Compilatori

Silviu Filote

July 2023

Contents

1 Introduzione

- Un **traduttore** è un componente software che legge un testo scritto in uno specifico **linguaggio formale** e lo traduce: in un altro linguaggio formale o in una serie di azioni (operazioni).
- Un **compilatore** è un traduttore che traduce un linguaggio di programmazione nella corrispondente versione binaria. É suddiviso in:
  - **Front-end:** la parte del compilatore che analizza il testo
  - **Back-end:** la parte che genera il codice binario della specifica piattaforma
- Un **interprete** è un traduttore che traduce un programma scritto in un linguaggio di programmazione in azioni (esegue direttamente il programma)

1.1 Differenze struttura sintattica e semantica di un linguaggio

- La **struttura sintattica** si occupa delle regole che governano la formazione delle frasi e delle proposizioni in modo che siano grammaticalmente corrette. $\Rightarrow$ Si occupa dunque della corretta costruzione delle frasi in una lingua.
- La **struttura semantica** riguarda il significato delle parole, delle frasi e dei testi. Si occupa di come le parole e le unità linguistiche connesse si combinano per formare un significato coerente $\Rightarrow$ riguarda l'interpretazione e il significato del linguaggio.

1.2 Struttura del Front-end (Compilatore)

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

1.3 Focus del Corso

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:** 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

2 Aspetti Formali

Alfabeto (insieme finito di caratteri terminali)	$\Sigma = \{a, b\}$
Stringhe (seq. caratteri terminali)	$aa, abab, b, baa$
Stringa vuota	$\varepsilon, \lvert\varepsilon\rvert = 0$
Linguaggio (insieme di stringhe)	$L = \{aa, aaa, aba, aab\}$
Cardinalitá	$\lvert\Sigma\rvert = 2,\ \lvert L\rvert = 4$
Linguaggio vuoto	$L = \emptyset,\ \lvert\emptyset\rvert = 0$
Casi particolari	$\emptyset^0 = \{\varepsilon\},\quad \lvert\{\varepsilon\}\rvert = 1,\quad \lvert\emptyset\rvert = 0$

2.1 Operazioni tra stringhe

- Due stringhe: $x = a_1 \dots a_h \qquad y = b_1 \dots b_k$
- La stringa $x = uyz$, dove u, y e z sono sottostringhe, dove $\boldsymbol{u}$ é il **prefisso** e $\boldsymbol{y}$ é il **suffisso** di x

Concatenamento	$x \bullet y = xy = a_1 \dots a_h b_1 \dots b_k$
Prefissi	***k : x** si indica il prefisso di **x** avente lunghezza **k***
Riflessione	$x^R = a_h \dots a_1$ *(reverse caratteri)*
Ripetizioni - potenze	$x^n = x^{n-1} \bullet x,\ x^0 = \varepsilon$

Osservazione: l'elevamento a potenza e la riflessione hanno precedenza rispetto al concatenamento.

2.2 Operazioni sui Linguaggi

Definizione: dato un alfabeto Σ e un linguaggio $L = \Sigma$ (linguaggio dei simboli terminali), Σ^* ***(monoide libero)*** contiene tutte le stringhe che possono essere costruite concatenando i caratteri terminali. Ogni linguaggio formale L di alfabeto Σ é incluso in Σ^*, $L \subseteq \Sigma^*$.

$$\emptyset^* = \{\varepsilon\}, \quad \{\varepsilon\}^* = \{\varepsilon\}$$

Concatenamento di L_1 e L_2	$L = L_1 \bullet L_2 = \{xy \mid \forall x \in L_1\ AND\ \forall y \in L_2\}$
*Linguaggio degli Inizi di Lunghezza **k** di **L***	$k : L = \{y \mid x \in L\ AND\ y = k : x\}$
*Linguaggio dei Prefissi (Propri) di **L***	$Prefissi(L) = \{y \mid x = yz\ AND\ x \in L\ AND\ z \neq \varepsilon\}$
*Linguaggio **L** privo di prefissi*	$Prefissi(L) \cap L = \emptyset$
*Riflessione di **L***	$L^R = \{x \mid y \in L\ AND\ x = y^R\}$
Ripetizione - potenze, $n > 0$	$L^n = L^{n-1} \bullet L,\quad L^0 = \{\varepsilon\} \neq \emptyset,\quad L \bullet \{\varepsilon\} = L,\quad L \bullet \emptyset = \emptyset$
Operatore stella	$L^* = \cup_{h=0\dots\infty} L^h = \{\varepsilon\} \cup L^1 \cup L^2 \cup \dots$
Operatore croce	$L^+ = \cup_{h=1\dots\infty} L^h = L^1 \cup L^2 \cup \dots$
Croce: casi Particolari	$L^* = L^+ \cup \{\varepsilon\},\quad L^+ = L \bullet L^* = L^* \bullet L$

2.3 Insiemistica

Unione	$L_1 \cup L_2$
Intersezione	$L_1 \cap L_2$
Differenza	$L_1 - L_2$
Inclusione	$L_1 \subseteq L_2$
Inclusione propria	$L_1 \subset L_2$
Uguaglianza	$L_1 = L_2$
Complemeto dato Σ	$\neg L = \Sigma^* - L = \{x \mid x \in \Sigma^*\ AND\ x \notin L\}$

Sostituzione: si considerino due alfabeti Σ e Δ e i linguaggi $L \subseteq \Sigma^*$ ***(linguaggio sorgente)*** e $L^{'} \subseteq \Delta^*$ ***(linguaggio pozzo)***. La sostituzione di $L^{'}$ al posto di un carattere $b \in \Sigma$ nella stringa $x = a_1 \dots a_h$ produce il linguaggio di alfabeto $(\Sigma - \{b\}) \cup \Delta$, cosí definita:

$$x = a_1 \dots a_n \qquad y = c_1 \dots c_n \qquad 1 \leq i \leq n$$

$$\phi_{b \to L'}(x) = \{y \mid x \in L \textit{ AND } \forall (\textit{IF } a_i \neq b \textit{ THEN } c_i = a_i \textit{ ELSE } c_i \in L^{'})\}$$

"Data la stringa $\boldsymbol{x = yb}$ sostituisco la b con ogni elemento in L'"

$$L^1 = L \bullet L^0 = L^1 \bullet \{\varepsilon\} = L$$

$$L^* = \{\varepsilon\} \cup L^1 \cup L^2 \cup L^\infty$$

$$L^+ = L^1 \cup L^2 \cup L^\infty$$

$$\textit{Se } \{\varepsilon\} \in L \;\; \Rightarrow \;\; L^+ = L \bullet L^* = L^* \bullet L = L^*$$

$$\textit{Se } \{\varepsilon\} \notin L \;\; \Rightarrow \;\; L^+ = L \bullet L^* = L^* \bullet L$$

3 Espressioni e Linguaggi Regolari

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***.

Definizione: Un linguaggio di alfabeto $\Sigma = \{a_1, \ldots, a_n\}$ é detto ***regolare*** se può essere espresso mediante le operazioni di *concatenamento, unione* e *stella* applicate un numero ***finito di volte*** ai linguaggi unitari $\{a_1\}, \ldots, \{a_n\}$ e al linguaggio vuoto $\emptyset$.

Definizione: un ***espressione regolare (er) r é 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

precedenza operatori: $*, \bullet, \cup$

$r = \emptyset$	$r = a \quad dove\ a \in \Sigma$
$r = (s \cup t) \quad dove\ s\ e\ t\ sono\ e.r.$	$r = (s \bullet t) \quad dove\ s\ e\ t\ sono\ e.r.$
$r = (s)^* \quad dove\ s\ e\ t\ sono\ e.r.$	$r = \varepsilon, \quad \varepsilon = \emptyset^*$

Notazioni Comode

$e^k = e \ldots e \qquad k \geq 0\ volte$	$e^+ = ee^*$
$[e]_k^h = e^k \cup e^{k+1} \cup \ldots \cup e^h \quad con\ 0 \leq k \leq h$	$[e] = [e]_0^1 = \varepsilon \cup e \ \ opzionalita$

3.1 Sotto-Espressione (S.E.)

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

- $\boldsymbol{e_k}$ é una S.E. di $(e_1 \cup e_2 \cup \ldots \cup \boldsymbol{e_k} \cup \ldots \cup e_j)$
- $\boldsymbol{e_k}$ é una S.E. di $(e_1 \bullet e_2 \bullet \ldots \bullet \boldsymbol{e_k} \bullet \ldots \bullet e_j)$
- $\boldsymbol{e}$ é una S.E. di $\boldsymbol{e}^*, \boldsymbol{e}^+$ e di $\boldsymbol{e}^k$
- $\boldsymbol{\varepsilon}$ é una S.E. di ogni espressione regolare

3.2 Relazione di Implicazione $\Rightarrow$

$e^{'} \Rightarrow e^{''}$ ($e^{'}$ implica $e^{''}$) se le due espressioni si possono ***fattorizzare*** come:

$$e^{'} = \alpha\beta\gamma \qquad e \qquad e^{''} = \alpha\delta\gamma$$

$$dove\ \alpha, \beta, \gamma\ sono\ S.E.\ di\ e^{'}$$

e tra β e δ vale una delle seguenti relazioni:

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

$e_0 \overset{n}{\Rightarrow} e_n$	*n passi implicazione*
$e_0 \overset{*}{\Rightarrow} e_n$	$n \geq 0$ *passi implicazione*
$e_0 \overset{+}{\Rightarrow} e_n$	$n > 0$ *passi implicazione*

NB: si ha **un'implicazione sinistra** se α è il più lungo prefisso comune ad $e^{'}$ e $e^{''}$ privo di meta simboli.

Osservazioni:

- Linguaggio generato da un'espressione regolare $\boldsymbol{r}$: $\quad L(r) = \{x \in \Sigma^* \mid r \overset{*}{\Rightarrow} x\}$
  con x priva di meta-simboli
- Un linguaggio é detto ***regolare*** se é generato da espressione regolare (e.r.)
- Due espressioni regolari sono ***equivalenti*** se generano lo stesso linguaggio

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: implicazione sinistra $\alpha = \varepsilon$

$$\alpha\beta\gamma \Rightarrow \alpha\delta\gamma$$
$$\varepsilon(a^* \cup b^+)\varepsilon \Rightarrow \varepsilon(a^*)\varepsilon \Rightarrow \varepsilon\varepsilon\varepsilon$$

$$(a^* \cup b^+) \Rightarrow a^* \Rightarrow \varepsilon$$
$$(a^* \cup b^+) \overset{2}{\Rightarrow} \varepsilon$$

3.3 Automa a stati finiti deterministico

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

Q	*Insieme degli stati*
Σ	*Alfabeto di ingresso*
δ	$\delta : Q \times \Sigma \rightarrow Q$ *funzione di transizione*
q_0	$q_0 \in Q$ *stato inziale*
F	$F \subseteq Q$ *stati finali*

Funzione di transizione transitiva:

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

Una stringa $x \in L(r)$, dove $\boldsymbol{r}$ é un'espressione regolare riconosciuta dall'automa se:

$$\delta^*(q_0, x) = q \quad con \quad q \in F$$

3.4 Automa a stati finiti non deterministico senza $\varepsilon-$mosse

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

Q	*Insieme degli stati*
Σ	*Alfabeto di ingresso*
δ	$\delta : Q \times \Sigma \rightarrow (2^Q - \{\emptyset\})$ *funzione di transizione* ***non deterministica***
q_0	$q_0 \in Q$ *stato inziale*
F	$F \subseteq Q$ *stati finali*

Insieme delle parti: dato un insieme Q, l'insieme delle sue parti 2^Q é l'insieme di tutti i suoi sottoinsiemi. Si noti che $|2^Q| = 2^{|Q|}$

Funzione di transizione transitiva non deterministica senza $\varepsilon-$mosse:

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

Una stringa $x \in L(r)$, dove $\boldsymbol{r}$ é un'espressione regolare riconosciuta dall'automa se:

$$\exists q \in F \quad tale\ che \quad q \in \delta^*(q_0, x)$$

Osservazioni:

- La funzione di transizione δ é definita per un insieme di coppie (Q, Σ) e restituisce in uscita un nuovo stato. **Mentre** la funzione di transizione transitiva δ^* ci dice come cambia lo stato dell'automa quando leggi un simbolo dall'input e fai delle modifiche alla pila, basandoti su ciò che hai già calcolato per lo stato precedente e l'input precedente.
- Quando ci spostiamo da uno stato ad un altro per messo di una transazione consumiamo il carattere associata alla transazione della stringa passata come input.
- Data una stringa di input, se arriviamo allo stato finale con una stringa vuota abbiamo una **esecuzione corretta**, altrimenti errore
- **Automa a stati finiti non deterministico senza $\varepsilon-$mosse:** il non determinismo é dato dal avere piú transizioni con lo stesso simbolo che escono dallo stesso stato
- Una ε-mossa è una transizione diretta da uno stato all'altro etichettata con ε. Viene detta **mossa spontanea**, perchè la transizione avviene consumando il carattere ε dal dispositivo di ingresso, **cioè senza consumare alcun carattere (spontaneamente).** In opposizione, le altre mosse vengono dette **non spontanee**.

3.5 Automa a stati finiti non deterministico con $\varepsilon-$mosse

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

Q	*Insieme degli stati*
Σ	*Alfabeto di ingresso*
δ	$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to (2^Q - \{\emptyset\})$ *funzione di transizione* ***non deterministica***
q_0	$q_0 \in Q$ *stato inziale*
F	$F \subseteq Q$ *stati finali*

Funzione di transizione transitiva non deterministica con $\varepsilon-$mosse:

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

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

Una stringa $x \in L(r)$, dove $\boldsymbol{r}$ é un'espressione regolare riconosciuta dall'automa se:

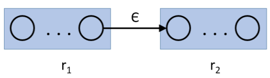
$$\exists q \in F \quad tale \ che \quad q \in \delta^*(q_0, x)$$

3.6 Utilità delle ε-mosse:

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

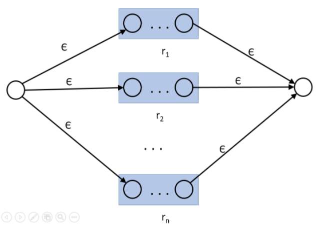
Composizione: Concatenamento

$$r = r_1 \bullet r_2$$



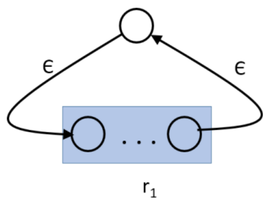
Composizione: Alternativa

$$r = (r_1 \cup r_2 \cup \ldots \cup r_n)$$



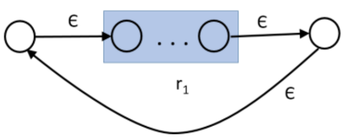
Composizione: Stella

$$r = (r_1)^*$$



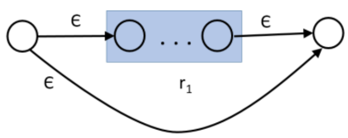
Composizione: Croce

$$r = (r_1)^+$$



Composizione: Opzionalità

$$r = [r_1]$$



3.7 Algoritmo Complemento

Dato un automa 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.

- Sia $M(Q, \Sigma, \delta, q_0, F)$ l'automa deterministico per L
- Sia $\overline{M}(\overline{Q}, \Sigma, \overline{\delta}, q_0, \overline{F})$ l'automa per $\neg L$ da calcolare

Passaggi:

1. Aggiungere agli stati Q uno **stato pozzo** p, dunque $\overline{Q} = Q \cup \{p\}$
2. $\forall q \in Q$ e $\forall a \in \Sigma$, $\quad \overline{\delta}(q, a) = \delta(q, a)$ se $\delta(q, a)$ é definita, altrimenti $\overline{\delta}(q, a) = p$
3. $\forall a \in \Sigma$, $\quad \overline{\delta}(p, a) = p$
4. gli stati finali sono dunque $\overline{F} = (Q - F) \cup \{p\}$

3.8 Algoritmo per Eliminare le ε-mosse

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

1. Inserire lo stato q_0 in Q' e in N, dove N é insieme dei nuovi stati
2. Impostiamo l'insieme $N' = \emptyset$. $\forall q \in N$, copiare da δ tutte le mosse non spontanee che escono da q, cioé $\delta(q, a) = \overline{q}$, creando $\delta'(q, a) = \overline{q}$ aggiungendo $\overline{q}$ in Q' e in N', se non è già presente in Q'.
3. $\forall q \in N$, cercare tutte le transizioni transitive $q \overset{+}{\Rightarrow} \overline{q}$ che contengono una sola mossa non spontanea preceduta e/o seguita da mosse spontanee, cioé:

   $$q \overset{\varepsilon}{\Rightarrow} \ldots \overset{\varepsilon}{\Rightarrow} q_i \overset{a}{\Rightarrow} \overline{q},$$
   $$q \overset{a}{\Rightarrow} \ldots \overset{\varepsilon}{\Rightarrow} q_i \overset{\varepsilon}{\Rightarrow} \overline{q},$$
   $$q \overset{\varepsilon}{\Rightarrow} \ldots \overset{a}{\Rightarrow} q_i \overset{\varepsilon}{\Rightarrow} \overline{q}$$

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

   Attenzione: gli stati $\overline{q}$ da considerare per i pattern sono tutti e solo gli stati $\overline{q}$ 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, perchè introdurrebbero inutile ridondanza).
4. $\forall q \in N$ se esiste un percorso che va da q ad uno stato $\overline{q}$ composto solo di ε-mosse, con $\overline{q} \in F$, inserire q in F' (stato finale di M').
5. Se $N' \neq \emptyset$, $N = N'$ e ripartire dal Passo 2, altrimenti terminare

3.9 Algoritmo per Eliminare il Non-Determinismo

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

1. Inserire lo stato q_0 in Q' e in N, dove N é insieme dei nuovi stati
2. Impostiamo l'insieme $N' = \emptyset$. $\forall q \in N$ che compare anche in Q, $\forall a \in \Sigma$ per cui $\delta(q, a) = \{q_1, q_2, \ldots, q_n\}$ non vuoto
   - se $n > 1$, creare uno stato **collettivo** $[q_1, q_2, \ldots, q_n]$ e aggiungere la mossa $\delta'(q, a) = [q_1, q_2, \ldots, q_n]$. Se $[q_1, q_2, \ldots, q_n]$ non é giá 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 é giá in Q'.
3. $\forall [q_1, q_2, \ldots, q_n] \in N$, $\forall q_i \in [q_1, q_2, \ldots, q_n]$ e $\forall a \in \Sigma$ calcolare il nuovo stato collettivo $[\delta(q_1, a) \cup \delta(q_2, a) \cup \ldots \cup \delta(q_n, a)]$
   - aggiungere questo stato in Q' e in N' se non é giá in Q'.
   - se il nuovo stato collettivo contiene un solo stato, diventa uno stato semplice.
   - definire $\delta'([q_1, q_2, \ldots, q_n], a) = [\delta(q_1, a) \cup \delta(q_2, a) \cup \ldots \cup \delta(q_n, a)]$

4. $\forall q = [q_1, q_2, \ldots, q_n] \in N$ (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', cioé $q \in F'$

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

3.10 Contenuto capitolo

- Le **espressioni regolari** definite con operatori $\{\cup, \bullet, ^*\}$ dispongono di un meccanismo efficiente di riconoscimento: gli **automi a stati finiti**
- **Definizione:** linguaggio regolare
- **Definizione:** espressione regolare $\boldsymbol{r}$ é una stringa composta da: $\Sigma \cup \{\bullet, \cup, ^*, \emptyset\} \cup \{(,)\}$
- **Definzione:** Sotto-Espressione (S.E.) di un'espressione regolare
- **Definzione:** Relazione di implicazione viene definita come $e' = \alpha\beta\gamma \Rightarrow e'' = \alpha\delta\gamma$
- Precedenza degli operatori: $^*, \bullet, \cup$
- $\cup$: rappresenta l'alternativa/opzionalitá
- **Definzione:** implicazione sinistra $e' \Rightarrow e''$, se α è il pù lungo prefisso comune ad e' ed e'' privo di meta-simboli
- **Definizione:** automa a stati finiti deterministico
- **Definizione:** Automa a stati finiti non deterministico senza $\varepsilon-$mosse
- **Definizione:** Automa a stati finiti non deterministico con $\varepsilon-$mosse
- Importanza $\varepsilon-$mosse: vengono usati per raccordare automi
- **Algoritmo Complemento:** si parte da un automa deterministimo M
- **Algoritmo per Eliminare le $\varepsilon-$mosse**
- **Algoritmo per Eliminare il Non-Determinismo**

4 Esercizio 1

4.1 Automa a stati finiti deterministico

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

Q	*Insieme degli stati*
Σ	*Alfabeto di ingresso*
δ	$\delta : Q \times \Sigma \to Q$ *funzione di transizione*
q_0	$q_0 \in Q$ *stato inziale*
F	$F \subseteq Q$ *stati finali*

Funzione di transizione transitiva:

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

$$con \quad \delta^*(q, \varepsilon) = q$$

Una stringa $x \in L(r)$, dove $\boldsymbol{r}$ é un'espressione regolare riconosciuta dall'automa se:

$$\delta^*(q_0, x) = q \quad con \quad q \in F$$

Osservazione: da ogni stato non esistono carateri che si ripetono ossia sono unici e dunque data una funzione di transizione so sempre in che stato finisco.

$a \cup b$	*alternativa o opzionalitá: scelgo a o b*
$[e] \equiv \varepsilon \cup e$	*alternativa o opzionalitá: scelgo* ε *o b*
$\delta : Q \times \Sigma \to Q$	$\delta(\underset{Q}{x}, \underset{\Sigma}{y}) \to \underset{Q}{z}$

4.2 Automa a stati finiti non deterministico senza $\varepsilon-$mosse

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

Q	*Insieme degli stati*
Σ	*Alfabeto di ingresso*
δ	$\delta : Q \times \Sigma \to (2^Q - \{\emptyset\})$ *funzione di transizione* ***non deterministica***
q_0	$q_0 \in Q$ *stato inziale*
F	$F \subseteq Q$ *stati finali*

Funzione di transizione transitiva non deterministica senza $\varepsilon-$mosse:

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

$$con \quad \delta^*(q, \varepsilon) = \{q\}$$

Una stringa $x \in L(r)$, dove $\boldsymbol{r}$ é un'espressione regolare riconosciuta dall'automa se:

$$\exists q \in F \quad tale\ che \quad q \in \delta^*(q_0, x)$$

Insieme delle parti: dato un insieme Q, l'insieme delle sue parti 2^Q é l'insieme di tutti i suoi sottoinsiemi. Si noti che $|2^Q| = 2^{|Q|}$

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

$$\delta : Q \times \Sigma \to (2^Q - \{\emptyset\})$$

$$\delta(x, y) \to \{z_1, z_2, z_3\}$$

Osservazione: il non determinismo é dato dal fatto che con la funzione di transizione posso raggiungere piú stati, dunque esiste in uscita da uno stato lo stessa carattere piú volte

4.3 Automa a stati finiti non deterministico con $\varepsilon-$mosse

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

Q	*Insieme degli stati*
Σ	*Alfabeto di ingresso*
δ	$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow (2^Q - \{\emptyset\})$ *funzione di transizione* ***non deterministica***
q_0	$q_0 \in Q$ *stato inziale*
F	$F \subseteq Q$ *stati finali*

Funzione di transizione transitiva non deterministica con $\varepsilon-$mosse:

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

$$con \; \delta^*(q, \varepsilon) = \{q\} \; se \; \delta(q, \varepsilon) \; non \; \acute{e} \; definita, \; dove \; a \in \Sigma \cup \{\varepsilon\}$$

Una stringa $x \in L(r)$, dove $\boldsymbol{r}$ é un'espressione regolare riconosciuta dall'automa se:

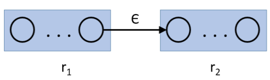
$$\exists q \in F \quad tale \; che \quad q \in \delta^*(q_0, x)$$

Osservazioni:

- Viene detta **mossa spontanea**, perchè la transizione avviene consumando il carattere ε dal dispositivo di ingresso, cioè senza consumare alcun carattere (spontaneamente). In opposizione, le altre mosse vengono dette **non spontanee**
- Le $\varepsilon-$mosse servono per costruire gli automi, ossia unire pezzi di automi tra di loro

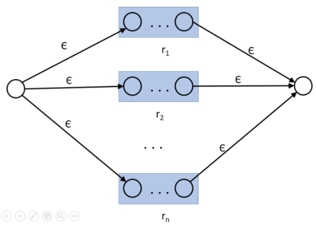
Composizione: Concatenamento

$$r = r_1 \bullet r_2$$



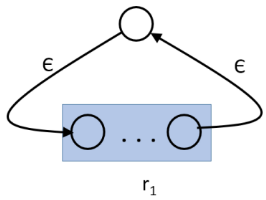
Composizione: Alternativa

$$r = (r_1 \cup r_2 \cup \ldots \cup r_n)$$



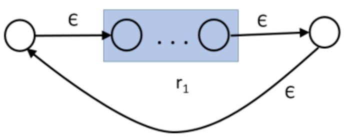
Composizione: Stella

$$r = (r_1)^*$$



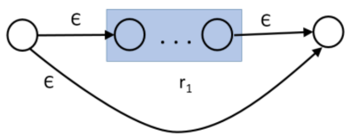
Composizione: Croce

$$r = (r_1)^+$$



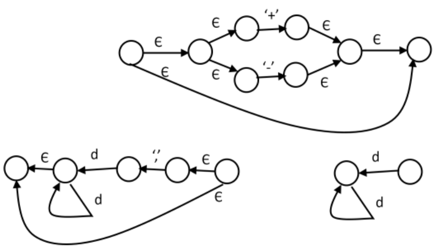
Composizione: Opzionalità

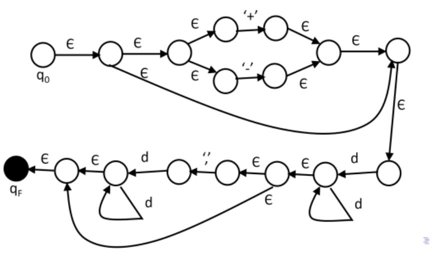
$$r = [r_1]$$



4.4 Prova: esercizio

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





4.5 Algoritmo Complemento - formale

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

- Sia $M(Q, \Sigma, \delta, q_0, F)$ l'automa deterministico per L
- Sia $\overline{M}(\overline{Q}, \Sigma, \overline{\delta}, q_0, \overline{F})$ l'automa per $\neg L$ da calcolare

Passaggi:

1. Aggiungere agli stati Q uno **stato pozzo** p, dunque $\overline{Q} = Q \cup \{p\}$
2. $\forall q \in Q$ e $\forall a \in \Sigma$, $\quad \overline{\delta}(q, a) = \delta(q, a)$ se $\delta(q, a)$ é definita, altrimenti $\overline{\delta}(q, a) = p$
3. $\forall a \in \Sigma$, $\quad \overline{\delta}(p, a) = p$
4. gli stati finali sono dunque $\overline{F} = (Q - F) \cup \{p\}$

4.6 Algoritmo per Eliminare le ε-mosse

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

1. Inserire lo stato q_0 in Q' e in N, dove N é insieme dei nuovi stati
2. Impostiamo l'insieme $N' = \emptyset$. $\forall q \in N$, copiare da δ tutte le mosse non spontanee che escono da q, cioé $\delta(q, a) = \overline{q}$, creando $\delta'(q, a) = \overline{q}$ aggiungendo $\overline{q}$ in Q' e in N', se non è già presente in Q'.
3. $\forall q \in N$, cercare tutte le transizioni transitive $q \overset{+}{\Rightarrow} \overline{q}$ che contengono una sola mossa non spontanea preceduta e/o seguita da mosse spontanee, cioé:

   $$q \overset{\varepsilon}{\Rightarrow} \ldots \overset{\varepsilon}{\Rightarrow} q_i \overset{a}{\Rightarrow} \overline{q},$$
   $$q \overset{a}{\Rightarrow} \ldots \overset{\varepsilon}{\Rightarrow} q_i \overset{\varepsilon}{\Rightarrow} \overline{q},$$
   $$q \overset{\varepsilon}{\Rightarrow} \ldots \overset{a}{\Rightarrow} q_i \overset{\varepsilon}{\Rightarrow} \overline{q}$$

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

   Attenzione: gli stati $\overline{q}$ da considerare per i pattern sono tutti e solo gli stati $\overline{q}$ 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, perchè introdurrebbero inutile ridondanza).
4. $\forall q \in N$ se esiste un percorso che va da q ad uno stato $\overline{q}$ composto solo di ε-mosse, con $\overline{q} \in F$, inserire q in F' (stato finale di M').
5. Se $N' \neq \emptyset$, $N = N'$ e ripartire dal Passo 2, altrimenti terminare

4.7 Algoritmo per Eliminare il Non-Determinismo

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

1. Inserire lo stato q_0 in Q' e in N, dove N é insieme dei nuovi stati
2. Impostiamo l'insieme $N' = \emptyset$. $\forall q \in N$ che compare anche in Q, $\forall a \in \Sigma$ per cui $\delta(q, a) = \{q_1, q_2, \ldots, q_n\}$ non vuoto
   - se $n > 1$, creare uno stato **collettivo** $[q_1, q_2, \ldots, q_n]$ e aggiungere la mossa $\delta'(q, a) = [q_1, q_2, \ldots, q_n]$. Se $[q_1, q_2, \ldots, q_n]$ non é giá 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 é giá in Q'.
3. $\forall [q_1, q_2, \ldots, q_n] \in N$, $\forall q_i \in [q_1, q_2, \ldots, q_n]$ e $\forall a \in \Sigma$ calcolare il nuovo stato collettivo $[\delta(q_1, a) \cup \delta(q_2, a) \cup \ldots \cup \delta(q_n, a)]$
   - aggiungere questo stato in Q' e in N' se non é giá in Q'.
   - se il nuovo stato collettivo contiene un solo stato, diventa uno stato semplice.
   - definire $\delta'([q_1, q_2, \ldots, q_n], a) = [\delta(q_1, a) \cup \delta(q_2, a) \cup \ldots \cup \delta(q_n, a)]$

4. $\forall q = [q_1, q_2, \ldots, q_n] \in N$ (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', cioé $q \in F'$

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

4.8 Algoritmo Complemento - informale

- Aggiungere stato pozzo p
- Tutte le transizioni che esistevano precedentemente vengono riportate in $\overline{M}$
- Tutte le transizioni mancanti vengono aggiunte per ogni stato ma vanno a finire nello stato pozzo:

$$\forall q_i \in Q : \forall k \in \Sigma \ \textit{where} \ k \notin \textit{uscita}(q_i) \overset{k}{\to} p$$

- Creare anello su p con tutte le transizioni Σ
- Scambiare stati finali con stati iniziali + p finale

4.9 Algoritmo per Eliminare le ε-mosse - informale

1. Parto da q_0 **copio** le mosse non spontanee e modello le mosse spontanee **creando** le nuove transizioni contenendo al massimo un carattere terminale e n mosse spontanee in questa maniera creo nuovi stati adiacenti a q_0, ossia i q_i

2. $\forall q_i$ dal quale **esce almeno una mossa non spotanea** applico gli stessi passaggi precedenti: **creando** nuove transizioni per le spotanee e **copiando** quelle esistenti per le non spotanee.

   Attenzione: gli stati $\overline{q}$ da considerare per i pattern sono tutti e solo gli stati $\overline{q}$ 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, perchè introdurrebbero inutile ridondanza).

3. Se $q_i \overset{n,\varepsilon}{\to} q_f$, dove $q_f \in F$, q_i diventa uno stato finale

4. Se non esisteno q_i nuovi mi fermo altrimenti riparto dal passo 2

4.10 Algoritmo per Eliminare il Non-Determinismo - informale

1. Parto da q_0 e $\forall t \in \Sigma$ creo un nuovo stato q_k:
   - se ho n transizioni con t che arrivano in n stati q_i, ossia $\delta_n(q_i, t)$ creo uno stato $q_k = [q_i]$
   - se ho una sola transizione con t che arriva in q, ossia $\delta(q, t)$, allora $q_k = q$

2. Per ogni stato collettivo $q_k = [q_i]$ itero: $\forall t \in \Sigma$ e $\forall q_i \in q_k$ e vedo dove portano le transizioni
   - viene creato un nuovo stato collettivo con la stessa transizione
   - viene creato uno stato semplice

3. Se lo stato collettivo contiene almeno uno stato finale diventa finale

4.11 Osservazioni

- L'algoritmo per calcolare il complemento é applicabile su un automa deterministico
- Se devo calcolare $r_1 \cap r_2 \equiv \neg(\neg r_1 \cup \neg r_2)$ con r_1 e r_2 devono essere automi deterministici poi applico il complemento
- Utilizzo ε mosse per:
  - scelta di percorso senza consumare alcun carattere: **opzionalitá** $[a] = a \cup \varepsilon$, **croce**, **unione** (alternativa)
- con $\delta^*(q, \varepsilon) = \{q\}$ se $\delta(q, \varepsilon)$ non é definita, ossia rimango nello stesso stato iniziale
- Si raccoglie solo se l'ordine dell'espressione regolare lo consente senza modificare la struttura dell'espressione regolare
- ***precedenza operatori:*** $*, \bullet, \cup$

4.12 Robe importanti:

$e^k = e \dots e \quad k \geq 0 \textit{ volte}$

$[e]_k^h = e^k \cup e^{k+1} \cup \dots \cup e^h \quad \textit{con } 0 \leq k \leq h$

$e^+ = ee^*$

$[e] = [e]_0^1 = \varepsilon \cup e \quad \textit{opzionalita}$

$$L^1 = L \bullet L^0 = L^1 \bullet \{\varepsilon\} = L$$

$$L^* = \{\varepsilon\} \cup L^1 \cup L^2 \cup L^\infty$$

$$L^+ = L^1 \cup L^2 \cup L^\infty$$

$$\textit{Se } \{\varepsilon\} \in L \;\; \Rightarrow \;\; L^+ = L \bullet L^* = L^* \bullet L = L^*$$

$$\textit{Se } \{\varepsilon\} \notin L \;\; \Rightarrow \;\; L^+ = L \bullet L^* = L^* \bullet L$$

5 Grammatiche BNF

5.1 Contenuto capitolo

Le espressioni regolari hanno una significativa limitazione: **non sono in grado di gestire gli annidamenti e le strutture parentetiche $\Rightarrow$ Grammatiche BNF (Backus-Naur Form)**

Definizione Grammatica BNF: Una Grammaticha BNF è descritta da una tupla $G(V, \Sigma, P, S)$

V	*alfabeto non terminale*
Σ	*alfabeto terminale*
P	*insieme delle regole di produzione*
$S \in V$	*assioma*

Si parte sempre da S

Definizione Regola di produzione: é una coppia ordina (X, α), $X \in V$ e $\alpha \in (V \cup \Sigma)^*$

$$X \to \alpha$$

Definizione Derivazioni: date 2 stringhe $\beta, \gamma \in (V \cup \sigma)^*$, si dice che γ **deriva** da β se:

γ deriva β per la grammatica G	$\beta \to \gamma \qquad \beta = \eta A \delta,\ \ \gamma = \eta\alpha\delta,\ \ A \to \alpha \in P$
Catena di derivazioni	$\beta_0 \to \ldots \to \beta_n \equiv \beta_0 \xrightarrow{n} \beta_n$

Definizione 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

Definizione 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 é il simbolo piú a destra la grammatica é detta **lineare destra**
- Se il non terminale in questione é il simbolo piú a sinitra la grammatica é detta **lineare sinistra**

Definizione: I **linguaggi regolari** sono descritti da grammatiche lineari destre o lineari sinistre.

Definizione Ambiguità:

- Data una grammatica G, una frase x del linguaggio $L(G)$ é **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

Simbologia

$V = \{A, B, C\}$	*alfabeto non terminale*
$\Sigma = \{a, b, c\}$	*alfabeto terminale*
P	*insieme delle regole di produzione*
$S \in V$	*assioma*

5.2 Definizioni

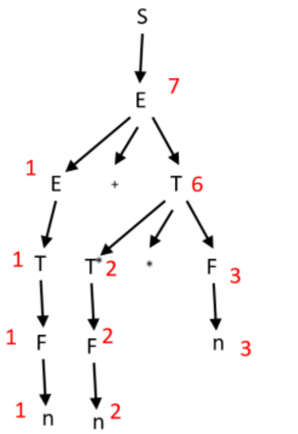
Grammatica:	$G(V, \Sigma, P, S)$
Regola di produzione:	$X \to \alpha, X \in V, \alpha \in (V \cup \Sigma)^*$
γ *deriva* β *per la grammatica* G	$\beta \to \gamma,\ \ \beta = \eta A \delta,\ \ \gamma = \eta\alpha\delta,\ \ A \to \alpha \in P$
α *si riduce ad* A	$A \to \alpha \in P$ *regola in* G
Catena di derivazioni	$\beta_0 \to \ldots \to \beta_n \equiv \beta_0 \xrightarrow{n} \beta_n$
Da non-t A*, il linguaggio generato da* G	$L_A(G) = \{x \in \Sigma^* \mid A \xrightarrow{+} x\}$
Da assioma S*, il lunguaggio generato da* G	$L(G) = L_S(G) = \{x \in \Sigma^* \mid S \xrightarrow{+} x\}$
Derivazione per "aann" (ese)	$S \to aL \to aaaL \to aann\varepsilon$
α *é una forma di frase*	$S \xrightarrow{*} \alpha, \alpha \in (V \cup \Sigma)^*$
Forma generata da G *partendo da* A *é una stringa* α	$A \xrightarrow{*} \alpha, A \in V, \alpha \in (V \cup \Sigma)^*$
Frase di $L(G)$ *é una forma di frase (stringa)*	*che non contiene simboli non-t*
Una grammatica G *é pulita o ridotta se valgono:*	$\forall A \in V, S \xrightarrow{+} \alpha A \beta$ *AND* $L_A(G) \neq \emptyset$
Una derivazione é detta ricorsiva	$A \xrightarrow{n} xAy$*, non-t* A *é detto ricorsivo*
Ricorsione sinistra	$A \xrightarrow{n} Ay, x = \varepsilon$
Ricorsione destra	$A \xrightarrow{n} xA, y = \varepsilon$
* $L(G)$ *sia infinito, dove* G *é pulita (priva di deri. circ.)*	*é che* G *permetti delle derivazioni ricorsive*
Derivazione canonica destra: $\beta_0 \to \beta_1 \to \ldots \to \beta_n$	$\beta_i = \eta_i A_i \delta_i,\ \ \beta_{i+1} = \eta_i \alpha_i \delta_i,$ *se* $\forall i, 0 \leq i \leq n-1, \delta_i \in \Sigma^*$
Derivazione canonica sinistra: $\beta_0 \to \beta_1 \to \ldots \to \beta_n$	$\beta_i = \eta_i A_i \delta_i,\ \ \beta_{i+1} = \eta_i \alpha_i \delta_i,$ *se* $\forall i, 0 \leq i \leq n-1, \eta_i \in \Sigma^*$

* ***condizione necessaria e sufficiente***

5.3 Esercizio da imparare a memoria: pattern stesso

Esercizio 3.09: Espressioni Matematiche

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



6 Parsing Discendente LL(1)

Ricostruisce le derivazioni Canoniche Sinistre $\to$ LL(1)

L = Left-to-Right *L = Left-most* *(1) = Look-ahead 1*

Definzione Look-ahead = prospezione: quanti simboli terminali occorre guardare in avanti per decidere (senza consumarli

Definizione LL(1): Un non-t $A \in V$ é *LL1* se, per ogni coppia di regole di produzione con la stessa parte sinsitra risulta $Gui(A \to \alpha_1) \cap Gui(A \to \alpha_2) = \emptyset$. Una Grammatica G è $LL(1)$ se tutti i suoi non-t sono $LL1$.

$$A \to \alpha_1, \quad A \to \alpha_2$$

Calcolo degli Insiemi degli Inizi di una stringa α, dove $A \to \alpha$ con $\alpha \in (V \cup \Sigma)^*$

- $a \in Ini(\alpha)$ se $\alpha = a\beta$
- $a \in Ini(\alpha)$ se $\alpha = A\beta$ e $a \in Ini(\gamma)$, con $A \to \gamma$
- $a \in Ini(\alpha)$ se $\alpha = A\beta$ e $Annullabile(A)$ e $a \in Ini(\beta)$ con $\beta \in (V \cup \Sigma)^*$
- $Ini(\varepsilon) = \emptyset$

Calcolo dei seguiti di un non-terminale $A \in V$, dove $A \to \alpha$

- $\{\swarrow\} \in Seg(S)$
- $a \in Seg(A)$ se $B \to \alpha A\beta$ e $a \in Ini(\beta)$
- $a \in Seg(A)$ se $B \to \alpha A$, con $B \neq A$, e $a \in Seg(B)$
- $a \in Seg(A)$ se $B \to \alpha A\beta$, con $B \neq A$, $Annullabile(\beta)$ e $a \in Seg(B)$

6.1 Definzioni:

L'Insieme degli Inizi di una stringa $\alpha \in (V \cup \Sigma)^*$	$Ini(\alpha) = \{a \in \Sigma \mid \alpha \xrightarrow{*} a\beta\}, Ini(\varepsilon) = \emptyset$
Una stringa $\alpha \in (V \cup \Sigma)^*$ *é annullabile se*	$\alpha \xrightarrow{*} \varepsilon$, *(derivo la string* ε*)*
Insieme dei Seguiti di un non-t, $A \in V$	$Seg(A) = \{a \in \Sigma \cup \{\swarrow\} \mid S \xrightarrow{*} \alpha A a \beta\}$
Insieme Guida di una Regola di Produzione: $A \to \alpha$	$Gui(A \to \alpha) = Ini(\alpha)$ *se* $\neg$ *Annullabile*(α)
	$Gui(A \to \alpha) = Ini(\alpha) \cup Seg(A)$ *se Annullabile*(α)
	$Gui(A \to \varepsilon) = Seg(A)$
Un non-t $A \in V$ *é LL(1) se* $\forall p \in P$:	$Gui(A \to \alpha_1) \cap Gui(A \to \alpha_2) = \emptyset$
Una Grammatica G è $LL(1)$ *se*	$\forall X \in V, X$ *é* $LL(1)$

6.2 Proprietá:

Fine stringa $\{\swarrow\}$	$\{\swarrow\} \in Seg(S)$ *e* $\{\swarrow\} \in Seg(A)$ *se* $S \xrightarrow{+} \alpha A$
non Annullabile :	$A \to Ba, B \to \varepsilon$

6.3 Esempio:

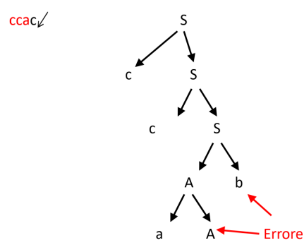
- $\Sigma = \{a, b, c\}$
- $V = \{S, A\}$
- Regole di Produzione
  $S \to A\ b$
  $S \to c\ S$
  $A \to a\ A$
  $A \to \epsilon$
- Assioma: S

- $Annullabile(A) = Vero$ $\quad Seg(A) = \{b\}$
  $Annullabile(S) = Falso$ $\quad Seg(S) = \{\swarrow\}$
- Equazioni Insiemistiche

$Ini(Ab) = Ini(A) \cup Ini(b)$	$Ini(Ab) = Ini(A) \cup \{b\}$
$Ini(A) = Ini(aA) \cup \emptyset$	$Ini(A) = \{a\}$
$Ini(cS) = \{c\}$	$Ini(cS) = \{c\}$

$Ini(Ab) = \{a\} \cup \{b\} = \{a, b\}$
$Ini(A) = \{a\}$
$Ini(cS) = \{c\}$

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



6.4 Risoluzione LL(1)

- Equazioni Insiemistiche per gli Insiemi degli Inizi

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

Regole
$S \to A\ B$
$A \to a\ A$
$A \to \epsilon$
$B \to B\ b$
$B \to C$
$C \to c$

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

- **Tecnica iterativa a Punto Fisso**
  - **Passo 0:** per ogni regola di produzione $A_i \to \alpha_i$ si definisce l'insieme degli **inizi immediati** (ottenibile direttamente da α_i)

$$Ini^0(A_i \to \alpha_i) = Ini^0(\alpha_i),\ \ inoltre$$

$$Ini^0(A) = \bigcup Ini^0(A_i \to \alpha_i), \forall A \in \Sigma$$

  - **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 $A_i \to \alpha_i$ risulta $Ini^j(A_i \to \alpha_i) \neq Ini^{j-1}(A_i \to \alpha_i)$, si continua, altrimenti ci si ferma (punto fisso raggiunto).

Equazioni Insiemistiche per gli Insiemi degli Inizi

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

Calcolo Insieme degli Inizi

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

Equazioni Insiemistiche per gli Insiemi dei Seguiti

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

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	
$S \to A\ B$	$\{a, c\}$	$\{a, c\}$	
$A \to a\ A$	$\{a\}$	$\{a\}$	
$A \to \epsilon$	$\emptyset$	$\{c\}$	
$B \to B\ b$	$\{c\}$	$\{c\}$	Conflitto
$B \to C$	$\{c\}$	$\{c\}$	Conflitto
$C \to c$	$\{c\}$	$\{c\}$	

6.5 Tecnica iterativa a Punto Fisso: insieme degli inizi

0. Calcolo degli **inizi immediati**. Supponiamo di avere $A \to \alpha$:
   - $\alpha = a \in \Sigma$, $Ini^0(A \to \alpha) = a$
   - $\alpha = Av \in (V \cup \Sigma)^*$, se $\neg Annullabile(A)$, $Ini^0(A \to \alpha) = \emptyset$
   - $\alpha = Av \in (V \cup \Sigma)^*$, se $Annullabile(A)$, $Ini^0(A \to \alpha) = v$
   - $\alpha = AB \in (V \cup \Sigma)^*$, se $Annullabile(A)$, $Ini^0(A \to \alpha) = \emptyset$
   - $\alpha = \varepsilon$, $Ini^0(A \to \alpha) = \emptyset$
1. Passo iterativo j, continuare fino a quando $Ini^j = Ini^{j-1}$. Supponiamo di avere $A \to \alpha$:
   - $\alpha = BC$, con $\neg Annnullabile(B)$, $Ini^j(A \to \alpha) = Ini(A \to \alpha)^0 \cup Ini^J(B)$
   - $\alpha = BC$, con $Annnullabile(B)$, $Ini^j(A \to \alpha) = Ini(A \to \alpha)^0 \cup Ini^J(B) \cup Ini^J(C)$

Con $Ini(X)$ nel passo iterativo j si pone "freccia" tutte le regole di produzione che hanno come regole di produzione $X \to x$ e si prendono i loro inizi.

6.6 Tecnica iterativa a Punto Fisso: insieme dei seguiti

-1. Per definizione $Seg(S) = \{\swarrow\}$

0. I **seguiti immediati** sono calcolati su tutti i non terminali. Occorre guardare la parte destra di ogni regola di produzione e cercare il simbolo terminale e vedere quello che lo succede. Supponiamo di avere $B \to \alpha A \beta \gamma$
   - se $|\beta| \neq 0$,$\neg Annullabile(\beta)$, $Ini(\beta) \in Seg(A)$
   - se $|\beta| \neq 0$,$Annullabile(\beta)$, $Ini(\beta), Ini(\gamma) \in Seg(A)$
   - se $|\beta| = 0$, $Seg(A)^0 = \emptyset$
   - se $\beta = b$, $Seg(A)^0 = \{b\}$
1. Passo iterativo j, continuare fino a quando $Seg^j = Seg^{j-1}$. Supponiamo di avere $B \to \alpha A \beta$:
   - se $|\beta \neq 0|$, $Annullabile(\beta)$, $Seg(A)^j = Seg(A)^0 \cup Seg(B)^j$, con $B \neq A$
   - se $|\beta = 0|$, $Seg(A)^j = Seg(A)^0 \cup Seg(B)^j$, con $B \neq A$

6.7 Insieme Guida di una Regola di Produzione

Data una regola di produzione $A \to \alpha$

- se $\neg$ $Annullabile(\alpha)$, $Gui(A \to \alpha) = Ini(A \to \alpha)$
- se $Annullabile(\alpha)$, $Gui(A \to \alpha) = Ini(A \to \alpha) \cup Seg(A)$
- $Gui(A \to \varepsilon) = Seg(A)$

Un non-t $A \in V$ é LL(1) se $\forall p \in P$:	$Gui(A \to \alpha_1) \cap Gui(A \to \alpha_2) = \emptyset$
Una Grammatica G è $LL(1)$ se	$\forall X \in V, X$ *é* $LL(1)$

7 Parsing Ascendente LR(0)

Ricostruisce le derivazioni Canoniche Destre $\rightarrow$ LR(0)

L = Left-to-Right *R = Right-most* *(0) = Look-ahead 0*

Cioè senza look-ahead, si consuma il simbolo e si fa qualche cosa (ma senza look-ahead risulta limitata)

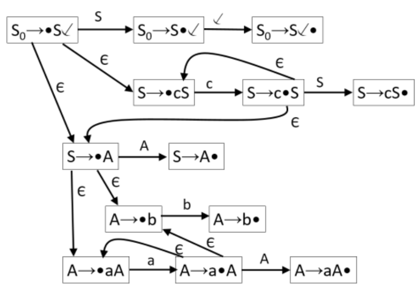
7.1 Parti Destre e Automi

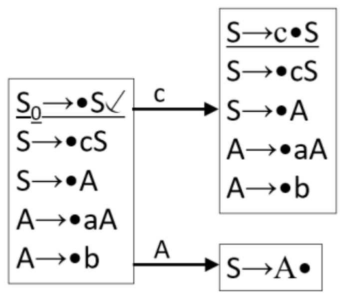
- La parte destra di una regola di produzione $A \rightarrow \alpha$ è una stringa basata sull'alfabeto $(V \cup \Sigma)$
- Un semplice automa a stati finiti con alfabeto $(V \cup \Sigma)$ é in grado di riconoscerla
- Possiamo immaginare di avere tanti micro-automi, uno per ogni regola
- Consideriamo la grammatica seguente, con $\Sigma = \{a, b, c\}$ e $V = \{S, A\}$ (S_0 é il **super Assioma**)

$$S_0 \rightarrow S \swarrow$$

7.2 Funzionamento

- Il simbolo $\bullet$ indica la testina di lettura, quando un simbolo viene consumato, si sposta a destra
- Cosa vuol dire consumare un non-terminale? vuol dire aver **riconosciuto il sotto-albero in esso radicato**, mettiamo delle ε-mosse di raccordo per collegare le regole di profuzione con lo stesso non-t, ma così facendo l'automa diventa non-deterministico e inoltre, occorre associare una pila per gestire gli annidamenti e i punti di decisione
- 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 $\bullet$





7.3 Costruzione Automa LR(0)

- Data la grammatica $G(V, \Sigma, P, S_0)$ con $S_0 \rightarrow S \swarrow \in P$ e S_0 non ricorsivo
- Uno stato s è un **insieme di candidate**
- Una candidata c ha la seguente forma: $N \rightarrow \alpha \bullet \beta$ *con* $\alpha, \beta \in (V \cup \Sigma)^*$ *e* $N \rightarrow \alpha\beta \in P$
- Una candidata é di **spostamento** se $N \rightarrow \alpha \bullet \beta$ con $|\beta| > 0$
- Una candidata é di **riduzione** se $N \rightarrow \alpha \bullet \beta$ con $|\beta| = 0$
- Una candidata é di **core** se $N \rightarrow \alpha \bullet \beta$ con $|\alpha| \neq 0$ o se é $S_0 \rightarrow \bullet S \swarrow$
- Una candidata é di **completamento** se $N \rightarrow \alpha \bullet \beta$ con $|\alpha| = 0$ e se vi è una candidata $\overline{c} = M \rightarrow \overline{\alpha} \bullet N\overline{\beta}$ nello stesso stato s
- **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 é un insieme di candidate $m(A) = \{c_{A_1}, \dots, c_{A_n}\} \neq 0$ con $c_{A_i} : N_i \rightarrow \alpha_i \bullet A\beta_i$ e in s_2 vi è un insieme $\overline{m}(A) = \{k_{A_1}, \dots, k_{A_n}\} \mid \forall c_{A_i} \in m(A) \ \exists \ k_{A_i} \mid k_{A_i} : N_i \rightarrow \alpha_i A \bullet \beta_i$

7.4 Procedimento Costruttivo

- **Passo 1:** lo stato iniziale I_1 contiene contiene la candidata core $S_0 \to S \swarrow$
- **Passo 2:** $\forall s, \forall c = M \to \alpha \bullet N\beta$ con $N \in V$ si aggiungono in s le candidate di completamento $N \to \bullet\alpha_i$, per ogni regola di produzione $N \to \alpha_i \in P$. Si continua ad aggiungere candidate di completamento fino a che tutti i simboli non-terminali marcati con $\bullet$ sono stati completati
- **Passo 3:** $\forall s$, si determinano le mosse uscenti. $\forall m(a) = \{c_{A_i}\}$ con $c_{A_i} : N_i \to \alpha_i \bullet A\beta_i$ $(A \in (V \cup \Sigma)$ cioè le candidate con lo stesso simbolo marcato da $\bullet$, si crea (se non è già presente) uno stato $\overline{s}$ le cui candidate core sono esattamente $\overline{m}(A) = \{k_{A_i}\} \mid k_{A_i} : N \to \alpha_i A \bullet \beta_i$ deriva da $c_{A_i} : N_i \to \alpha_i \bullet A\beta_i$. Si aggiunge la transizione $S \xrightarrow{A} \overline{s}$. Per ogni nuovo stato s, si ripete dal passo.

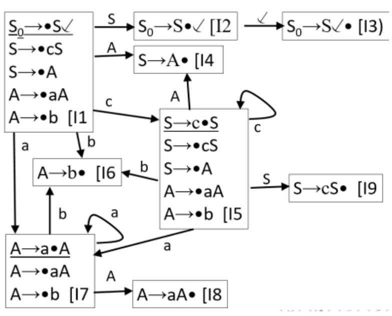
7.5 Conflitti

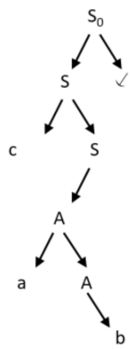
- Si dice che uno stato s presenta un **conflitto spostamento/riduzione** se contiene sia una candidata di spostamento che una candidata di riduzione.
- Si dice che uno stato s presenta un **conflitto riduzione/riduzione** se contiene due diverse candidate di riduzione
- Una stato s é **LR(0)** se non contiene conflitti di alcun tipo, mentre un automa è detto **LR(0)** se tutti i suoi stati sono **LR(0)** e conseguentemente, anche la grammatica è detta **LR(0)**

7.6 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 é definita segnalare errore, altrimenti mettere s' in cima alla pila
- **Passo 4:** se lo stato s in cima alla pila può ridurre la regola $N \to \alpha$ (dalla candidata $N \to \alpha\bullet$) 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

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 $A \to b$
$\swarrow$	$I_1 c I_5 a I_7 A I_8$	Riduzione $A \to aA$
$\swarrow$	$I_1 c I_5 A I_4$	Riduzione $S \to A$
$\swarrow$	$I_1 c I_5 S I_9$	Riduzione $S \to cS$
$\swarrow$	$I_1 S I_2$	
	$I_1 S I_2 \swarrow I_3$	Riduzione $S_0 \to S \swarrow$
	$I_1 S_0$	OK





7.7 Insiemi di Prospezione o look ahead 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
- 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 $LA(s, N \to \alpha\bullet)$ o, se lo stato s è sottinteso, $LA(N \to \alpha\bullet)$

$$LA(s, N \to \alpha\bullet) = \{a \in \Sigma \mid \exists S_0 \xrightarrow{*} \gamma N a z \to \gamma\alpha a z \ AND \ \delta^*(I_1, \gamma\alpha) = s\} \quad \textit{derivazione destra}$$

7.8 Condizioni LALR(1)

Uno stato s contenente $LA(s, N \to \alpha\bullet)$ si dice **Adeguato se**

- Per ogni coppia di candidate $N_1 \to \alpha_1\bullet$ e $N_2 \to \alpha_2\bullet \Rightarrow LA(s, N_1 \to \alpha_1\bullet) \cap LA(s, N_2 \to \alpha_2\bullet) = \emptyset$
- Per ogni candidata di riduzione $LA(s, N \to \alpha\bullet) \Rightarrow LA(s, N \to \alpha\bullet) \cap \{b \in \Sigma \mid \delta(s, b)$ *é definita*$\} = \emptyset$

Definzione: la grammatica G é **LALR(1)** se tutti gli stati dell'automa sono adeguati (uno stato LR(0) è, di per sé, adeguato), quindi se $LR(0) \Rightarrow LALR(1)$

7.9 Metodo Operativo iterativo Calcolo LA

In uno stato s, data una candidata $c : N \to \alpha \bullet \beta$ vogliamo calcolare $Seg_s(N \to \alpha \bullet \beta)$

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

$$Seg_s(N \to \alpha \bullet \beta) = \bigcup_{s_j} Seg_s(N \to \bullet\alpha\beta), \forall s_j : \delta^*(s_j, \alpha) = definita$$

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

$$Seg_s(N \to \bullet\beta) = \bigcup_i Seg_s(N, c_i), \forall c_i : M_i \to \alpha_i \bullet N\beta_i$$

$$\begin{aligned} &Seg_s(N, c_i) = Ini(\beta_i) && se\ \neg Annullabile(\beta_i) \\ &Seg_s(N, c_i) = Ini(\beta_i) \cup Seg_s(M_i \to \alpha_i \bullet N\beta_i) && se\ Annullabile(\beta_i) \end{aligned}$$

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

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

7.10 Automa LR(1)

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

7.11 Creazione Automa LR(1)

- Nello stato I_1 la candidata core $S_0 \to \bullet S \swarrow$ ha $LA(I_1, S_0 \to \bullet S \swarrow) = \emptyset$
- In ogni stato s, nelle candidate di completamento $N \to \bullet\beta$ e le candidate $c_j : M_j \to \alpha_j \bullet N\beta_j$ dove:

$$\begin{aligned} &Seg_s(N, c_j) = Ini(\beta_j) && se\ \neg Annullabile(\beta_j) \\ &Seg_s(N, c_j) = Ini(\beta_j) \cup LA(s, M_j \to \alpha_j \bullet N\beta_j) && se\ Annullabile(\beta_j) \end{aligned}$$

$$LA(s, N \to \bullet\beta) = \bigcup_{c_j} Seg(N, c_j)$$

- Le candidate core di uno stato diverso da I_1 mantengono il look-ahead set delle candidate di provenienza, quindi se esiste: $s \xrightarrow{X} \overline{s}$, (con $X \in (\Sigma \cup V)$), abbiamo $N \to \alpha\bullet X\beta \in s$ e $N \to \alpha X \bullet \beta \in \overline{s}$ con $LA(\overline{s}, N \to \alpha X \bullet \beta) = LA(s, N \to \alpha \bullet X\beta)$
- 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**
- Le candidate di completamento di uno stato cambiano generalemnte look-ahead
- 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

7.12 Definzioni LR(0)

S_0 *é il super Assioma*	$S_0 \rightarrow S \swarrow$
*Uno stato **s/I***	*é un insieme di candidate*
*Una candidata **c***	$N \rightarrow \alpha \bullet \beta$ *con* $\alpha, \beta \in (V \cup \Sigma)^*$ *e* $N \rightarrow \alpha\beta \in P$
*Candidata di **spostamento***	$N \rightarrow \alpha \bullet \beta,$ *con* $\vert\beta\vert > 0$
*Candidata di **riduzione***	$N \rightarrow \alpha \bullet \beta,$ *con* $\vert\beta\vert = 0$
*Candidata di **core***	$N \rightarrow \alpha \bullet \beta,$ *con* $\vert\alpha\vert \neq 0,$ *o se é* $S_0 \rightarrow \bullet S \swarrow$
*Candidata di **completamento***	$N \rightarrow \alpha \bullet \beta,$ *con* $\vert\alpha\vert = 0$ *e* $\overline{c} = M \rightarrow \overline{\alpha} \bullet N\overline{\beta} \in s$
Mosse uscenti $s_1 \xrightarrow{A} s_2, A \in (V \cup \Sigma)$	$\forall c_{A_i} \in s_1, c_{A_i} : N_i \rightarrow \alpha_i \bullet A\beta_i$
	$\forall k_{A_i} \in s_2, k_{A_i} : N_i \rightarrow \alpha_i A \bullet \beta_i$
$N \in V$ ***marcato*** *(deve essere letto)*	$M \rightarrow \alpha \bullet N\beta$
Gruppo di candidate	$m(a) = \{c_{A_i}\}$
Conflitto spotamento/riduzione	$\exists c_a, c_b$ *con* $ca \neq c_b : c_a \in spostamento, c_b \in riduzione$
Conflitto riduzione/riduzione	$\exists c_a, c_b$ *con* $ca \neq c_b : c_a \in riduzione, c_b \in riduzione$
*Una stato s é **LR(0)***	*se non contiene alcun conflitto*
*Un Automa è detto **LR(0)***	$\forall s \in LR(0)$

7.13 Definizioni LALR(1)

$s \notin LR(0)$	$\forall$ *candidate di riduzione* $\in s$ *si associa* $LA(s, N \rightarrow \alpha\bullet)$
LA	$LA(s, N \rightarrow \alpha\bullet) = \{a \in \Sigma \mid \exists S_0 \xrightarrow{*} \gamma Naz \rightarrow \gamma\alpha az \; AND \; \delta^*(I_1, \gamma\alpha) = s\}$
s adeguato se:	$N_1 \rightarrow \alpha_1 \bullet$ *e* $N_2 \rightarrow \alpha_2\bullet \Rightarrow LA(s, N_1 \rightarrow \alpha_1\bullet) \cap LA(s, N_2 \rightarrow \alpha_2\bullet) = \emptyset$
	$\forall LA(s, N \rightarrow \alpha\bullet) \Rightarrow LA(s, N \rightarrow \alpha\bullet) \cap \{b \in \Sigma \mid \delta(s, b)$ *é definita*$\} = \emptyset$
dato s	*é LALR(1) se:* $\forall s \in adeguato$
	se $s \in LR(0) \Rightarrow LALR(1)$
$ini(\beta)$	*carattere immediatamente dopo il* β *marcato*

8 Esercizio 2

8.1 Calcolo LR(0)

- Data la grammatica $G(V, \Sigma, P, S_0)$ si aggiunge $S_0 \to S \swarrow \in P$
- Si crea il primo stato I_1 con candidata core (per definizione) $S_0 \to \bullet S \swarrow$
- Si aggiungono tute le candidate di completamento per ogni simbolo non-terminale marcato (esempio: $\bullet A \in V$) all'interno dello stesso stato
- Si definiscono tutte le mosse uscenti per ogni regola di produzione: $s_1 \xrightarrow{\alpha} s_2$, $\alpha \in (V \cup \Sigma)$
- Si e prosegue iterativamente allo stesso modo per ogni stato

8.2 Calcolo LALR(1)

- Calcolo il LR(0)
- Controllo se esistono conflitti negli stati: se stato s presenta conflitto, s non é LR(0)
  - **Conflitto riduzione/riduzione:** $\exists c_a, c_b \ con \ c_a \neq c_b : c_a \in riduzione, c_b \in riduzione$
  - **Conflitto spostamento/riduzione:** $\exists c_a, c_b \ con \ c_a \neq c_b : c_a \in spostamento, c_b \in riduzione$
- $\forall s$ non LR(0) calcolo **Look-ahead set** $Seg_s(N \to \alpha \bullet \beta)$ solo sulle candidate di riduzione. Si tratta di un processo iterativo, inizio da una candidata di riduzione per poi applicare (a) e (b):
  (a) se $|\alpha| > 0$, candidata core, i seguiti arrivano da stati esterni:
  $$Seg_s(N \to \alpha \bullet \beta) = \bigcup_{s_j} Seg_s(N \to \bullet \alpha\beta), \forall s_j : \delta^*(s_j, \alpha) = definita$$
  (b) se $|\alpha| = 0$, candidata completamento, i seguiti arrivano da altre candidate nello stesso stato s:
  $$Seg_s(N \to \bullet\beta) = \bigcup_{i} Seg_s(N, c_i), \forall c_i : M_i \to \alpha_i \bullet N\beta_i, \quad M_i \neq N$$
  $$Seg_s(N, c_i) = Ini(\beta_i) \qquad se \ \neg Annullabile(\beta_i)$$
  $$Seg_s(N, c_i) = Ini(\beta_i) \cup Seg_s(M_i \to \alpha_i \bullet N\beta_i) \qquad se \ Annullabile(\beta_i)$$
- **NB:** in uno stato s NON LR(0), data una candidata di riduzione $N \to \alpha\bullet$ si calcola
  $$LA(N \to \alpha\bullet) = Seg_s(N \to \alpha\bullet)$$

8.3 Nota bene:

Candidata core dello stato:	$M_i \to \alpha_i \bullet N\beta_i$
Candidata di completamento:	$N \to \bullet\beta$

8.4 Condizioni LALR(1)

Uno stato s contenente $LA(s, N \to \alpha\bullet)$ si dice **Adeguato se**

- Per ogni coppia di candidate $N_1 \to \alpha_1\bullet$ e $N_2 \to \alpha_2\bullet \Rightarrow LA(s, N_1 \to \alpha_1\bullet) \cap LA(s, N_2 \to \alpha_2\bullet) = \emptyset$
- Per ogni candidata di riduzione $LA(s, N \to \alpha\bullet) \Rightarrow LA(s, N \to \alpha\bullet) \cap \{b \in \Sigma \mid \delta(s, b) \ \text{é definita}\} = \emptyset$

Definzione: la grammatica G é **LALR(1)** se tutti gli stati dell'automa sono adeguati (uno stato LR(0) è, di per sé, adeguato), quindi se $LR(0) \Rightarrow LALR(1)$

Esempio: $I_2 : A \rightarrow \alpha \bullet$

Gli LA vengono calcolati solo per le candidate di riduzione. ***Parto da una candidata di core:*** *devo andare negli stati esterni I_i che hanno come mossa uscente α e cercare all'interno di ogni I_i la regola di produzione che deve consumare α* (a).

$I_{10} : X \rightarrow vX\bullet$	*andare stati adiacenti*	$c \in core$
$I_6 \xrightarrow{X} I_{10}$	$I_6 : X \rightarrow v \bullet X$	$c \in core$
$I_4 \xrightarrow{v} I_6$	$I_4 : X \rightarrow \bullet vX$	$c \in completamento$

Ora mi ritrovo ad avere una ***regola di produzione di completamento:*** *devo cercare all'interno dello stesso stato $\overline{I}_i$ la regola di produzione che mi ha portato a quel punto, ossia il terminale marcato in questione e selezioni tutte le candidate c_i con quel simbolo marcato*

c di completamento:	*simbolo marcato nello stato corrente*
$I_4 : \boldsymbol{X} \rightarrow \bullet vX$	$c_1 : L \rightarrow \bullet \boldsymbol{X} g$
	$\cancel{c_2 : X \rightarrow \bullet \boldsymbol{X} n}$
	$c_3 : Z \rightarrow \bullet \boldsymbol{X} Z$
	$c_4 : Z \rightarrow \bullet \boldsymbol{X} W$

$\forall c_i$ selezionata calcolo i seguiti:

$$Seg_{I_4}(X, c_1) = Ini(g) = g \qquad se\ \neg Annullabile(a)$$
$$Seg_{I_4}(X, c_3) = Ini(Z) \qquad se\ \neg Annullabile(Z)$$
$$Seg_{I_4}(X, c_4) = Ini(Z) \cup Seg_{I_4}(Z \rightarrow \bullet XW) \qquad se\ Annullabile(W)$$

Nell'ultimo caso: $eg_{I_4}(Z \rightarrow \bullet XW)$, devo trovare tutti i $\bullet \boldsymbol{Z}$ (Z marcati) precedenti risalendo nella pila

8.5 Calcolo LR(1)

- Nello stato I_1 la candidata core $S_0 \rightarrow \bullet S \swarrow$ ha $LA(I_1, S_0 \rightarrow \bullet S \swarrow) = \emptyset$
- 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:

$$Seg_s(N, c_j) = Ini(\beta_j) \qquad se\ \neg Annullabile(\beta_j)$$
$$Seg_s(N, c_j) = Ini(\beta_j) \cup LA(s, M_j \rightarrow \alpha_j \bullet N\beta_j) \qquad se\ Annullabile(\beta_j)$$

$$LA(s, N \rightarrow \bullet\beta) = \bigcup_{c_j} Seg(N, c_j)$$

- Le candidate core di uno stato diverso da I_1 mantengono il look-ahead set delle candidate di provenienza, quindi se esiste: $s \xrightarrow{X} \overline{s}$, (con $X \in (\Sigma \cup V)$), abbiamo $N \rightarrow \alpha \bullet X\beta \in s$ e $N \rightarrow \alpha X \bullet \beta \in \overline{s}$ con $LA(\overline{s}, N \rightarrow \alpha X \bullet \beta) = LA(s, N \rightarrow \alpha \bullet X\beta)$
- 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**
- La candidate core mantengono gli stessi LA dallo stato di partenza al nuovo stato, mentre le candidate di completamento all'interno del nuovo stato cambiando quansi sempre LA.
- 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

8.6 Osservazioni:

- quando la candidata é una core, $A \rightarrow \alpha\beta \bullet S$, devo andare indietro di: $\alpha\beta$ mosse, fino ad arrivare ad ottenere $A \rightarrow \bullet\alpha\beta S$ e cercare ora all'interno dello stato $\overline{A}$
- quando la candidata é di completamento, $A \rightarrow \bullet\alpha\beta S$, devo cercare nello stato corrente $\overline{A}$ initemize

- quando la candidata é una core, $A \to \alpha\beta \bullet S\gamma$, devo andare indietro di: $\alpha\beta$ mosse, fino ad arrivare ad ottenere $A \to \bullet\alpha\beta S$ e cercare ora all'interno dello stato $\overline{A}$
- Se la $\overline{A}$ é tipo: $N \to \alpha\beta \bullet A\gamma$, LA é dato da:

$$\begin{array}{ll} Annullabile(\gamma) & ini(\beta) \\ \neg Annullabile(\gamma) & ini(\beta) \cup LA(N \to \alpha\beta \bullet A\gamma) \\ & \textit{e poi si riparte iterativamente} \end{array}$$