$, $c[5].\text{Value}$)
 $E[0].\text{err} :=$
 $A[2].\text{err} \text{ OR } C[4].\text{err} \text{ OR } E[0].\text{bad}$

- 4) $A \rightarrow a u s A$
 $A[0].\text{dup} :=$
CheckAttr($A[0].\text{attrs}$, $a[1].\text{value}$)
 $A[4].\text{attrs} := \text{AddAttr}(A[0].\text{attrs},$
 $a[1].\text{Value}$, $A[0].\text{dup}$)
 $A[0].\text{err} := A[0].\text{dup} \text{ OR } A[4].\text{err}$
- 5) $A \rightarrow \epsilon$
 $A[0].\text{err} := \text{False}$
 $A[0].\text{dup} := \text{False}$

- 6) $C \rightarrow t C$
 $C[0].\text{err} := C[2].\text{err}$
- 7) $C \rightarrow E C$
 $C[0].\text{err} := E[1].\text{err} \text{ OR } C[2].\text{err}$
- 8) $C \rightarrow \epsilon$
 $C[0].\text{err} := \text{False}$

Valutazione Efficiente

- Il metodo di valutazione generale delle grammatiche ad attributi è efficace ma non efficiente
- Sono state identificate delle tecniche di valutazione efficiente, adatte a specifiche **Famiglie** di Grammatiche ad Attributi.

Valutazione One-Sweep

- La valutazione One-Sweep (a una spazzata dell'albero) è molto efficiente,
- perché viene fatta facendo una sola visita dell'albero sintattico
- Il caso particolare detto **TIPO L** può essere valutato durante il parsing discendente.

Semantica dei Linguaggi

Condizioni One-Sweep

• Condizione 1

Per ogni regola di produzione $p : X_0 \rightarrow X_1 X_2 \dots X_h$, non devono esserci regole semantiche del tipo
 $a \text{ of } X_i := f(\dots, a_j \text{ of } X_{pj}, \dots)$.
 con $a \text{ of } X_i \in Ere(X_i)$, con $i \geq 1$
 e con $a_j \text{ of } X_{pj} \in Sin(X_{pj})$, con $pj = 0$

Semantica dei Linguaggi

Condizioni One-Sweep

- Per ogni regola di produzione $p : X_0 \rightarrow X_1 X_2 \dots X_h$, si calcola il **BROTHER GRAPH** $BG = (N, E)$
 Nodi $N = \{X_1, \dots, X_h\}$
 Archi $E = \{X_{pj} \rightarrow X_i, \dots\}$ se esiste una regola semantica
 $a \text{ of } X_i := f(\dots, a_j \text{ of } X_{pj}, \dots)$,
 con $i, pj > 0$.
- Non si considerano le dipendenze tali che
 - $i = pj$ e
 - $a_j \in Ere(X_{pj})$

Condizioni One-Sweep

- **Condizione 2**

Si verifica se il Brother Graph è **Aciclico**, calcolando l'ordinamento topologico $O(p)$:
se non lo è, si genera errore

- La grammatica ad attributi è **One-Sweep** se le condizioni 1 e 2 valgono per tutte le reg. di prod.

Valutazione One-Sweep

- Per ogni regola di produzione p , prendiamo l'ordinamento topologico $O(p) = \langle X_{p1}, X_{p2}, \dots \rangle$
- Una procedura che visita il non-terminale X_0 in cui è radicata la regola di produzione p
Per ogni figlio $X_{pi} \in O(p)$ (nell'ordine)
 - valuta tutti gli attributi $a \in Ere(X_{pi})$ (rispettando eventuali **Dipendenze Interne**)
 - Visita X_{pi}
- Valuta tutti gli attributi sintetizzati $a \in Sin(X_0)$ (rispettando eventuali **Dipendenze Interne**)
Si termina la visita

Valutazione One-Sweep

- Cosa si intende con **rispettando eventuali dipendenze interne?**
- Prendiamo un non-terminale X_i (con $i \geq 0$) se esiste una regola semantica $a \text{ of } X_i := f(\dots, b \text{ of } X_i, \dots)$.
- Questa è una **Dipendenza Interna**: occorre valutare prima b e poi a

Esempio: XML

- La regola 4 non rispetta la condizione 1
- 4) $A \rightarrow a \ u \ s \ A$
 $A[0].dup :=$
 $\text{CheckAttr}(A[0].attrs, a[1].value)$
 $A[4].attrs := \text{AddAttr}(A[0].attrs,$
 $\text{a[1].Value, A[0].dup})$
 $A[0].err := A[0].dup \text{ OR } A[4].err$

Brother Graphs

1) E

2) A

3) A C

4) A

5)

6) C

7) E C

8)

Grammatica Modificata

- Spostiamo l'attributo `dup` del Non-Terminale A da sintetizzato ad ereditato.
- $Ere(S) = \emptyset$
 $Sin(S) = \{\text{err}\}$
- $Ere(E) = \emptyset$
 $Sin(E) = \{\text{err, bad}\}$
- $Ere(C) = \emptyset$
 $Sin(C) = \{\text{err}\}$
- $Ere(A) = \{\text{attrs, dup}\}$
 $Sin(A) = \{\text{err}\}$

Semantica dei Linguaggi

- 2) $E \rightarrow o A e$
 $E[0].err := A[2].err$
 $A[2].attrs := \emptyset$
 $E[0].bad := \text{False}$
 $A[2].dup := \text{False}$

- 3) $E \rightarrow o A n C c$
 $A[2].attrs := \emptyset$
 $E[0].bad :=$
 $\quad \text{CheckNames}(o[1].Value, c[5].Value)$
 $E[0].err :=$
 $\quad A[2].err \text{ OR } C[4].err \text{ OR } E[0].bad$
 $A[2].dup := \text{False}$

Semantica dei Linguaggi

- 4) $A \rightarrow a u s A$
 $A[4].dup :=$
 $\quad \text{CheckAttr}(A[0].attrs, a[1].value)$
 $A[4].attrs := \text{AddAttr}(A[0].attrs,$
 $\quad a[1].Value, A[4].dup)$
 $A[0].err := A[4].dup \text{ OR } A[4].err$
- 5) $A \rightarrow \epsilon$
 $A[0].err := \text{False}$

- Così la condizione 1 è soddisfatta per tutte le regole di produzione
- I Brother Graphs rimangono vuoti, quindi un qualsiasi ordinamento topologico va bene.

Grammatiche di Tipo L

- Una grammatica ad attributi è detta di **Tipo L** se è **One-Sweep**
e se ammette, per tutti i Brother Graphs,
l'ordinamento topologico $O(p) = \langle X_1, X_2, \dots, X_h \rangle$
- Le grammatiche di Tipo L possono essere valutate
durante il parsing a discesa ricorsiva, senza
materializzare l'albero.
- La grammatica ad attributi modificata per l'XML è di
Tipo L.

Brother Graphs

Esercizio

- La grammatica ad attributi per la definizione e uso delle variabili è **One-Sweep**?
- È di **Tipo L**?

- 1) LI
- 2) I 
- 4) VDEF
- 5) VUSE

Anche la condizione 1 è rispettata, quindi la grammatica è One-Sweep.
In più, è di Tipo L,

Che fare se ...

- ... la grammatica ad attributi NON È ONE-SWEEP?
- Se l'albero sintattico è materializzato, si può pensare di fare **passate multiple** sull'albero
- creando delle partizioni degli attributi, in modo da fare una passata per ogni partizione.
- Queste sono le grammatiche ad attributi **Multi-Sweep o Multi-Pass**.

Grammatiche Multi-Sweep

- Costruire il **Grafo Semplice** (Simple Graph) $S = (N, E)$ delle dipendenze tra gli attribuiti, indipendentemente dalla regola di produzione e dal Non-Terminale
Nodi: $N = \{a_1, a_2, \dots\}$ (attributi)
Archi: $E = \{b \rightarrow a, \dots\}$ se esiste una regola semantica in una regola di produzione p tale che $a \text{ of } X_i := f(\dots, b \text{ of } X_j p j, \dots)$.

Semantica dei Linguaggi

Grammatiche Multi-Sweep

- Identificare le **Componenti Connesse** C_1, C_2, \dots, C_n , dove $C_i = \{a_{i,1}, a_{i,2}, \dots\} \subseteq N$ e per ogni coppia C_i, C_j , deve essere $C_i \cap C_j = \emptyset$

Semantica dei Linguaggi

Grammatiche Multi-Sweep

- Creare il **Grafo Collassato** (Collapsed Graph) $CG = (\overline{N}, \overline{E})$
 $\overline{N} = \{\overline{C}_1, \overline{C}_2, \dots, \overline{C}_n\}$
 $\overline{E} = \{C_i \rightarrow C_j, \dots\}$
 se $b \rightarrow a \in E$ tale che $b \in C_i$ e $a \in C_j$.
- Calcolare l'ordinamento topologico $OC = \langle C_1, C_2, \dots \rangle$ del grafo collassato.
- Ogni componente connessa C_i deve rispettare le condizioni One-Sweep.
 La valutazione consiste di n visite dell'albero di tipo One-Sweep

Definizione Ritardata delle Variabili (late binding)

- Se il linguaggio ammette la possibilità di usare delle variabili (o, in generale, dei simboli) **Prima** che questi vengono definiti
- Il controllo che ogni variabile usata sia anche definita diventa più complesso
- Possibile approccio:
raccogliere prima tutte le definizioni di variabili, ovunque esse siano (verificando la doppia definizione)
Poi verificare l'uso delle variabili.

Attributi

- $Ere(PR) = \emptyset$
 $Sin(PR) = \{\text{err}\}$
- $Ere(LI) = \{\text{ev}, \text{fv}\}$
 $Sin(LI) = \{\text{derr}, \text{uerr}, \text{sv}\}$
- $Ere(I) = \{\text{ev}, \text{fv}\}$
 $Sin(I) = \{\text{derr}, \text{uerr}, \text{sv}\}$
- $Ere(VDEF) = \{\text{ev}\}$
 $Sin(VDEF) = \{\text{derr}, \text{sv}\}$
- $Ere(VUSE) = \{\text{fv}\}$
 $Sin(VUSE) = \{\text{uerr}\}$

Nuovi Attributi

- Aggiungiamo l'attributo **fv** (full variable list)
- distinguiamo due tipi di errori:
derr, errore di definizione
uerr, errore d'uso

- 1) $PR \rightarrow LI$
 $LI[1].ev := \emptyset$
 $PR[0].err := LI[1].derr \text{ OR } LI[1].uerr$
 $LI[1].fv = LI[1].sv$
- 2) $LI \rightarrow I$ LI
 $I[1].ev := LI[0].ev$
 $LI[2].ev := I[1].sv$
 $LI[0].derr := I[1].derr \text{ OR } LI[2].derr$
 $LI[0].uerr := I[1].uerr \text{ OR } LI[2].uerr$
 $LI[0].sv := LI[2].sv$
 $I[1].fv := LI[0].fv$
 $LI[2].fv := LI[0].fv$

- 3) $LI \rightarrow \epsilon$

```
LI[0].derr := False
LI[0].uerr := False
LI[0].sv := LI[0].ev
```

- 4) $I \rightarrow VDEF$

```
VDEF[1].ev := I[0].ev
I[0].sv := VDEF[1].sv
I[0].derr := VDEF[1].derr
I[0].uerr := False
```

- 5) $I \rightarrow VUSE$

```
VUSE[1].fv := I[0].fv
I[0].sv := I[0].ev
I[0].uerr := VUSE[1].uerr
I[0].derr := False
```

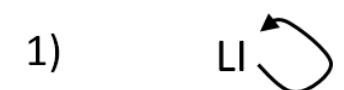
- 6) $VDEF \rightarrow Def\ Name$

```
VDEF[0].derr :=
CheckDef(VDEF[0].ev, Name[2].Value)
VDEF[0].sv :=
AddDef(VDEF[0].ev, Name[2].Value,
VDEF[0].derr)
```

- 7) $VUSE \rightarrow Name$

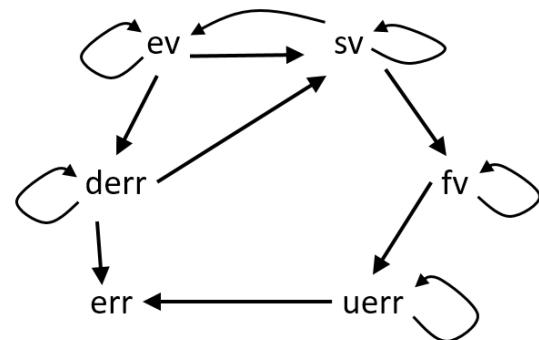
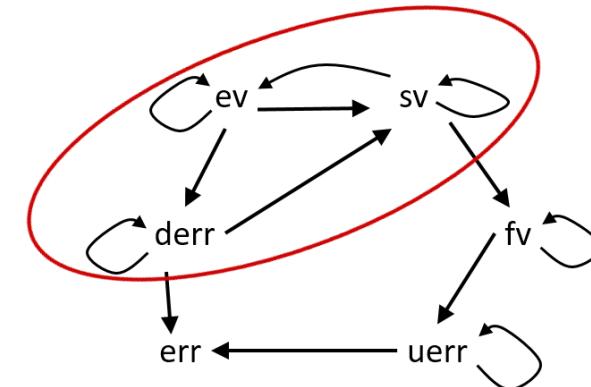
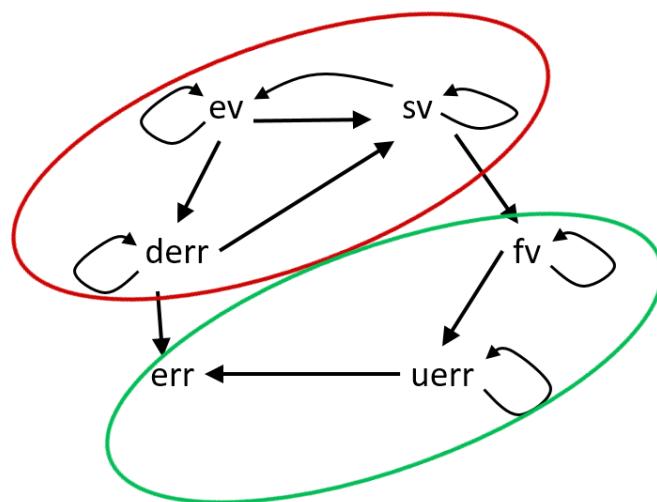
```
VUSE[0].uerr :=
CheckUse(VUSE[0].fv, Name[1].Value)
```

Brother Graphs Vi è un ciclo: non è One-Sweep



4)

5)

Simple Graph**Componenti Connesse****Semantica dei Linguaggi****Semantica dei Linguaggi****Componenti Connesse****Collapsed Graph**

C1 → C2

- $C_1 = \{ev, sv, derr\}$, prima passata (colore ROSSO)
- $C_2 = \{fv, uerr, err\}$, seconda passata (colore VERDE)

- 1) PR → LI

$LI[1].ev := \emptyset$

$PR[0].err := LI[1].derr \text{ OR } LI[1].uerr$

$LI[1].fv = LI[1].sv$

- 2) LI → I LI

$I[1].ev := LI[0].ev$

$LI[2].ev := I[1].sv$

$LI[0].derr := I[1].derr \text{ OR } LI[2].derr$

$LI[0].uerr := I[1].uerr \text{ OR } LI[2].uerr$

$LI[0].sv := LI[2].sv$

$I[1].fv := LI[0].fv$

$LI[2].fv := LI[0].fv$

- 3) LI → ϵ

$LI[0].derr := False$

$LI[0].uerr := False$

$LI[0].sv := LI[0].ev$

- 4) I → VDEF

$VDEF[1].ev := I[0].ev$

$I[0].sv := VDEF[1].sv$

$I[0].derr := VDEF[1].derr$

$I[0].uerr := False$

- 5) I → VUSE

$VUSE[1].fv := I[0].fv$

$I[0].sv := I[0].ev$

$I[0].uerr := VUSE[1].uerr$

$I[0].derr := False$

- 6) VDEF → Def Name

$VDEF[0].derr :=$

$\text{CheckDef}(VDEF[0].ev, \text{Name}[2].Value)$

$VDEF[0].sv :=$

$\text{AddDef}(VDEF[0].ev, \text{Name}[2].Value, VDEF[0].derr)$

- 7) VUSE → Name

$VUSE[0].uerr :=$

$\text{CheckUse}(VUSE[0].fv, \text{Name}[1].Value)$

- Le tecniche di valutazione semantica lavorano con casi particolari di grammatiche **Acicliche**, cioè che non presentano circolarità nel grafo delle dipendenze per nessun albero sintattico.
- Ma è possibile stabilire se una grammatica è **Aciclica** in modo generale?
- Donald Knuth, l'inventore delle grammatiche ad attributi, ha affrontato fin da subito il problema ma al primo tentativo ha sbagliato, l'algoritmo proposto non era esatto

- Esistono quindi due algoritmi (entrambi proposti da Knuth):
 - **Aciclicità Assoluta** (il primo)
 - **Acaciclicità Esatta** (quello corretto)

Aciclicità Assoluta

Passo 0

- Per ogni regola di produzione $p : A_0 \rightarrow A_1A_2 \dots A_n$ Dato il grafo delle dipendenze funzionali Dip_p , se ne calcola la sua **Chiusura Transitiva** (Dip_p)⁺
Si estrae $Dip_p^0(A_0)$, l'insieme degli archi η of $A_0 \rightarrow \sigma$ of A_0 ($\eta \in Ere(A_0)$, $\sigma \in Sin(A_0)$).
- Per ogni simbolo non-terminale $A \in V$, si calcola $D^0(A) = \cup_p Dip_p^0(A_0)$ per tutte le regole di produzione p con $A_0 = A$.

Aciclicità Assoluta

Passo Iterativo $i \geq 1$

- Per ogni regola di produzione $p : A_0 \rightarrow A_1A_2 \dots A_n$ e per ogni non-terminale figlio A_j (con $1 \leq j \leq n$) Si calcola $Dip'_p = Dip_p \cup_j D^{i-1}(A_j)$ e la sua chiusura transitiva $(Dip'_p)^+$ da cui si estrae $Dip_p^i(A_0)$.
- Per ogni non-terminale A , si calcola $D^i(A) = \cup_p Dip_p^i(A_0)$, per le regole p con $A = A_0$.
- Se succede che un $(Dip'_p)^+$ contiene cicli,
 \Rightarrow La grammatica non è aciclica.
- Se per almeno un non-terminale A succede che $D^i(A) \neq D^{i-1}(A)$, si ripete il passo iterativo altrimenti STOP (la grammatica è aciclica)

1) $S \rightarrow A \ B \ B$

$A[1].x := f1(S[0].x)$

$B[2].x := f2(A[1].y, B[3].y)$

$B[3].x := A[1].y$

$S[0].y := f1(B[2].y)$

2) $A \rightarrow a$

$A[0].y := f1(A[0].x)$

3) $B \rightarrow b$

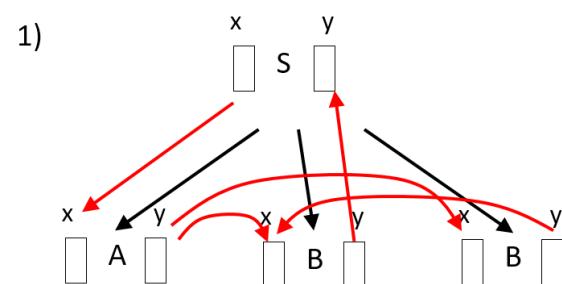
$B[0].y := 0$

4) $B \rightarrow S \ b$

$B[0].y := f2(B[0].x, S[1].y)$

$S[1].x := f1(B[0].x)$

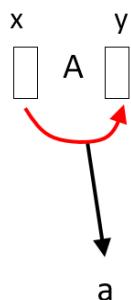
Passo 0



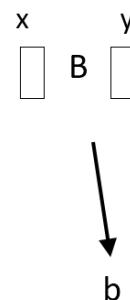
$$Dip_1^0(S) = \emptyset$$

Passo 0

2)

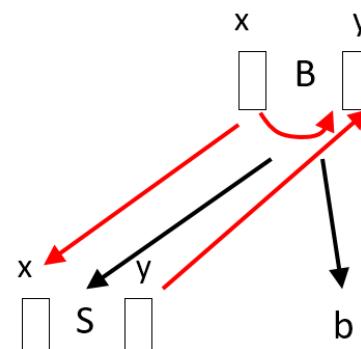


3)



Passo 0

4)



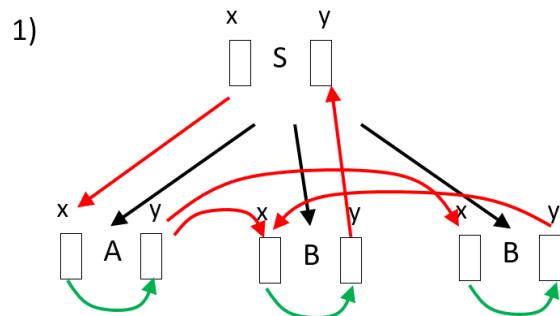
$$Dip_2^0(A) = \{x \rightarrow y\}$$

$$Dip_3^0(B) = \emptyset$$

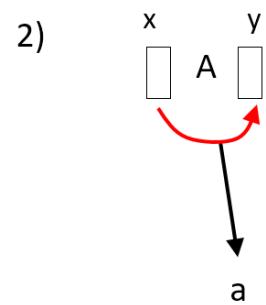
$$Dip_4^0(B) = \{x \rightarrow y\}$$

Passo 1**Passo 0**

- $D^0(S) = Dip_1^0(S) = \emptyset$
- $D^0(A) = Dip_2^0(A) = \{x \rightarrow y\}$
- $D^0(B) = Dip_3^0(B) \cup Dip_4^0(B) = \emptyset \cup \{x \rightarrow y\} = \{x \rightarrow y\}$

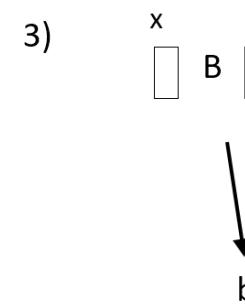
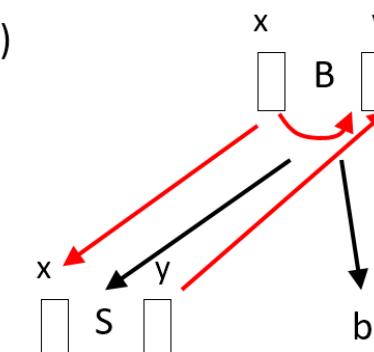


$$Dip_1^1(S) = \{x \rightarrow y\}$$

Passo 1

$$Dip_2^1(A) = \{x \rightarrow y\}$$

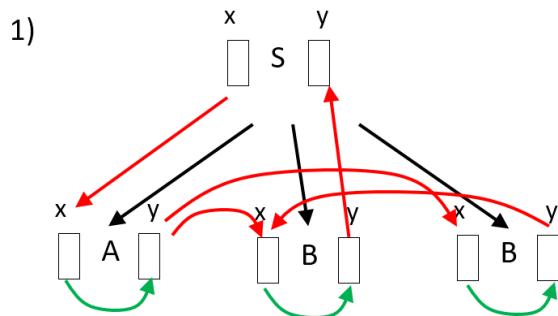
$$Dip_3^1(B) = \emptyset$$

**Passo 1**

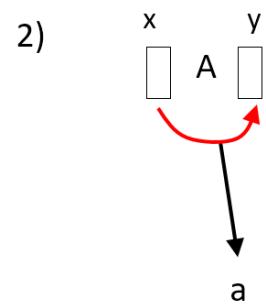
$$Dip_4^1(B) = \{x \rightarrow y\}$$

Passo 2**Passo 1**

- $D^1(S) = Dip_1^1(S) = \{x \rightarrow y\}$ Cambiamento
- $D^1(A) = Dip_2^1(A) = \{x \rightarrow y\}$
- $D^1(B) = Dip_3^1(B) \cup Dip_4^1(B) = \emptyset \cup \{x \rightarrow y\} = \{x \rightarrow y\}$

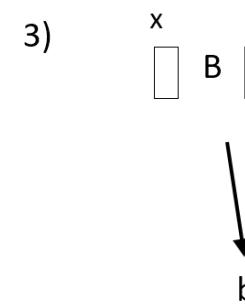
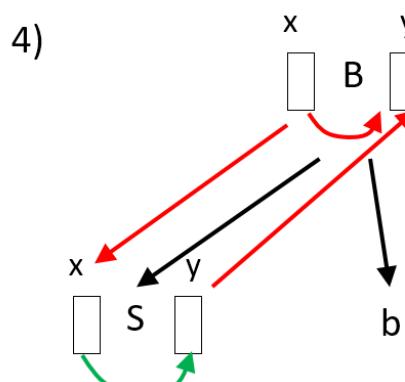


$$Dip_1^2(S) = \{x \rightarrow y\}$$

Passo 2

$$Dip_2^2(A) = \{x \rightarrow y\}$$

$$Dip_3^2(B) = \emptyset$$

**Passo 2**

$$Dip_4^2(B) = \{x \rightarrow y\}$$

Passo 2

- $D^2(S) = Dip_1^2(S) = \{x \rightarrow y\}$
- $D^2(A) = Dip_2^2(A) = \{x \rightarrow y\}$
- $D^2(B) = Dip_3^2(B) \cup Dip_4^2(B) = \emptyset \cup \{x \rightarrow y\} = \{x \rightarrow y\}$

Nessun ciclo e nessun cambiamento: la grammatica è aciclica

$$\textcircled{1} \quad S \rightarrow Z$$

$$Z[1].a := S[0].a$$

$$S[0].c := Z[1].c$$

$$\textcircled{2} \quad Z \rightarrow A \ Z$$

$$A[1].a := Z[0].a$$

$$Z[2].a := f2(Z[0].a, A[1].c)$$

$$Z[0].c := Z[2].c$$

$$\textcircled{3} \quad Z \rightarrow A$$

$$A[1].a := Z[0].a$$

$$Z[0].c := f2(Z[0].a, A[1].c)$$

$$\textcircled{4} \quad A \rightarrow B$$

$$B[1].a := f2(A[0].a, B[1].c)$$

$$B[1].b := B[1].d$$

$$A[0].c := f2(B[1].c, B[1].d)$$

$$\textcircled{5} \quad B \rightarrow 2$$

$$B[0].c := f1(B[1].b)$$

$$B[0].d := 0$$

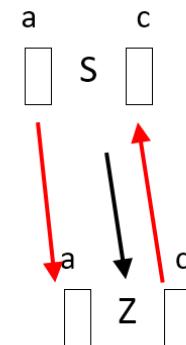
$$\textcircled{6} \quad B \rightarrow 3$$

$$B[0].c := 0$$

$$B[0].d := f1(B[1].a)$$

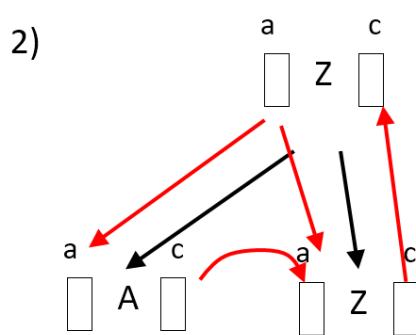
Passo 0

1)



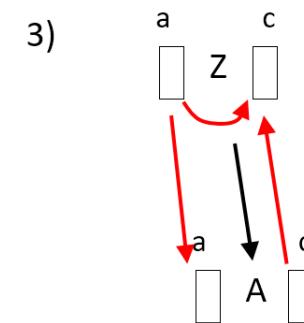
$$Dip_1^0(S) = \emptyset$$

Passo 0



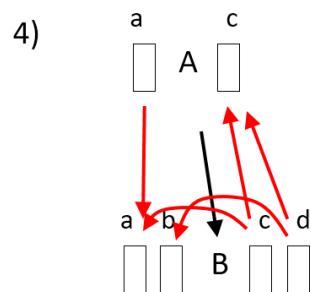
$$Dip_2^0(Z) = \emptyset$$

Passo 0



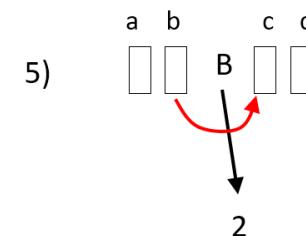
$$Dip_3^0(Z) = \{a \rightarrow c\}$$

Passo 0



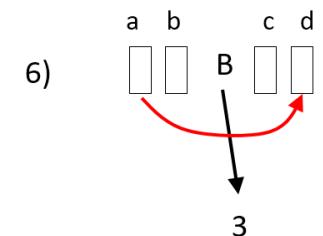
$$Dip_4^0(A) = \emptyset$$

Passo 0



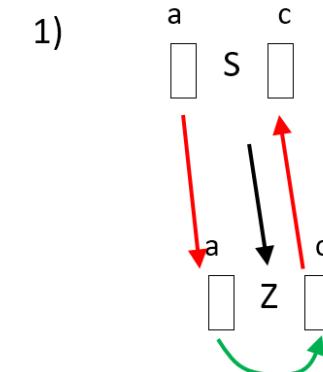
$$Dip_5^0(B) = \{b \rightarrow c\}$$

$$Dip_6^0(B) = \{a \rightarrow d\}$$

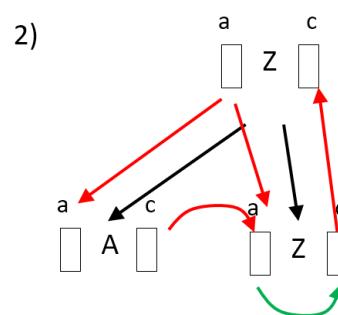


Passo 1**Passo 0**

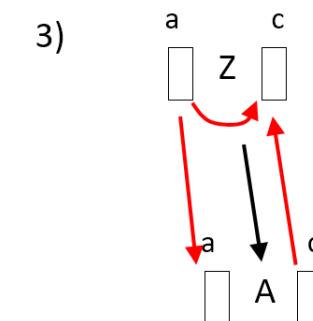
- $D^0(S) = Dip_1^0(S) = \emptyset$
- $D^0(Z) = Dip_2^0(Z) \cup Dip_3^0(Z) = \emptyset \cup \{a \rightarrow c\} = \{a \rightarrow c\}$
- $D^0(A) = Dip_4^0(A) = \emptyset$
- $D^0(B) = Dip_5^0(B) \cup Dip_6^0(B) = \{b \rightarrow c\} \cup \{a \rightarrow d\} = \{b \rightarrow c, a \rightarrow d\}$



$$Dip_1^1(S) = \{a \rightarrow c\}$$

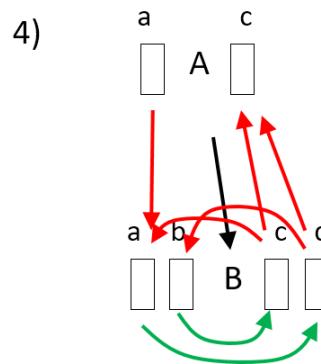
Passo 1

$$Dip_2^1(Z) = \{a \rightarrow c\}$$

Passo 1

$$Dip_3^1(Z) = \{a \rightarrow c\}$$

Passo 1



$$Dip_4^1(A) = \{a \rightarrow c\}$$

ATTENZIONE !!! CICLO

- La grammatica non è **Assolutamente Aciclica**
- Ma la circolarità è data dalla fusione delle dipendenze relative al non-terminale B
- Tuttavia, queste dipendenze non possono mai essere presenti insieme
- Knuth ha quindi modificato l'algoritmo, proponendo l'algoritmo esatto e chiamando il precedente Aciclicità Assoluta.

Aciclicità Esatta

Passo 0

- Per ogni regola di produzione $p : A_0 \rightarrow A_1A_2 \dots A_n$ Dato il grafo delle dipendenze funzionali Dip_p , se ne calcola la sua **Chiusura Transitiva** $(Dip_p)^+$ Si estrae $Dip_p^0(A_0)$, l'insieme degli archi η of $A_0 \rightarrow \sigma$ of A_0 ($\eta \in Ere(A_0)$, $\sigma \in Sin(A_0)$).
- Per ogni simbolo non-terminale $A \in V$, si calcola $DD^0(A) = \cup_p \{Dip_p^0(A_0)\}$ per tutte le regole di produzione p con $A_0 = A$ ($DD^0(A)$ è un **Insieme di Grafi**).

Aciclicità Esatta

Passo Iterativo $i \geq 1$

- Per ogni regola di produzione $p : A_0 \rightarrow A_1A_2 \dots A_n$ e per ogni combinazione di grafi $c = \langle D(A_1), \dots, D(A_n) \rangle$ con $D(A_j) \in DD^{i-1}(A_j)$ Si calcola $Dip_{p,c}' = Dip_p \cup_j D(A_j)$ e la sua chiusura transitiva $(Dip_{p,c}')^+$ da cui si estrae $Dip_{p,c}^i(A_0)$ l'insieme degli archi η of $A_0 \rightarrow \sigma$ of A_0 ($\eta \in Ere(A_0)$, $\sigma \in Sin(A_0)$).
- Per ogni non-terminale A , si calcola $DD^i(A) = DD^{i-1}(A) \cup_{p,c} \{Dip_{p,c}^i(A)\}$.

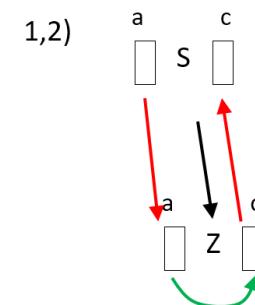
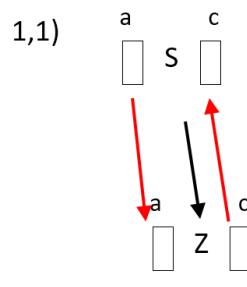
- Se succede che un $(Dip'_{p,c})^+$ contiene cicli,
⇒ La grammatica non è aciclica.
- Se per almeno un non-terminale A succede che
 $DD^i(A) \neq DD^{i-1}(A)$, si ripete il passo iterativo
altrimenti STOP (la grammatica è aciclica)

Riprendiamo la grammatica che non è assolutamente aciclica

Passo 0

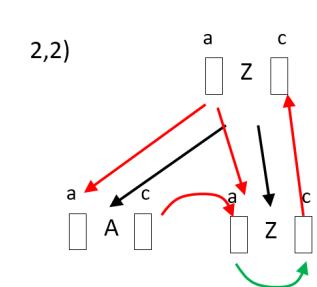
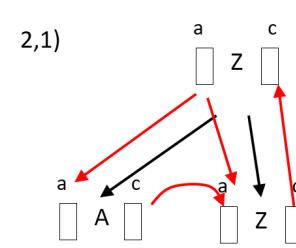
- $DD^0(S) = \{Dip_1^0(S)\} = \{\emptyset\}$
- $DD^0(Z) = \{Dip_2^0(Z), Dip_3^0(Z)\} = \{\emptyset, \{a \rightarrow c\}\}$
- $DD^0(A) = \{Dip_4^0(A)\} = \{\emptyset\}$
- $DD^0(B) = \{Dip_5^0(B), Dip_6^0(B)\} = \{\{b \rightarrow c\}, \{a \rightarrow d\}\}$

Passo 1

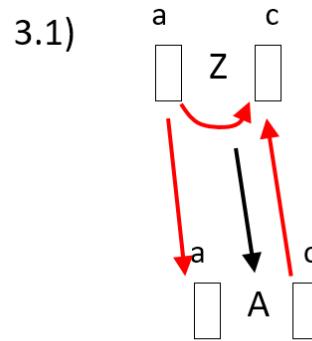


$$\begin{aligned}Dip_{1,1}^1(S) &= \emptyset \\Dip_{1,2}^1(S) &= \{a \rightarrow c\}\end{aligned}$$

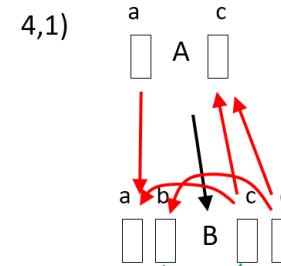
Passo 1



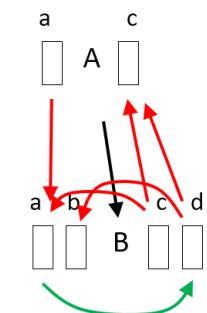
$$\begin{aligned}Dip_{2,1}^1(Z) &= \emptyset \\Dip_{2,2}^1(Z) &= \{a \rightarrow c\}\end{aligned}$$

Passo 1

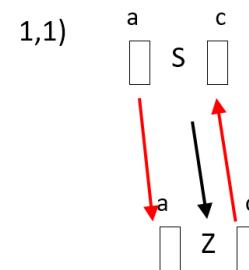
$$Dip_{3,1}^1(Z) = \{a \rightarrow c\}$$

Passo 1

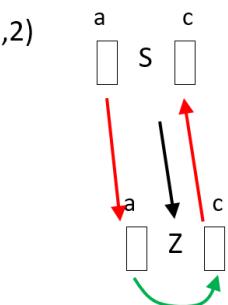
$$\begin{aligned} Dip_{4,1}^1(A) &= \emptyset \\ Dip_{4,2}^1(A) &= \{a \rightarrow c\} \end{aligned}$$

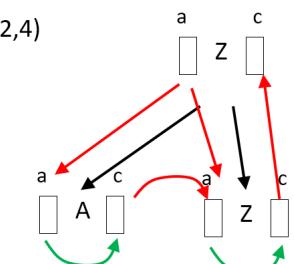
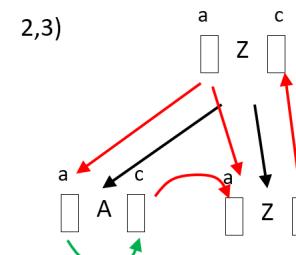
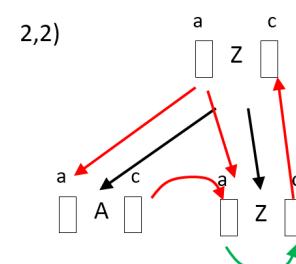
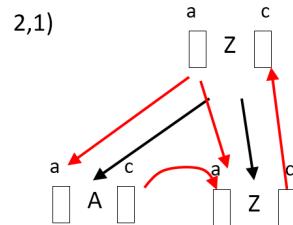
**Passo 1**

- $DD^1(S) = DD^0(S) \cup \{Dip_{1,1}^1(S), Dip_{1,2}^1(S)\} = \{\emptyset\} \cup \{\emptyset, \{a \rightarrow c\}\} = \{\emptyset, \{a \rightarrow c\}\}$ cambiato
- $DD^1(Z) = DD^0(Z) \cup \{Dip_{2,1}^1(Z), Dip_{2,2}^1(Z), Dip_{3,1}^1(Z)\} = \{\emptyset, \{a \rightarrow c\}\} \cup \{\emptyset, \{a \rightarrow c\}, \{a \rightarrow c\}\} = \{\emptyset, \{a \rightarrow c\}\}$
- $DD^1(A) = DD^0(A) \cup \{Dip_{4,1}^1(A), Dip_{4,2}^1(A)\} = \{\emptyset\} \cup \{\emptyset, \{a \rightarrow c\}\} = \{\emptyset, \{a \rightarrow c\}\}$ cambiato
- $DD^1(B) = DD^0(B) \cup \{Dip_{5,1}^1(B), Dip_{6,1}^1(B)\} = \{\{b \rightarrow c\}, \{a \rightarrow d\}\} \cup \{\{b \rightarrow c\}, \{a \rightarrow d\}\} = \{\{b \rightarrow c\}, \{a \rightarrow d\}\}$

Passo 2

$$\begin{aligned} Dip_{1,1}^2(S) &= \emptyset \\ Dip_{1,2}^2(S) &= \{a \rightarrow c\} \end{aligned}$$



Passo 2

$$Dip_{2,1}^2(Z) = \emptyset$$

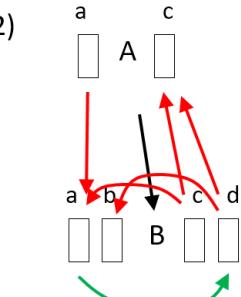
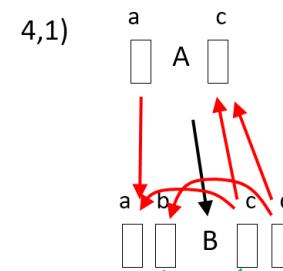
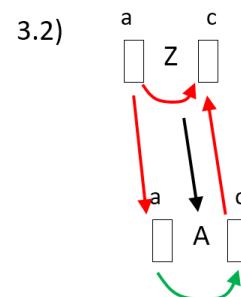
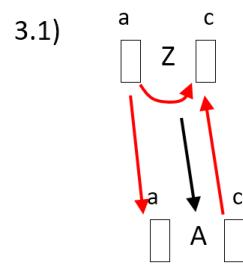
$$Dip_{2,2}^2(Z) = \{a \rightarrow c\}$$

$$Dip_{2,3}^2(Z) = \emptyset$$

$$Dip_{2,4}^2(Z) = \{a \rightarrow c\}$$

Acicità Esatta

Acicità Esatta

Passo 2

$$Dip_{3,1}^2(Z) = \{a \rightarrow c\}$$

$$Dip_{3,2}^2(Z) = \{a \rightarrow c\}$$

$$Dip_{4,1}^2(A) = \emptyset$$

$$Dip_{4,2}^2(A) = \{a \rightarrow c\}$$

Passo 2

- $DD^2(S) = DD^1(S) \cup \{Dip_{1,1}^2(S), Dip_{1,2}^2(S)\} = \\ = \{\emptyset, \{a \rightarrow c\}\} \cup \{\emptyset, \{a \rightarrow c\}\} = \{\emptyset, \{a \rightarrow c\}\}$
- $DD^2(Z) = DD^1(Z) \cup \{Dip_{2,1}^2(Z), Dip_{2,2}^2(Z), Dip_{2,3}^2(Z), \\ Dip_{2,4}^2(Z), Dip_{3,1}^2(Z), Dip_{3,2}^2(Z)\} = \\ = \{\emptyset, \{a \rightarrow c\}\} \cup \{\emptyset, \{a \rightarrow c\}, \emptyset, \{a \rightarrow c\}, \\ \{a \rightarrow c\}, \{a \rightarrow c\}\} = \{\emptyset, \{a \rightarrow c\}\}$
- $DD^2(A) = DD^1(A) \cup \{Dip_{4,1}^2(A), Dip_{4,2}^2(A)\} = \\ = \{\emptyset, \{a \rightarrow c\}\} \cup \{\emptyset, \{a \rightarrow c\}\} = \{\emptyset, \{a \rightarrow c\}\}$
- $DD^2(B) = DD^1(B) \cup \{Dip_{5,1}^2(B), Dip_{6,1}^2(B)\} = \\ = \{\{b \rightarrow c\}, \{a \rightarrow d\}\} \cup \{\{b \rightarrow c\}, \{a \rightarrow d\}\} = \\ = \{\{b \rightarrow c\}, \{a \rightarrow d\}\}$

Passo 2

- Nessun ciclo
- Nessun cambiamento, quindi la grammatica è aciclica in modo esatto (ma non era assolutamente aciclica)
- Prezzo da pagare: l'algoritmo esatto è esponenziale