Nota ai lettori

Questi appunti sono basati sulle lezioni dell A.A. 2023/2024 tenute dal Prof. Alessio Bottrighi, integrate con passi tratti dal libro "Linguaggi Formali e Compilazione" ed appunti forniti dal docente.

1 Alfabeto e linguaggio

Un *alfabeto* è un insieme finito di elementi chiamati *simboli terminali* o *caratteri*. Ad esempio

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

è un alfabeto composto da 3 elementi a, b, c (la sua cardinalità è 3).

Una *stringa* (o *parola*) è una sequenza, ovvero un insieme ordinato eventualmente con ripetizioni, di caratteri.

Un *linguaggio* è un insieme di stringhe di un alfabeto specifico. Dato un linguaggio, una stringa che gli appartiene è detta *frase*. Ad esempio, possiamo definire un linguaggio L

$$L = \{a, ab, bc, cccc\}$$

le cui parole al suo interno sono formate esclusivamente delle lettere dell'alfabeto specificato in precedenza.

La *cardinalità* di un linguaggio è definita dal numero di frasi che contiene. Se la cardinalità è finita, il linguaggio si dice *finito*.

Un linguaggio finito è una collezione di parole, solitamente chiamate *vocabolario*. Il linguaggio che non contiene frasi è chiamato *insieme vuoto* o *linguaggio* $\emptyset$.

La *lunghezza* $|x|$ di una stringa x è il numero di caratteri che contiene.

1.1 Operazioni sulle stringhe

Stringa vuota

La *stringa vuota* (o *nulla*), denotata con ε, soddisfa l'identità:

$$x \cdot \varepsilon = \varepsilon \cdot x = x$$

La stringa vuota non deve essere confusa con l'insieme vuoto; infatti, l'insieme vuoto è un linguaggio che non contiene stringhe, mentre l'insieme $\{\varepsilon\}$ ne contiene una, la stringa vuota.

Sottostringa

Sia la stringa $x = uyv$ il prodotto della concatenazione delle stringhe u, y e v: le stringhe u, y e v sono *sottostringhe* di x. In questo caso, la stringa u è un *prefisso* di x e la stringa v è un *suffisso* di x. Una sottostringa non vuota è detta *propria* se non coincide con x.

Concatenazione

Date le stringhe

$$x = a_1 a_2 \dots a_h \qquad\qquad y = b_1 b_2 \dots b_k$$

la *concatenazione*, indicata con '$\cdot$', è definita come:

$$x \cdot y = a_1 a_2 \dots a_h b_1 b_2 \dots b_k$$

La concatenazione non è commutativa, ma è associativa.

Inversione di stringa

L'*inverso* di una stringa $x = a_1 a_2 \dots a_h$ è la stringa $x^R = a_h a_{h-1} \dots a_1$.

Ripetizione

La potenza m-esima x^m di una stringa x è la concatenazione di x con se stessa per $m-1$ volte. Esempi:

$$x = ab \qquad x^0 = \varepsilon \qquad x^2 = (ab)^2 = abab$$

1.2 Operazioni sul linguaggio

Linguaggio neutro

Il linguaggio contenente esclusivamente la stringa vuota è detto *linguaggio neutro*. Ha cardinalità pari a 1.

$$L_N = \{\varepsilon\}$$
$$L \cdot L_N = L_N \cdot L = L$$

Linguaggio vuoto

Il linguaggio vuoto non contiene alcuna stringa, quindi la sua cardinalità è 0. Si indica con $\emptyset$.

$$L \cdot \emptyset = \emptyset \cdot L = \emptyset$$

Concatenazione

La concatenazione tra due linguaggi è il prodotto cartesiano tra le stringhe di entrambi i linguaggi. Ad esempio, dati i linguaggi L_1 e L_2

$$L1 = \{a, b, c\} \qquad L_2 = \{bb, cc\}$$

concatenandoli si ottiene

$$L_1 \cdot L_2 = \{abb, acc, bbb, bcc, cbb, ccc\}$$

Inversione

L'inverso L^R di un linguaggio L è l'insieme delle stringhe che sono l'inverso di una frase di L.

Ripetizione

Come per le stringhe, è possibile l'elevamento a potenza.

$$L^m = L^{m-1} \cdot L \text{ per } m \geq 1 \qquad L^0 = \{\varepsilon\}$$

1.3 Operazioni sugli insiemi

Dato che un linguaggio è un insieme, si possono usare gli operatori unione '$\cup$', intersezione '$\cap$' e differenza '$\setminus$'. Sono applicabili inoltre le relazioni di inclusione '$\subseteq$', inclusione stretta '$\subset$' ed uguaglianza '$=$'.

Il *linguaggio universale* è l'insieme di tutte le stringhe, su un alfabeto Σ, di ogni lunghezza inclusa 0. Il linguaggio universale è infinito.

$$L_{universale} = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots$$

Il *complemento* di un linguaggio L su un alfabeto Σ, denotato con $\neg L$, è la differenza insiemistica

$$\neg L = L_{universale} \setminus L$$

1.4 Operatore di Kleene e croce

Per definire linguaggi infiniti, si usano due operatori: l'operatore di Kleene '$*$' e l'operatore croce '$+$'.

Operatore di Kleene

Questa operazione è definita come unione di tutte le potenze del linguaggio base:

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

Può generare un numero infinito di parole composte da un numero infinito di caratteri.

Operatore croce

Questo operatore è derivato da quello precedente:

$$L^+ = \bigcup_{h=1}^{\infty} L^h = L^1 \cup L^2 \cup L^3 \cup \dots$$

2 Linguaggi regolari

2.1 Definizione di espressione regolare

Un linguaggio su un alfabeto $\Sigma = \{a_1, a_2, \dots, a_n\}$ è *regolare* se può essere espresso applicando finite volte le operazioni di concatenazione, unione e Kleene, a partire dai linguaggi unitari $\{a_1\}, \{a_2\}, \dots, \{a_n\}$ o la stringa vuota ε.

Più precisamente, un'*espressione regolare* è una stringa r contenente i caratteri terminali dell'alfabeto Σ e i metasimboli '$\cup$' (unione), '$\cdot$' (concatenazione), '$*$' (iterazione), 'ε' (stringa vuota) e parentesi, in accordo con le seguenti regole:

regola	significato
$r = \varepsilon$	stringa vuota
$r = a$	linguaggio unitario
$r = (s \cup t)$	unione di espressioni
$r = (s \cdot t)$	concatenazione di espressioni
$r = (s)^*$	iterazione di un'espressione

dove i simboli s e t sono espressioni regolari.

Esempio di espressione regolare

Proviamo a creare un linguaggio che generi tutti i numeri naturali, con o senza segno. L'alfabeto per questo linguaggio sarà:

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -\}$$

Il primo simbolo della parola dovrà necessariamente essere '$+$' o '$-$', quindi la prima parte della nostra espressione regolare sarà:

$$(+ \cup - \cup \varepsilon)$$

I simboli a destra dovranno essere delle cifre, quindi:

$$(+ \cup - \cup \varepsilon)(1 \cup 9)(0 \cup 9)*$$

In questo modo, il primo simbolo sarà almeno 1. Per poter generare anche lo 0:

$$(+ \cup - \cup \varepsilon)((1 \cup 9)(0 \cup 9)^*) \cup 0$$

2.2 Derivazioni

Formalizziamo il processo mediante il quale una data espressione regolare e produce il linguaggio in questione. Prendiamo in esame l'espressione regolare e_0:

$$e_0 = (((a \cup (bb)^{)}*)((c^+) \cup (a \cup (bb))))$$

Questa espressione regolare è data dalla concatenazione delle due *sottoespressioni* e_1 ed e_2:

$$e_1 = ((a \cup (bb))^*) \qquad\qquad e_2 = ((c^+) \cup (a \cup (bb)))$$

La sottostringa s

$$s = (a \cup (bb))$$

è una sottoespressione di e_2 ma non di e_0.

Un operatore di unione o iterazione offre diversi modi per produrre stringhe. Effettuando una scelta, si può ottenere un'espressione regolare che definisce un linguaggio meno espressivo (in grado di generare meno parole), incluso in quello originale. Si dice che un'espressione regolare è una *scelta* di un'altra nei seguenti tre casi:

Derivazione da unione

Un'espressione regolare e_k, con $1 \leq k \leq m$ e $m \geq 2$, è una scelta dell'unione:

$$(e_1 \cup \cdots \cup e_k \cup \cdots \cup e_m)$$

Derivazione da * o +

Un'espressione regolare $e^m = e \ldots\ldots e$, con $m \geq 1$ è una scelta di e^* o e^+.

Derivazione da stringa vuota

La stringa vuota ε è una scelta di e^*.

Derivazione immediata

Si dice che un'espressione regolare e' *deriva* un'espressione regolare e'' ($e' \Rightarrow e''$) se una delle seguenti proposizioni è vera:

1. l'espressione regolare e'' è una scelta di e'
2. l'espressione regolare e' è la concatenazione di $m \geq 2$ sottoespressioni, e e'' è ottenuta da e' sostituendo una sottoespressione, ad esempio e'_k, con una scelta di e'_k, ad esempio e''_k:

$$\exists k, 1 \leq k \leq m \text{ tale che } e''_k \text{ è una scelta di } e'_k \wedge e'' = e'_1 \ldots e''_k \ldots e'_m$$

Tale derivazione è detta *immediata* in quanto effettua una sola scelta. Si dice che un'espressione regolare e_0 deriva un'espressione regolare e_n in $n \geq 1$ passi ($e_0 \Rightarrow^n e_n$) se le seguenti derivazioni immediate sono applicabili:

$$e_0 \Rightarrow e_1 \qquad\qquad e_1 \Rightarrow e_2 \qquad\qquad \ldots \qquad\qquad e_{n-1} \Rightarrow e_n$$

Esempi di derivazioni immediate:

$$a^* \cup b^+ \Rightarrow a^* \qquad\qquad a^* \cup b^+ \Rightarrow b^+ \qquad\qquad (a^* \cup bb)^* \Rightarrow (a^* \cup bb)(a^* \cup bb)$$

Le sottostringhe di un'espressione regolare sono scelte in ordine da più esterna a più interna.

2.3 Limiti dei linguaggi regolari

Le espressioni regolari, nonostante siano utili e pratiche, sono costrutti molto limitati. Ad esempio, l'espressione $a^n b^n$ con $n \geq 0$ non è formalizzabile come espressione regolare, in quanto $a^n b^n \neq a^* b^*$: infatti, il numero di a è vincolato al numero di b, non traducibile solo mediante operatori di espressioni regolari.

3 Grammatiche

Una *grammatica generativa* o *sintassi* è un insieme di semplici regole che possono essere applicate ripetutamente per generare tutte e sole le stringhe valide.

Esempio: palindromi

Il linguaggio L, descritto dall'alfabeto $\Sigma = \{a, b\}$, è definito, tramite l'operazione di inversione, come:

$$L = \{uu^R | u \in \Sigma^*\} = \{\varepsilon, aa, bb, abba, baab, \ldots, abbbba, \ldots\}$$

Contiene stringhe specularmente simmetriche. La grammatica G contiene tre regole:

$$pal \to \varepsilon \qquad pal \to a\ pal\ a \qquad pal \to b\ pal\ b$$

Per derivare le stringhe, basta rimpiazzare il simbolo *pal*, detto *non terminale*, con la parte destra della regola di generazione, ad esempio:

$$pal \Rightarrow apala \Rightarrow abpalba \Rightarrow abbpalbba \Rightarrow \ldots$$

Il processo di derivazione termina quando l'ultima stringa ottenuta non contiene più nessun carattere non terminale. Completiamo la derivazione:

$$abbpalbba \Rightarrow abb\varepsilon bba = abbbba$$

Il linguaggio dei palindromi non è regolare.

3.1 Definizione di grammatica context-free

Una grammatica *context-free* è una grammatica G definita da quattro entità:

- V *alfabeto dei non terminali*, un insieme di simboli non terminali
- Σ *alfabeto dei terminali*, un insieme di simboli terminali
- P insieme di *regole di produzione* (un linguaggio senza regole di produzione è un linguaggio vuoto)
- S un particolare non terminale detto *assioma*, dal quale parte la generazione ($S \in V$)

Una regola dell'insieme P è una coppia ordinata $X \to \alpha$, con $X \in V$ e $\alpha \in (V \cup \Sigma)^*$.

3.2 Derivazione e generazione

Sia $\beta = \delta A \mu$ una stringa contenente un non terminale A, dove δ e μ sono stringhe, possibilmente vuote. Sia $A \to \alpha$ una regola della grammatica G e sia $\gamma = \delta \alpha \mu$ la stringa ottenuta rimpiazzando il non terminale A in β con la regola nella parte destra α. Questa relazione è detta *derivazione*. Si dice che la stringa β deriva la stringa γ per la grammatica G, e scriviamo:

$$\beta \Rightarrow \gamma$$

La regola $A \to \alpha$ viene applicata in tale derivazione e la stringa α *riduce* al non terminale A. Il *linguaggio generato* o *definito* da una grammatica G, iniziando da un non terminale A, è l'insieme delle strignhe terminali che derivano da un non terminale A in uno o più passi:

$$L_A(G) = \{x \in \Sigma^* | A \Rightarrow^+ x\}$$

Se il non terminale è l'assioma S, si ha il linguaggio generato da G:

$$L(G) = L_S(G) = \{x \in \Sigma^* | S \Rightarrow^+ x\}$$

Un linguaggio è context-free se esiste una grammatica context-free che lo genera. Due grammatiche G e G' sono *equivalenti* se generano lo stesso linguaggio.

3.3 Grammatiche ridotte

Una grammatica G è detta *ridotta* se:

1. non sono presenti regole di produzione inutili (non torno alla stringa di partenza)
2. qualsiasi non terminale A è *raggiungibile* dall'assioma
3. qualsiasi non terminale A è *ben definito*, ovvero non genera un linguaggio vuoto

Esempio: espressioni aritmetiche

La grammatica G:

$$G = (\{E, T, F\}, \{i, +, \times, ', (',)'\}, P, E)$$

ha l'insieme di regole di produzione P:

$$E \to E + T | T \qquad T \to T \times F | F \qquad F \to (E) | i$$

Il linguaggio di G:

$$L(G) = \{i, i + i, i \times i, (i + i) \times i, \dots\}$$

è l'insieme di espressioni aritmetiche generato. Dato che la grammatica G è ridotta e non circolare, il linguaggio $L(G)$ è infinito.

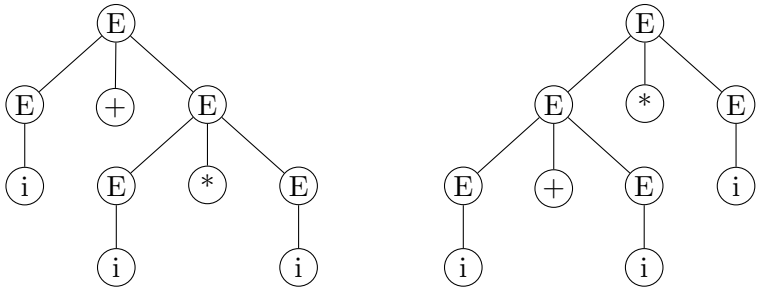
3.4 Albero di sintassi e derivazione

Il processo di derivazione può essere visualizzato come un albero, che descrive l'ordine in cui sono state derivate le parole.

La radice rappresenta l'assioma, mentre le foglie rappresentano i terminali. I nodi intermedi contengono solo nodi non terminali. Si può ricavare la parola leggendo le foglie da sinistra a destra.

Esempio: albero di derivazione delle espressioni aritmetiche

Visualizziamo l'albero di derivazione generato dal linguaggio riportato nel paragrafo precedente.



Notiamo che la stessa parola può essere generata in modi diversi (e quindi avere alberi di generazione differenti). Grammatiche di questo tipo sono dette *grammatiche ambigue*.

Per eliminare le ambiguità è possibile utilizzare delle parentesi, esplicitando la priorità tra operazioni

$$E \to E + E | E * E | i | (E)$$

ed ampliare la grammatica inserendo altre regole:

$$E \to E + T | T \qquad T \to T * F | F \qquad F \to (E) | i$$

Questa è una *grammatica stratificata*, in quanto ad ogni livello viene generata una sola operazione.

$$E \to E + T \to E + T * F \to E + T * i \to T + T * i \to F + T * i \to i + T * i \to i + F * i \to i + i * i$$

Questa grammatica non è ambigua.

3.5 Grammatiche di Chomsky e Greibach

Le grammatiche di Chomsky sono molto semplici, in quanto hanno solo due regole:

- da un non terminale si possono generare due non terminali: $A \to BC$ dove $B, C \in V$
- un non terminale può generare un simbolo dell'alfabeto o la stringa vuota ε: $A \to a$ dove $a \in \Sigma$

Con la forma normale di Greibach si possono scrivere grammatiche context free con regole di un solo tipo:

- ogni non terminale genera un terminale: $A \to a\alpha$ dove $a \in \Sigma$ e $\alpha \in V^*$

Ogni regola inizia con un terminale, seguito da zero o più non terminali.

3.6 Conversione della ricorsione da sinistra a destra

Un'altra forma normale, detta *ricorsiva non-sinistra*, è caratterizzata dall'assenza di regole ricorsive sinistre o derivazioni (l-ricorsioni).

Trasformazione della ricorsione immediata sinistra

Consideriamo tutte le alternative l-ricorsive per un non terminale A:

$$A \to A\beta_1 | A\beta_2 | \dots | A\beta_h \quad h \geq 1$$

dove nessuna stringa β_i è vuota, e siano le rimanenti alternative di A, che sono necessarie per terminare la ricorsione:

$$A \to \gamma_1 | \gamma_2 | \dots | \gamma_k \quad k \geq 1$$

Creiamo un nuovo terminale ausiliario A' e rimpiazziamo le regole precedenti con quelle mostrate in seguito:

$$A \to \gamma_1 A' | \gamma_2 A' | \dots | \gamma_k A' | \gamma_1 | \gamma_2 | \dots | \gamma_k$$
$$A' \to \beta_1 A' | \beta_2 A' | \dots | \beta_h A' | \beta_1 | \beta_2 | \dots | \beta_k$$

Ora, ogni derivazione che originalmente coinvolgeva passi l-ricorsivi, ad esempio

$$A \Rightarrow A\beta_2 \Rightarrow \beta_3\beta_2 A' \Rightarrow \gamma_1\beta_3\beta_2$$

è rimpiazzata dalla derivazione equivalente:

$$A \Rightarrow \gamma_1 A' \Rightarrow \gamma_1\beta_3 A' \Rightarrow \gamma_1\beta_3\beta_2$$

4 Automi a stati finiti

4.1 Automa riconoscitore

Per controllare se una stringa è valida per un determinato linguaggio, serve un algoritmo di riconoscimento, un tipo di algoritmo che produce una risposta positiva o negativa all'input fornito.

Gli automi sono rappresentati come grafi orientati. I nodi rappresentano gli stati; ogni arco è contrassegnato da un terminale e rappresenta la transizione causata dalla lettura di quel terminale.

5 Automi a pila e parsing

5.1 Automi a pila

Gli *automi a pila* sono automi a stati finiti che utilizzano una *pila* (stack) come memoria aggiuntiva.

Un automa a pila è definito dalla 7-upla

$$\langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$$

con:

- Q: insieme degli stati
- Σ: alfabeto che descrive il linguaggio
- Γ: alfabeto della pila
- δ: funzione di transizione
- q_0: stato iniziale
- Z_0: fondo della pila

- F: stato (o stati) finale

L'input è una tripla, denotata come:

$$(q, a, A) \to (x, XX)$$

con:

- q: stato corrente
- a: il simbolo della stringa da leggere
- A: il contenuto dello stack

Il simbolo di fine stringa è $\swarrow$.

5.1.1 Tipi di accettazione

Negli automi a pila ci sono due tipi di accettazione: l'*accettazione per stato finale*, quando è stato consumato tutto l'input e si giunge ad uno stato finale, e l'*accettazione per pila vuota*, quando è stato consumato tutto l'input e la pila è vuota (anche senza Z_0).

Essendo l'automa non deterministico, bisogna fare tutte le computazioni possibili (quindi esplorare tutte le possibilità).

5.1.2 Esempio di accettazione per stato finale

Processiamo una stringa nella forma ca^nb^n con $n \geq 1$, ad esempio $caabb\swarrow$.

$$\Gamma = \{Z_0, X\}$$

Z_0 è sempre presente, X è il simbolo che gestisce il bilanciamento. Consumando c, si passa dallo stato q_0 allo stato q_1.

$$(q_0, c, Z_0) \to (q_1, Z_0)$$

Ora non sarà più possibile incontrare c. Consumiamo a:

$$(q_1, a, Z_0) \to (q_1, Z_0X)$$

X serve a contare le a. La funzione (q_1, Z_0X) rimane in q_1; la testa della pila contiene X, quindi la funzione (q_1, a, Z_0) non può scattare. Bisogna definire una nuova:

$$(q_1, a, X) \to (q_1, XX)$$

Se la parola è corretta, prima o poi si incontrerà una b. Passiamo allo stato q_2 per non incontrare più a.

$$(q_1, b, X) \to (q_2, \varepsilon)$$

Non può esserci Z_0, altrimenti sarebbe come se non avessimo mai incontrato nessuna a. Il passaggio a q_2 è obbligato.

$$(q_2, b, X) \to (q_2, \varepsilon)$$

$$(q_2, \swarrow, X) \to (q_3, Z_0)$$

Incontrerò il fine stringa quando aavrò rimosso tutti gli X dalla pila. Lo stato finale conterrà solo q_3.

Input	Pila	Stato	Commenti
$caabb\swarrow$	Z_0	q_0	devo consumare c e Z_0
$aabb\swarrow$	Z_0	q_1	devo consumare a
$abb\swarrow$	Z_0X	q_1	devo trovare tripla (q_1, a, X)
$bb\swarrow$	Z_0XX	q_1	ogni volta che incontro una a, metto X sulla pila
$b\swarrow$	Z_0X	q_2	consumo la testa della pila, non scrivo nulla
$\swarrow$	Z_0	q_2	
	Z_0	q_3	la parola appartiene al linguaggio

5.1.3 Regole di produzione

Una *grammatica context free* genera da un non terminale una sequenza di terminali e non terminali, combinati in qualunque modo; è una quadrupla nella forma

$$G = \langle V, \Sigma, P, S \rangle$$

È possibile usare l'automa a pila per simulare la fase di generazione: quando trovo un non terminale, posso sostituirlo con un terminale o un non terminale.

La costruzione della funzione di transizione viene guidata dalle regole di produzione. Il funzionamento dell'automa a pila è il seguente: controllo l'elemento in cima alla pila, individuo la regola di produzione corrispondente e la applico.

Esistono 4 categorie di regole di generazione: regola di *inizializzazione*, regola di *terminazione*, regole *derivate da* P e regole *derivate da* Σ. Per qualunque tripla, si può applicare più di una regola.

Inizializzazione Questa regola permette di far partire la generazione, corrisponde a mettere sulla pila l'assioma S.

$$(q_0, \varepsilon, Z_0) \to (q_0, \swarrow S)$$

Implico il trovarmi in q_0 e dover transizionare in q_0. Non consumo nulla, ma modifico il contenuto della pila. Accettando per pila vuota, non bisogna includere Z_0.

Terminazione Questa regola permette di terminare la generazione; l'ultimo simbolo in input è quello di fine stringa ($\swarrow$).

$$(q_0, \swarrow, \swarrow) \to (q_0, \varepsilon)$$

La generazione termina quando l'automa incontra il simbolo di fine stringa. Non viene scritto nulla sulla pila, ma si rimuove $\swarrow$, terminando la generazione.

Regole per Σ Esiste una regola per ogni simbolo dell'alfabeto ($\forall a \in \Sigma$).

$$(q_0, a, a) \to (q_0, \varepsilon)$$

Il simbolo in cima alla pila viene consumato. Esistono due tipi di regole di produzione per a, quelle che *iniziano con un terminale*

$$(q_0, a, A) \to (q_0, \beta^R) \quad per \quad A \to a\beta$$

e quelle che *iniziano con un non terminale*

$$(q_0, \varepsilon, A) \to (q_0, \beta^R X) \quad per \quad A \to X\beta$$

5.1.4 Esempio di accettazione per pila vuota

Creiamo un automa a stati finiti non deterministico che accetta per *pila vuota*:

- $Q = \{q_0\}$: perchè si può gestire il tutto con un solo stato (dato il non determinismo) e l'insieme degli stati finali è vuoto.
- $\Sigma = \Sigma$: l'alfabeto è quello del linguaggio
- $\Gamma = \{Z_0, \dots\}$: conterrà sicuramente il simbolo di fine pila, più tutti i simboli scrivibili sulla pila
- $F = \{\emptyset\}$: l'insieme degli stati finali è vuoto

L'alfabeto della pila è definito come

$$\{Z_0\} \cup \Sigma \cup V$$

ovvero l'unione del simbolo di fine pila e gli insiemi dei simboli terminali e non terminali.

Regole di produzione:

$$S \to aBA \qquad S \to bcS \qquad B \to Ba \qquad B \to A \qquad A \to ac \qquad A \to AA$$

La funzione di transizione è composta da 11 regole. Le seguenti regole di inizializzazione e terminazione

$$(q_0, \varepsilon, Z_0) \to (q_0, \swarrow S) \qquad\qquad (q_0, \swarrow, \swarrow) \to (q_0, \varepsilon)$$

sono comuni a tutti i linguaggi.

Le regole

$$(q_0, a, a) \to (q_0, \varepsilon) \qquad (q_0, b, b) \to (q_0, \varepsilon) \qquad (q_0, c, c) \to (q_0, \varepsilon)$$

non scrivono nulla sulla pila.

Infine

$$\begin{array}{lll} (q_0, a, S) \to (q_0, AB) & (q_0, b, S) \to (q_0, Sc) & (q_0, \varepsilon, B) \to (q_0, aB) \\ (q_0, \varepsilon, B) \to (q_0, A) & (q_0, a, A) \to (q_0, c) & (q_0, \varepsilon, A) \to (q_0, AA) \end{array}$$

Generiamo la stringa *aacac* sequendo le regole di produzione ed esaminiamola.

$$S \to aBA \to aAA \to aacA \to aacac$$

Input	Pila	Stato	Regola di produzione
$aacac \swarrow$	Z_0	q_0	$(q_0, \varepsilon, Z_0) \to (q_0, \swarrow S)$
$aacac \swarrow$	$\swarrow S$	q_0	$(q_0, a, S) \to (q_0, AB)$
$acac \swarrow$	$\swarrow AB$	q_0	$(q_0, \varepsilon, B) \to (q_0, A)$
$acac \swarrow$	$\swarrow AA$	q_0	$(q_0, a, A) \to (q_0, c)$
$cac \swarrow$	$\swarrow Ac$	q_0	$(q_0, c, c) \to (q_0, \varepsilon)$
$ac \swarrow$	$\swarrow A$	q_0	$(q_0, a, A) \to (q_0, c)$
$c \swarrow$	$\swarrow c$	q_0	$(q_0, c, c) \to (q_0, \varepsilon)$
$\swarrow$	$\swarrow$	q_0	$(q_0, \swarrow, \swarrow) \to (q_0, \varepsilon)$

5.2 Parsing

L'*albero di derivazione* è creato durante la parsificazione.

Si possono avere due politiche diverse durante la derivazione di un albero: *dall'alto verso il basso* e *dal basso verso l'alto*. Parser di questo tipo sono automi a pila.

5.2.1 Parser di tipo $LR(0)$

Vediamo un parser di tipo $LR(0)$.

Con 0 intendiamo che, oltre a consumare un simbolo in input, *legge 0 altri simboli.*

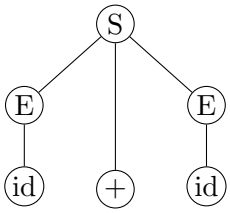
Con L intendiamo *left*: il parser parte da sinistra con la lettura.

Con R intendiamo *rightmost*: il parser cerca la regola della grammatica da utilizzare partendo da quella più a destra.

Si inseriscono nodi nell'albero ogni volta che si effettua una riduzione. Ad esempio, date le seguenti regole di produzione

$$E \to id \qquad\qquad S \to E + E$$

si ottiene l'albero



$LR(0)$ è un'*automa a pila deterministico*: in ogni momento della parsificazione è possibile compiere una sola azione (o nessuna). Il suo compito è accettare o rifiutare una stringa in input. Sono inoltre possibili due operazioni:

- *SHIFT*: leggo input e lo trascrivo sulla pila
- *REDUCE*: operazione legata ad una regola grammaticale; consuma simboli dalla pila e li sostituisce

L'operazione di REDUCE non modifica la pila, fa una serie di pop e poi fa una push.

Finora, gli stati sono stati identificati per label. In $LR(0)$ gli stati sono etichettati con "SHIFT" o "REDUCE" e contengono informazioni utili a determinare il tipo di operazione da svolgere.

Un parser $LR(0)$ non gestisce tutte le grammatiche context free, ma è possibile costruire un parser a partire da una di queste.

Durante la parsificazione di possono verificare due problemi:

- il comportamento non è deterministico: alcuni stati hanno due o più comandi
- si possono avere più operazioni di reduce, ognuna legata ad una regola diversa (qual è quella corretta)

Inoltre, *non* è possibile fare contemporaneamente operazioni di SHIFT e REDUCE oppure due operazioni di REDUCE in parallelo.

Un automa a pila deterministico ha all'interno dei suoi stati dei candidati legati alla regola di produzione. $A \rightarrow a^{\beta}$