Nota ai lettori

Questi appunti sono basati sulle lezioni dell A.A. 2023/2024, integrate con passi tratti dal libro "Linguaggi Formali e Compilazione", e formattati seguendo la suddivisione in paragrafi di quest'ultimo.

1 Teoria formale del linguaggio

1.1 Alfabeto e linguaggio

Un **alfabeto** è un insieme finito di elementi chiamati **simboli terminali** o **caratteri**. $\Sigma = \{a_1, a_2, \ldots, a_k\}$ è un alfabeto composto da k elementi (la sua cardinalità è k). 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**.

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

Un linguaggi 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.

1.1.1 Operazioni sulle stringhe

Date le stringhe

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

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

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

La concatenazione non è commutativa, ma è associativa.

1.1.2 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 il set $\{\varepsilon\}$ ne contiene una, la stringa vuota.

1.1.3 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.

1.1.4 Inversione di stringa

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

1.1.5 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

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

2 Automi a pila e parsing

2.1 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:

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

con:

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

Il simbolo di fine stringa è $\swarrow$.

2.1.1 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).

2.1.2 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

2.1.3 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**

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

2.1.4 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

$$(q_0, a, S) \to (q_0, AB) \qquad (q_0, b, S) \to (q_0, Sc) \qquad (q_0, \varepsilon, B) \to (q_0, aB)$$
$$(q_0, \varepsilon, B) \to (q_0, A) \qquad (q_0, a, A) \to (q_0, c) \qquad (q_0, \varepsilon, A) \to (q_0, AA)$$

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

2.2 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.

2.2.1 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**.

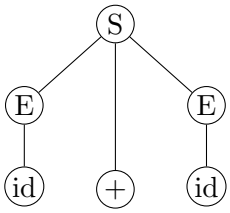
Con L intendiamo **left**: il parser parte da sinistra con la lettura.

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. $A \to a^{\beta}$