UNIVERSITÀ
DEGLI STUDI
DI TORINO

Parsificazione top-down e bottom-up

a.a. 2018-2019

Analisi sintattica

Data una grammatica G l'*analizzatore sintattico* o *parsificatore* legge la stringa sorgente e se appartiene al linguaggio L(G) ne produce una derivazione o un albero sintattico, altrimenti si ferma segnalando l'errore (in alternativa prosegue ignorando le sottostringhe contaminate dall'errore).

Due classi importanti di analizzatori

Discendenti o top-down

Ascendenti o bottom-up

Come per l'analisi lessicale, anche per l'analisi sintattica sono stati sviluppati strumenti per la generazione automatica di parsificatori, sia per l'analisi bottom-up (Yacc) sia per quella top-down (Antlr).

Analisi sintattica

Analizzatori discendenti

Costruiscono un albero di parsificazione iniziando dalla radice e creando i nodi in preordine.

Equivalentemente la parsificazione top-down può essre vista come ricerca di una derivazione leftmost per la stringa in input.

Un analizzatore discendente espande le parti sinistre delle regole della grammatica nelle corrispondenti parti destre.

Analizzatori ascendenti

Costruiscono l'albero dalle foglie alla radice.

L'analisi ascendente può essere vista come ricerca di una derivazione rightmost della stringa in ordine contrario.

Un analizzatore ascendente riduce le parti destre delle regole della grammatica nei corrispondenti non terminali.

Analisi sintattica

Produzioni:

$S \rightarrow S\#L \mid L \qquad L \rightarrow a,L \mid b,L \mid a \mid b$

Derivazione left-most

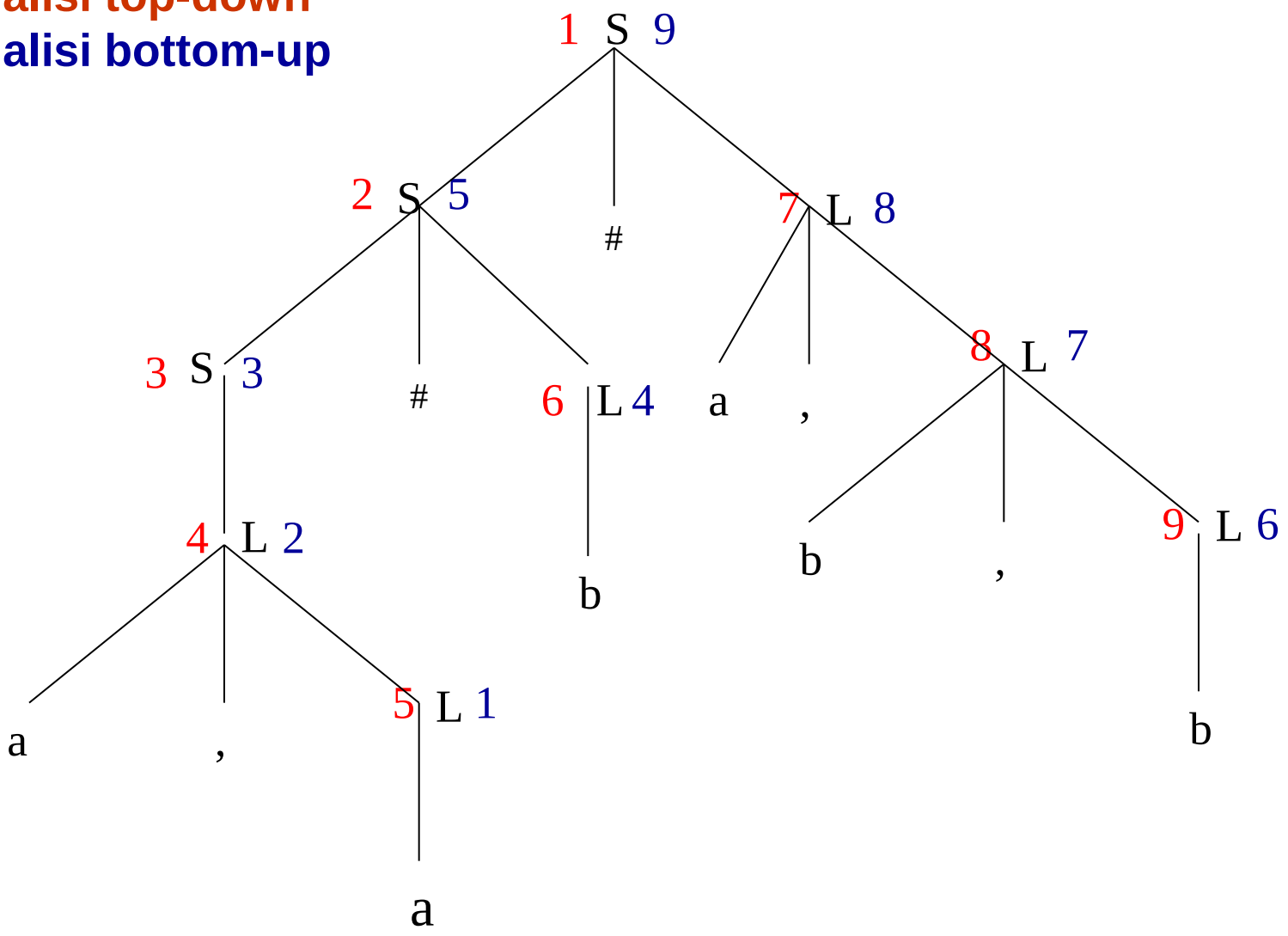
S
1 ⇒ S#L
2 ⇒ S#L#L
3 ⇒ L#L#L
4 ⇒ a,L#L#L
5 ⇒ a,a#L#L
6 ⇒ a,a#b#L
7 ⇒ a,a#b#a,L
8 ⇒ a,a#b#a,b,L
9 ⇒ a,a#b#a,b,b

Derivazione rightmost (al contrario…)

a,a#b#a,b,b
1 ⇒ a,L#b#a,b,b
2 ⇒ L#b#a,b,b
3 ⇒ S#b#a,b,b
4 ⇒ S#L#a,b,b
5 ⇒ S#a,b,b
6 ⇒ S#a,b,L
7 ⇒ S#a,L
8 ⇒ S#L
9 ⇒ S

Analisi sintattica

Analisi top-down
Analisi bottom-up



Analisi sintattica

Poichè i riconoscitori dei linguaggi context-free sono gli automi non deterministici, la parsificazione nella forma più generale può richiedere il backtracking per trovare la produzione corretta da applicare durante l'esame di una stringa.

Vi sono diversi modi per assicurare a priori che un linguaggio sia analizzabile deterministicamente imponendo restrizioni sul linguaggio o sulla grammatica.

In molte aree dell'informatica i linguaggi sono progettati in modo da essere analizzabili deterministicamente.
Quasi tutti i linguaggi di programmazione e i linguaggi XML della rete sono in questa classe.

Analisi sintattica

La costruzione generale dell'automa a pila (non deterministico) dalla grammatica sostituisce ad una variabile A in cima alla pila la parte destra di una produzione per A con una ε-mossa (quindi "alla cieca"). Se invece, per fare questa mossa, permettiamo di utilizzare k caratteri dell'input ancora da analizzare possiamo in certi casi scegliere in modo univoco la produzione da sostituire ad A sulla pila (cioè la produzione da usare in una derivazione a sinistra).

Prendiamo per esempio il linguaggio $\{a^n c b^n \mid n \geq 0\}$ generato dalla grammatica con le produzioni: $S \rightarrow aSb \mid c$

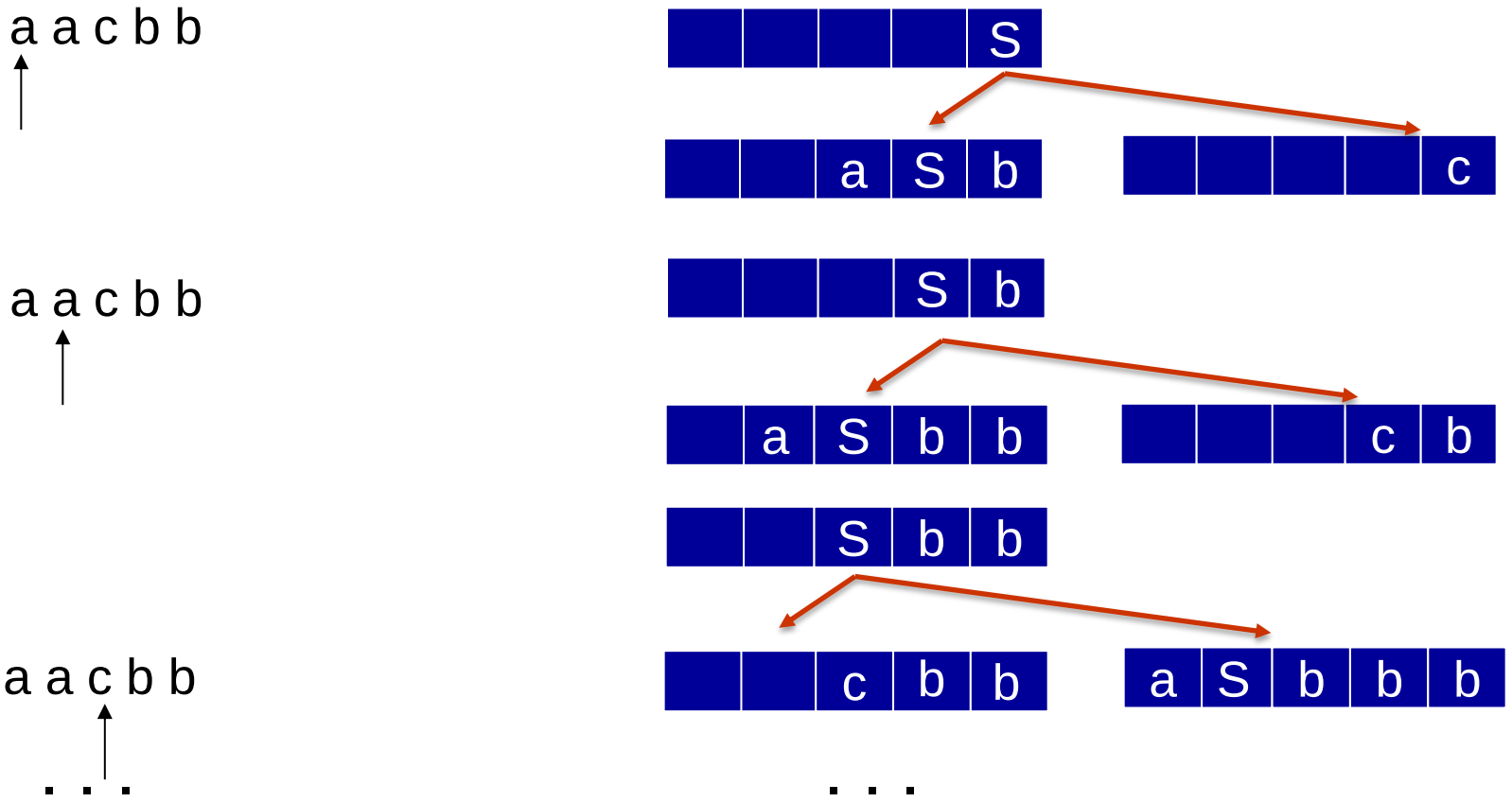
Avendo S sulla pila l'automa lo sostituirà con aSb se il primo carattere in input è 'a', con 'c' se il primo carattere dell'input è 'c' e segnalerà un errore se il primo carattere in input non è né 'a' né 'c' .

Non può sbagliare!

Ovviamente non sempre questo è possibile….

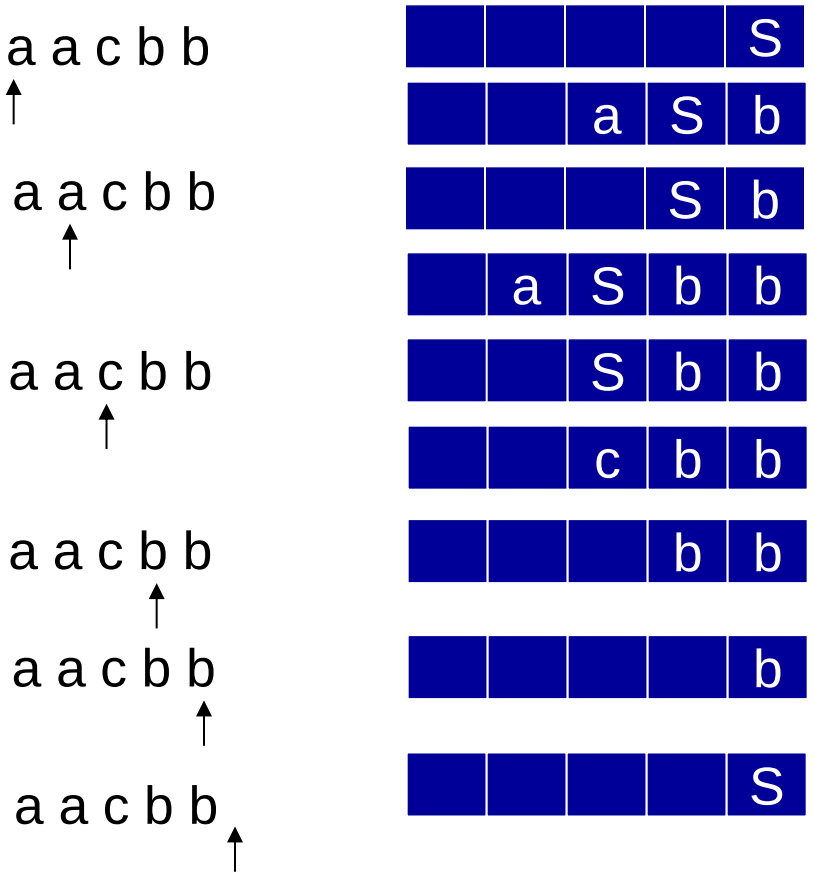
Analisi sintattica

Consideriamo il comportamento dell'automa costruito dall'algoritmo sulla stringa aacbb.



Analisi sintattica

Se si tiene conto del carattere in input, tale comportamento può essere modificato, scegliendo in modo deterministico la riscrittura per S:



Analisi sintattica

Consideriamo il linguaggio generato dalla grammatica:

$$S \rightarrow 0S1 \mid 0A1 \qquad A \rightarrow 2A \mid \varepsilon$$

Il non determinismo dell'automa costruito a partire dalla grammatica, è dovuto, da un lato, alla presenza, nella definizione di $\delta(q, \varepsilon, S)$, di due possibili transizioni: {(q, 0S1), (q, 0A1)}, dall'altro alla scelta, in presenza della variabile A sul top dello stack, tra la transizione (q, 2A) e (q, ε).

Osservazioni:

- Avendo un carattere a disposizione quando A è sul top dello stack l'automa è in grado di decidere se applicare la transizione spontanea o "sostituire" sullo stack A con 2A.

- Un carattere non basta invece per decidere se sostituire S sullo stack con 0S1 o 0A1, ma se il carattere successivo è un altro 0 deve essere usata la prima, se è 1 o 2 la seconda: sono sufficienti due caratteri per scegliere la produzione senza errore.

Analisi sintattica deterministica

Leggendo 1 o più caratteri in input si possono *eliminare le incertezze* e scegliere sempre la strada giusta che porta al riconoscimento della stringa: analizzatore sintattico *deterministico*.

Il modello su cui si basano tutti gli analizzatori sintattici (più o meno fedelmente) è l'automa a pila.

Le grammatiche che permettono analisi sintattica top-down deterministica o "**parsing predittivo discendente**" sono chiamate **LL(k)**, quelle che permettono analisi sintattica bottom-up deterministica o "**parsing predittivo ascendente**" sono chiamate **LR(k)**, dove k è il numero di simboli necessari per individuare la produzione senza ambiguità.

La definizione di LL(k) e LR(k) viene estesa in modo ovvio ai linguaggi: un linguaggio è LL(k) (LR(k)) se esiste una grammatica LL(k) (LR(k)) che lo genera.

Analisi sintattica deterministica

- Gli analizzatori top-down sono in generale più intuitivi e facili da realizzare.
- L'analisi ascendente (bottom-up) è più "*potente*" di quella discendente (top-down), cioè permette di analizzare una classe più ampia di linguaggi.

Non tutti i linguaggi che hanno riconoscitori deterministici sono generabili da grammatiche LL(k), cioè la famiglia dei linguaggi LL(k) è *strettamente contenuta nella famiglia dei linguaggi che hanno riconoscitori deterministici*.
Ogni linguaggio regolare è generato da una grammatica LL(1).

La famiglia dei linguaggi analizzabili in modo deterministico coincide con quella dei linguaggi generati da grammatiche LR(1), cioè, per ogni linguaggio analizzabile in modo deterministico esiste una grammatica LR(1) che lo genera.