Limbaje Formale și Compilatoare (LFC)
- Curs -

Ș.l.dr.ing Octavian MACHIDON

octavian.machidon@unitbv.ro

Universitatea
Transilvania
din Brașov

Astăzi

- Analiza sintactică
  - Automate push-down
  - Parsere LL
  - Analiza sintactică LR

Tipuri de analiză sintactică

- Descendenta (top-down, de sus in jos)
  - Inlocuieste cate un neterminal cu partea dreapta a unei productii, pana ramane doar cu terminali
- Ascendenta (bottom-up, de jos in sus)
  - Porneste de la sirul de atomi lexicali, abstractizeaza din sir simbolul de start prin reduceri succesive
- Analiza descendenta – derivare stanga
  - Tot timpul inlocuim cel mai din stanga neterminal
  - LL (**L**eft to right, **L**eftmost derivation)
- Analiza ascendenta - derivarea dreapta
  - primul neterminal înlocuit este cel mai din dreapta din forma propozi ională curentă
  - LR (**L**eft to right, **R**ightmost derivation)

Automatizarea parsării LL

- Echivalenta cu un automat push-down
- Parsarea se poate face cu un automat si o tabela.

Generarea vs. recunoașterea unui limbaj

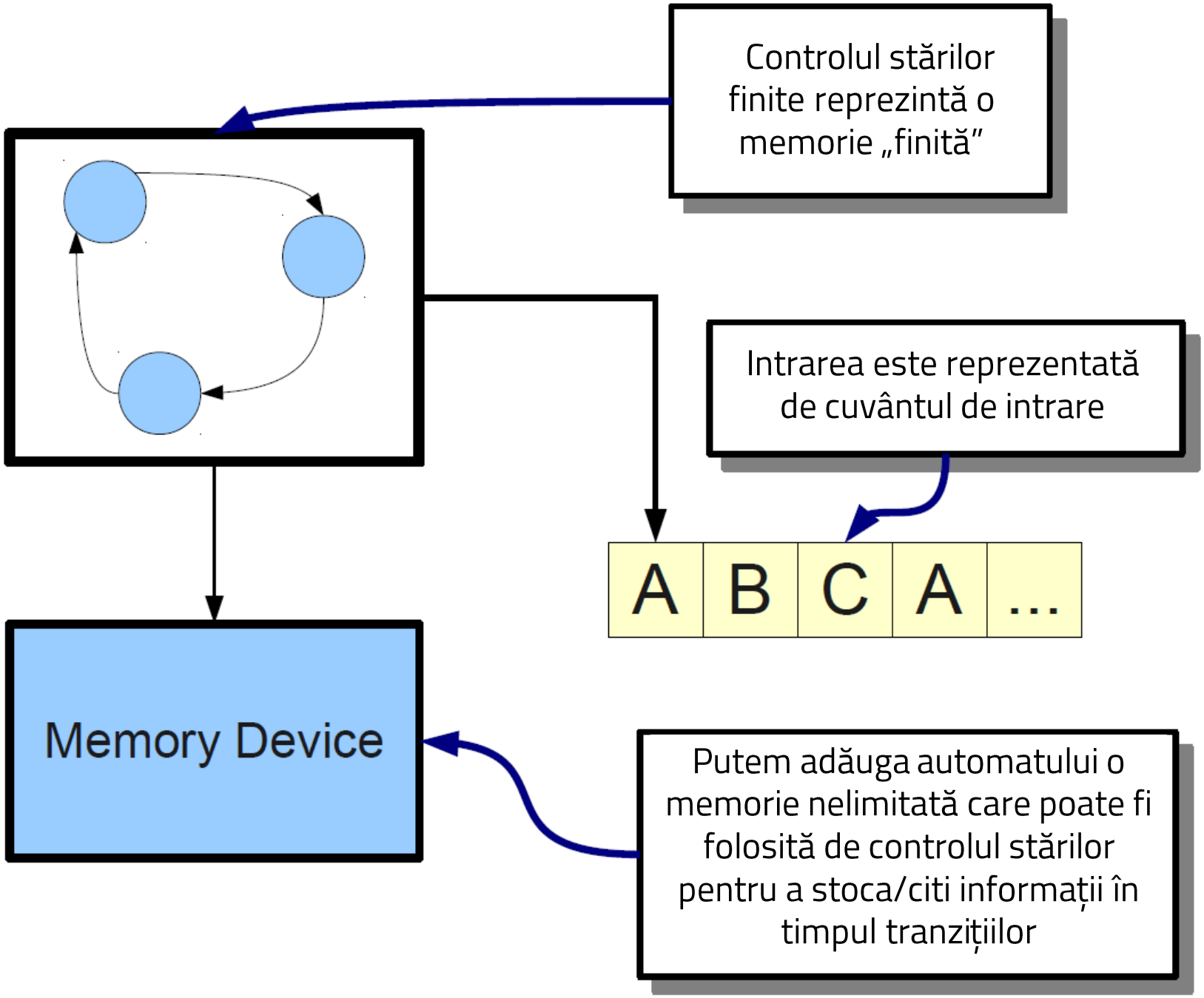
- Limbaje regulate:
  - Expresiile regulate **generează** toate cuvintele unui limbaj
  - Automatele finite **recunosc (acceptă)** doar cuvinte ale unui anumit limbaj
- Limbaje independente de context:
  - **Generate** de gramatici independente de context
  - Există vreun tip de automat care să recunoască (accepte) cuvintele unui limbaj independent de context?

Generarea vs. recunoașterea unui limbaj

- Diferența între gramatici/limbaje regulate vs. independente de context (Chomsky):
  - **Gramatici de tip 2 (independente de context)**
    - reguli de forma $A \to \gamma$ unde $A \in N$ și $\gamma \in (N \cup T)^*$
  - **Gramatici de tip 3 (regulate)**
    - reguli $A \to a$ sau $A \to aB$ unde $A, B \in N$ și $a \in T^*$.
- Datorită regulilor gramaticii (și deci formei cuvintelor limbajului)

$$\text{e.g. } \{\ 0^n 1^n \mid n \in \mathbb{N}\ \}$$

pentru a recunoaște limbaje ***independente de context***, un automat finit are nevoie de o ***memorie asociată „nelimitată”*** (unbounded memory)


Controlul stărilor finite reprezintă o memorie „finită”
Intrarea este reprezentată de cuvântul de intrare
A B C A ...
Memory Device
Putem adăuga automatului o memorie nelimitată care poate fi folosită de controlul stărilor pentru a stoca/citi informații în timpul tranzițiilor

Adăugarea unei memorii automatului

- Tranzițiile automatului vor fi determinate atât de starea curentă și de simbolul de intrare, cât și de informațiile stocate în memorie
- La fiecare tranziție automatul poate efectua operații asupra memoriei: adăugare de informțaii noi, ștergere, modificare, etc.
- Există mai multe arhitecturi de memorie, cea mai simplu de utilizat în acest caz: Stiva

Adăugarea unei memorii automatului

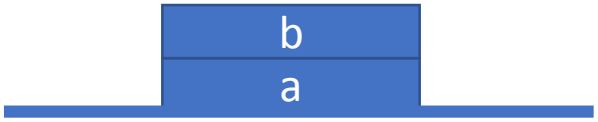
- Tranzițiile automatului vor fi determinate atât de starea curentă și de simbolul de intrare, cât și de informațiile stocate în memorie
- La fiecare tranziție automatul poate efectua operații asupra memoriei: adăugare de informțaii noi, ștergere, modificare, etc.
- Există mai multe arhitecturi de memorie, cea mai simplu de utilizat în acest caz: Stiva

Adăugarea unei memorii automatului

- Tranzițiile automatului vor fi determinate atât de starea curentă și de simbolul de intrare, cât și de informațiile stocate în memorie
- La fiecare tranziție automatul poate efectua operații asupra memoriei: adăugare de informțaii noi, ștergere, modificare, etc.
- Există mai multe arhitecturi de memorie, cea mai simplu de utilizat în acest caz: Stiva

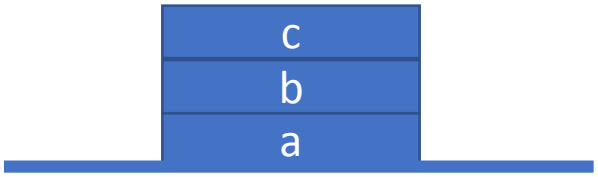
Adăugarea unei memorii automatului

- Tranzițiile automatului vor fi determinate atât de starea curentă și de simbolul de intrare, cât și de informațiile stocate în memorie
- La fiecare tranziție automatul poate efectua operații asupra memoriei: adăugare de informțaii noi, ștergere, modificare, etc.
- Există mai multe arhitecturi de memorie, cea mai simplu de utilizat în acest caz: Stiva



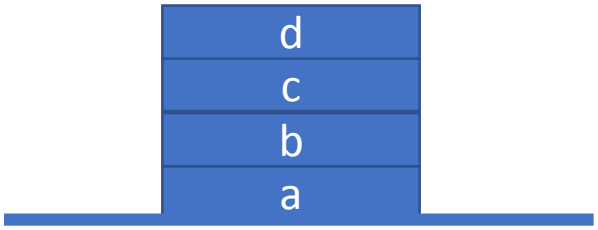
Adăugarea unei memorii automatului

- Tranzițiile automatului vor fi determinate atât de starea curentă și de simbolul de intrare, cât și de informațiile stocate în memorie
- La fiecare tranziție automatul poate efectua operații asupra memoriei: adăugare de informțaii noi, ștergere, modificare, etc.
- Există mai multe arhitecturi de memorie, cea mai simplu de utilizat în acest caz: Stiva



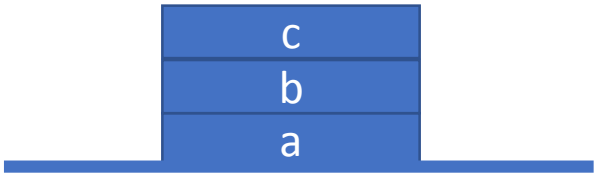
Adăugarea unei memorii automatului

- Tranzițiile automatului vor fi determinate atât de starea curentă și de simbolul de intrare, cât și de informațiile stocate în memorie
- La fiecare tranziție automatul poate efectua operații asupra memoriei: adăugare de informțaii noi, ștergere, modificare, etc.
- Există mai multe arhitecturi de memorie, cea mai simplu de utilizat în acest caz: Stiva



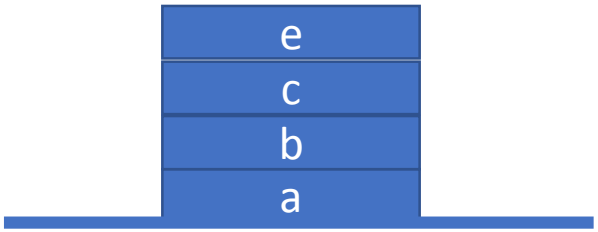
Adăugarea unei memorii automatului

- Tranzițiile automatului vor fi determinate atât de starea curentă și de simbolul de intrare, cât și de informațiile stocate în memorie
- La fiecare tranziție automatul poate efectua operații asupra memoriei: adăugare de informțaii noi, ștergere, modificare, etc.
- Există mai multe arhitecturi de memorie, cea mai simplu de utilizat în acest caz: Stiva



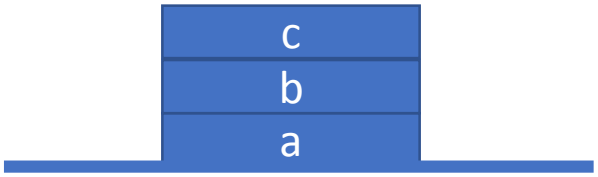
Adăugarea unei memorii automatului

- Tranzițiile automatului vor fi determinate atât de starea curentă și de simbolul de intrare, cât și de informațiile stocate în memorie
- La fiecare tranziție automatul poate efectua operații asupra memoriei: adăugare de informțaii noi, ștergere, modificare, etc.
- Există mai multe arhitecturi de memorie, cea mai simplu de utilizat în acest caz: Stiva



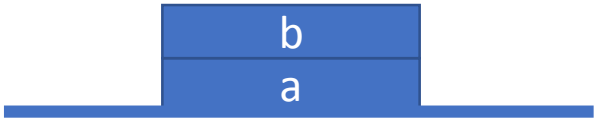
Adăugarea unei memorii automatului

- Tranzițiile automatului vor fi determinate atât de starea curentă și de simbolul de intrare, cât și de informațiile stocate în memorie
- La fiecare tranziție automatul poate efectua operații asupra memoriei: adăugare de informțaii noi, ștergere, modificare, etc.
- Există mai multe arhitecturi de memorie, cea mai simplu de utilizat în acest caz: Stiva



Adăugarea unei memorii automatului

- Tranzițiile automatului vor fi determinate atât de starea curentă și de simbolul de intrare, cât și de informațiile stocate în memorie
- La fiecare tranziție automatul poate efectua operații asupra memoriei: adăugare de informțaii noi, ștergere, modificare, etc.
- Există mai multe arhitecturi de memorie, cea mai simplu de utilizat în acest caz: Stiva



Adăugarea unei memorii automatului

- Tranzițiile automatului vor fi determinate atât de starea curentă și de simbolul de intrare, cât și de informațiile stocate în memorie
- La fiecare tranziție automatul poate efectua operații asupra memoriei: adăugare de informțaii noi, ștergere, modificare, etc.
- Există mai multe arhitecturi de memorie, cea mai simplu de utilizat în acest caz: Stiva

Memoria de tip stivă

- Doar vârful stivei este vizibil la orice moment
- Noi simboluri pot fi adăugate la vârf, printr-o operațiune de ***push***
- Vârful stivei poate fi eliminat printr-o operațiune de ***pop,*** urmând ca elementul „de sub” el să devină noul vârf

Automatul de tip push-down

- Automatul de tip push-down (PDA – Push-down automata) este un automat finit de stări echipat cu o memorie de tip stivă
- Fiecare tranziție:
  - Este determinată de simbolul de intrare curent, de starea curentă și de simbolul de la vârful stivei
  - Poate, opțional, să realizeze operațiunea de ***pop*** pe elementul din vârful stivei
  - Poate, opțional, să realizeze operațiunea de ***push*** a unui nou simbol în vârful stivei
- Inițial, stiva conține un simbol special $\mathbf{Z_0}$ care indică faptul că stiva este goală

Un exemplu de PDA

- Considerăm limbajul $L=\{w \in \Sigma^* \mid w$ reprezintă un șir de caractere cu număr egal de simboluri 0 și 1$\}$ peste alfabetul $\Sigma=\{0,1\}$
- În acest caz, funcționarea stivei automatului ar fi următoarea:
  - La întâlnirea unui simbol 0, acesta este introdus prin push în stivă
  - La întâlnirea unui simbol 1, se realizează pop pentru simbolul 0 aflat în vârful stivei (sau eroare dacă stiva este goală)
  - Când au fost parcurse toate simbolurile din cuvântul de intrare, dacă stiva este goală atunci cuvântul este acceptat

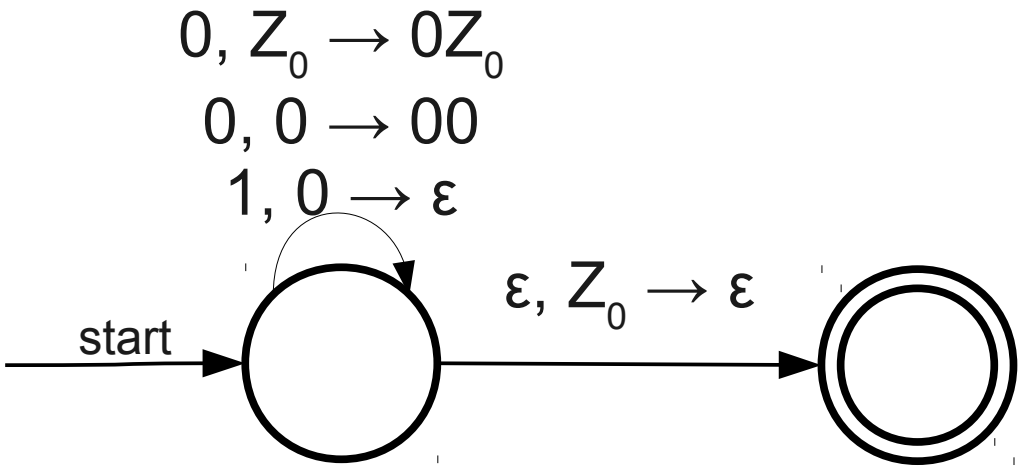
Automatele push-down

- Formal, un automat push-down este definit ca un ansamblu de 7 elemente $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ unde:
  - Q este o mulțime finită de stări
  - Σ este un alfabet
  - Γ este alfabetul de simboluri al stivei
  - δ: $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow \mathcal{P}(Q \times \Gamma^*)$ este funcția de tranziție
  - q_0 este starea inițială
  - $Z_0 \in \Gamma$ este simbolul inițial al stivei
  - $F \subseteq Q$ este mulțimea stărilor acceptoare
- Automatul acceptă un cuvânt dacă la parcurgerea acestuia se va regăsi într-o stare acceptoare

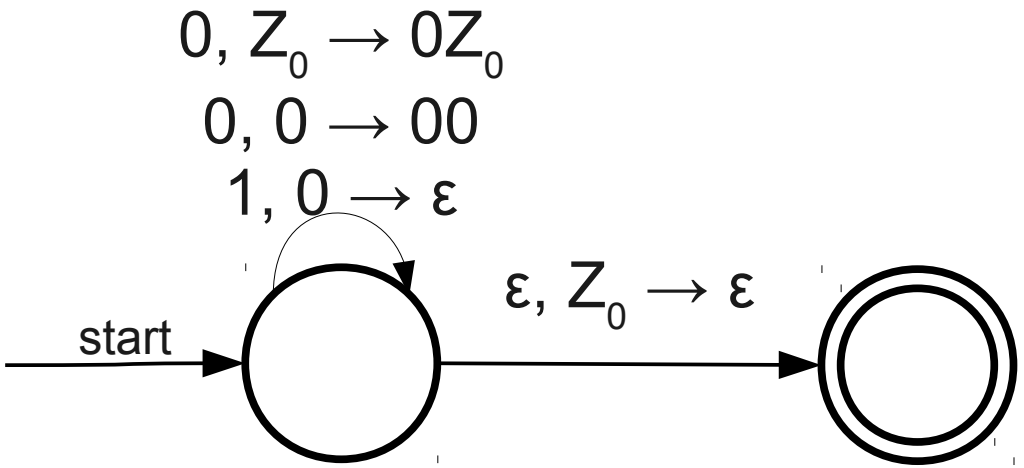
Limbajul unui automat push-down

- Limbajul unui automat push-down este mulțimea cuvintelor acceptate de acel automat:
  - $\mathcal{L}(P) = \{ w \in \Sigma^* \mid P \text{ acceptă } w\}$
- Dacă P este un automat push-down și $\mathcal{L}(P)=L$, atunci se afirmă faptul că P recunoaște L

A Simple Pushdown Automaton

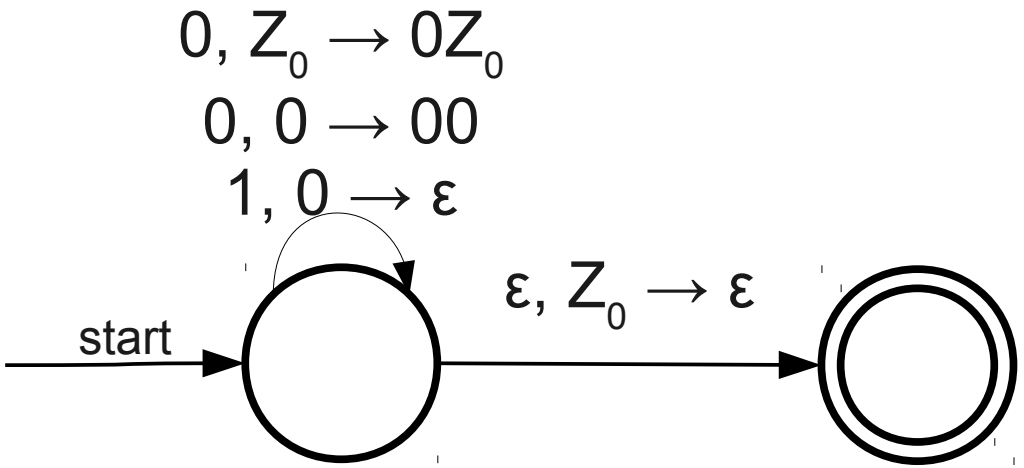


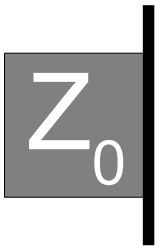
A Simple Pushdown Automaton



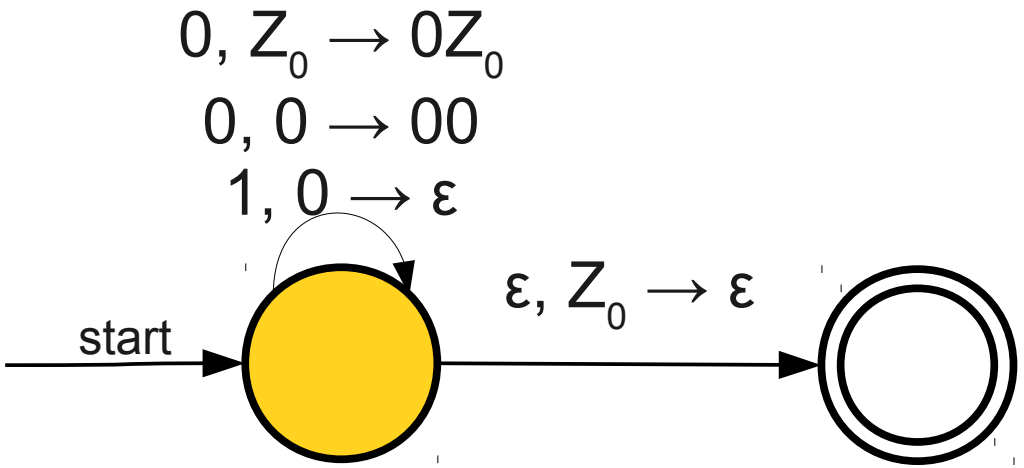
0 0 0 1 1 1

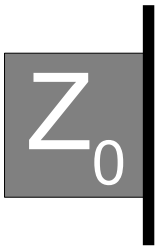
A Simple Pushdown Automaton



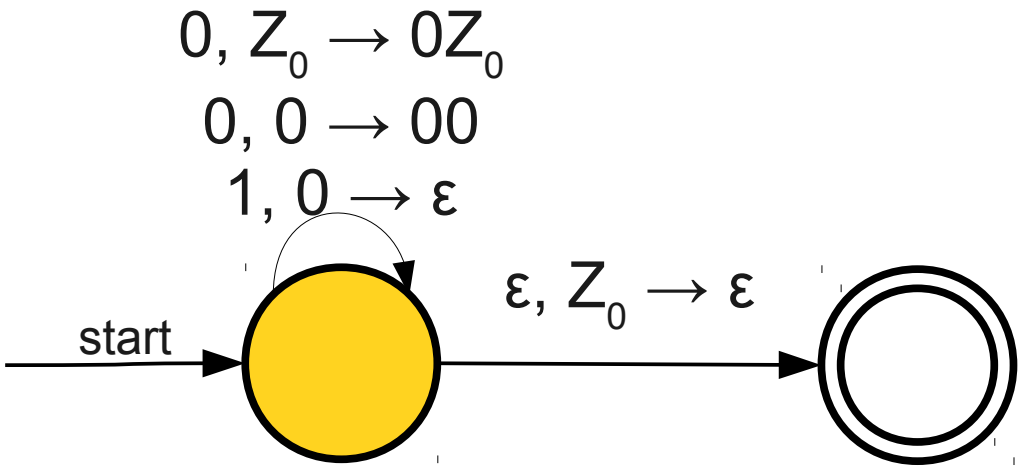


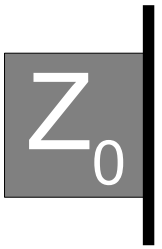
A Simple Pushdown Automaton



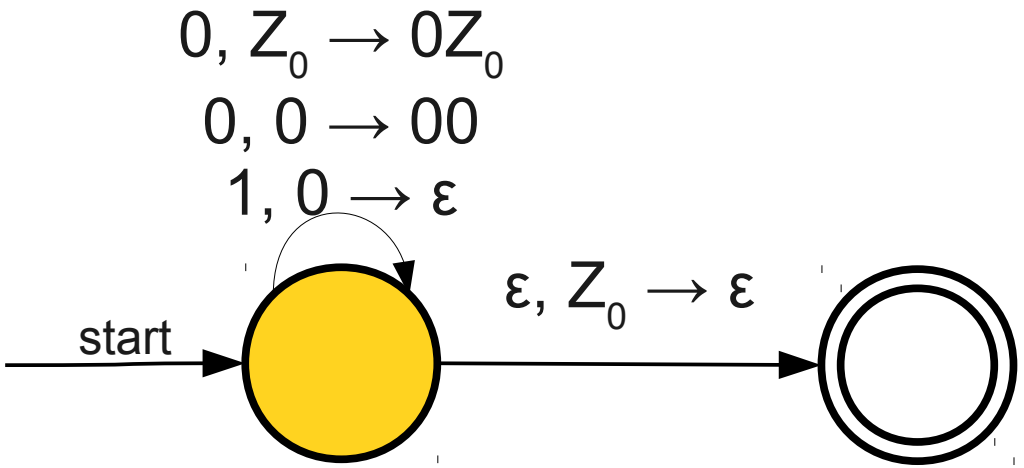


A Simple Pushdown Automaton

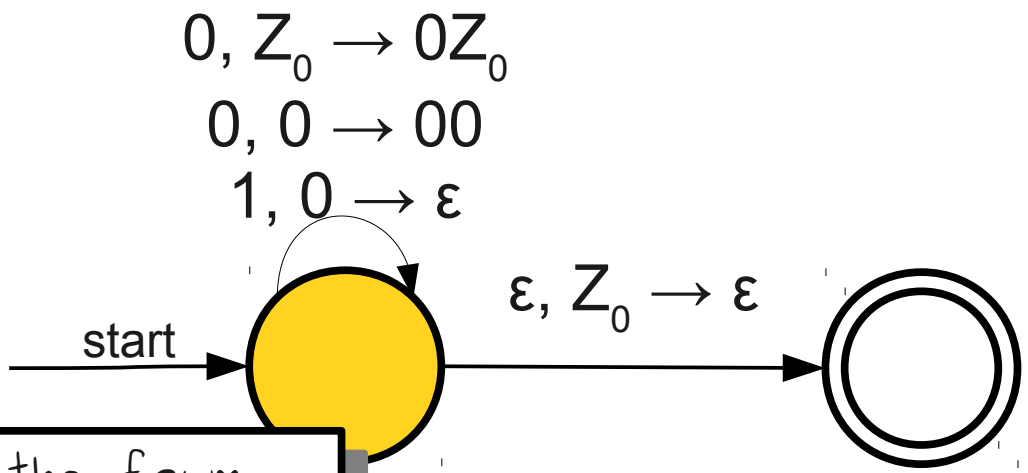




A Simple Pushdown Automaton



A Simple Pushdown Automaton

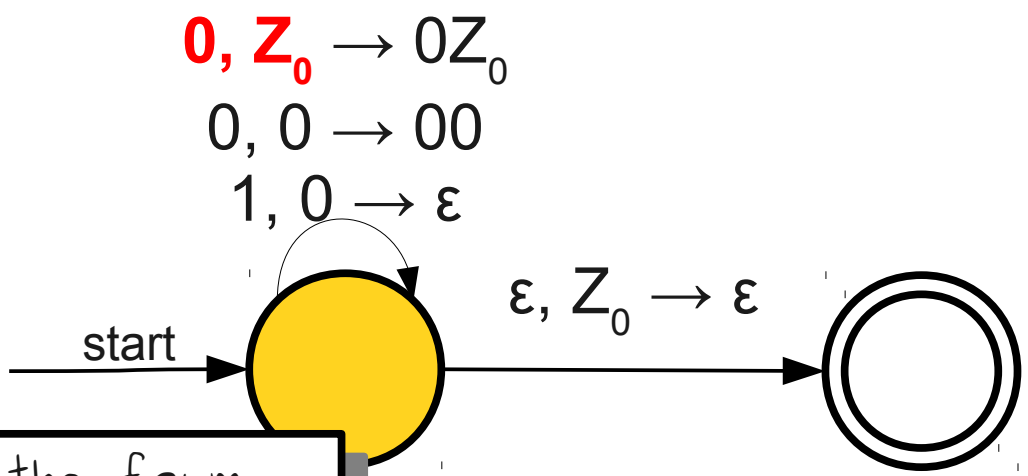


A transition of the form

a, b → z

Means "If the current input symbol is a and the current stack symbol is b, then follow this transition, pop b, and push the string z.

A Simple Pushdown Automaton



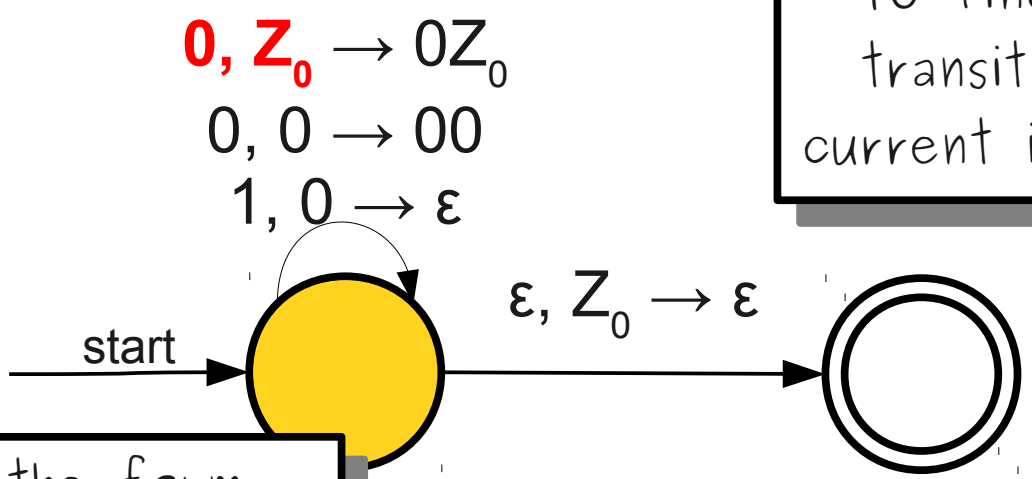
A transition of the form

a, b → z

Means "If the current input symbol is a and the current stack symbol is b, then follow this transition, pop b, and push the string z.

A Simple Pushdown Automaton

To find an applicable transition, match the current input/stack pair.

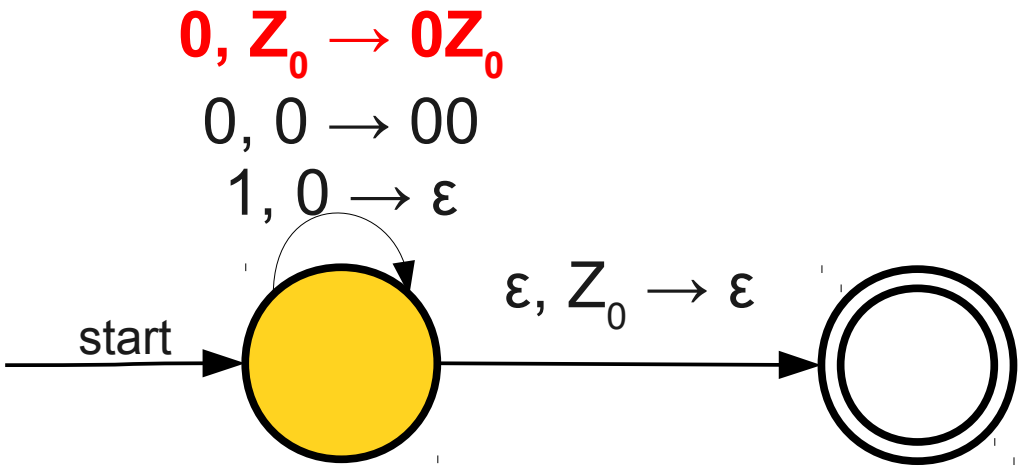


A transition of the form

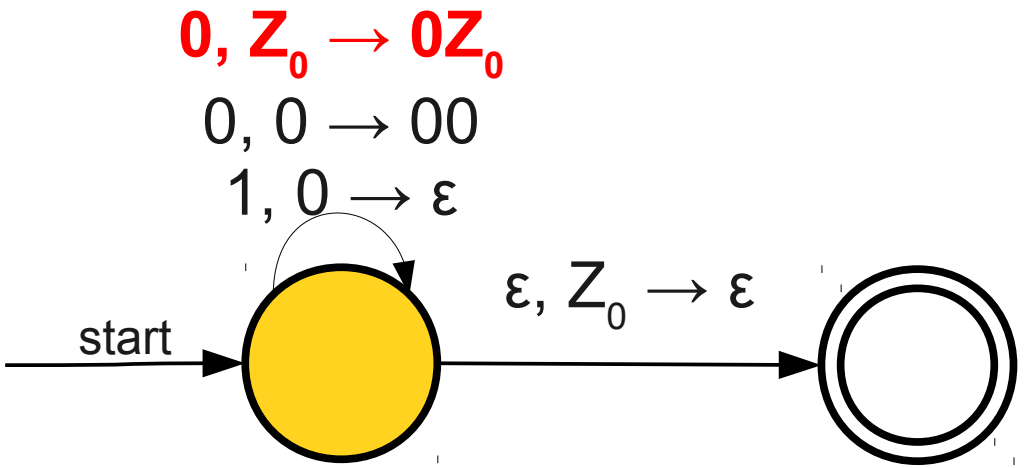
a, b → z

Means "If the current input symbol is a and the current stack symbol is b, then follow this transition, pop b, and push the string z.

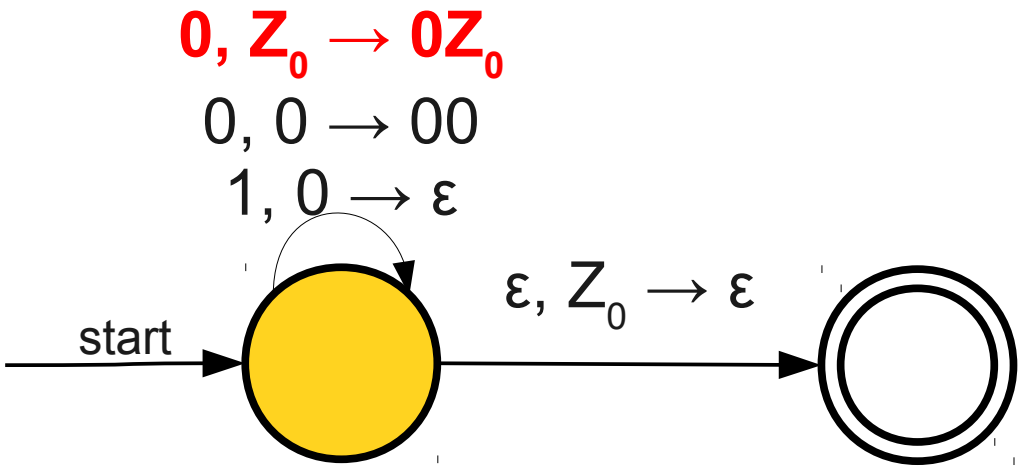
A Simple Pushdown Automaton



A Simple Pushdown Automaton

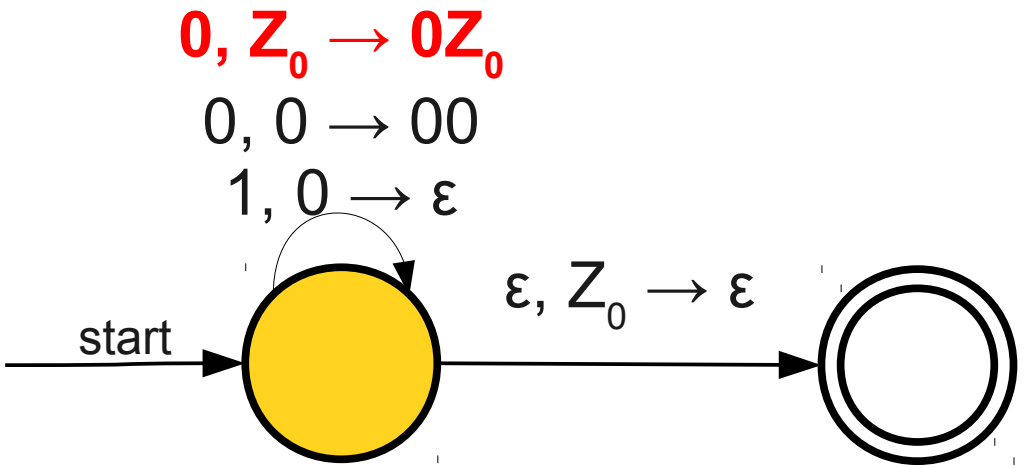


A Simple Pushdown Automaton

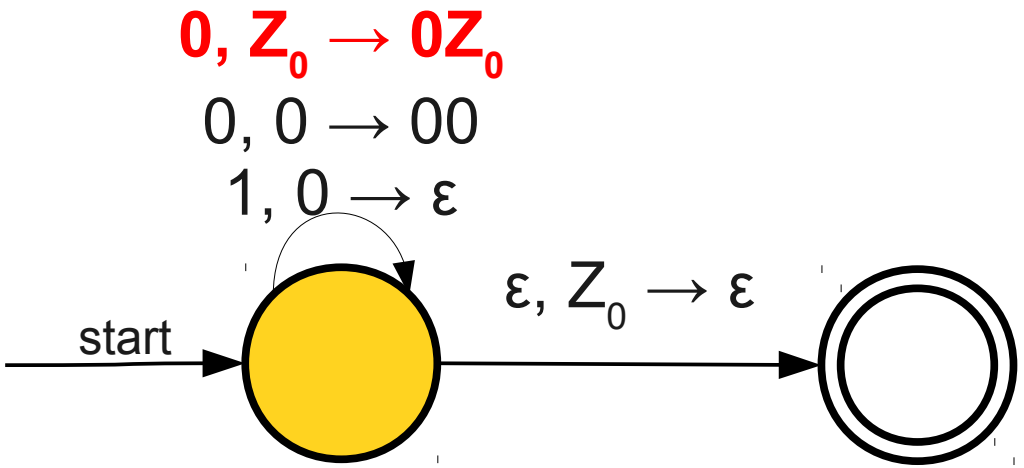


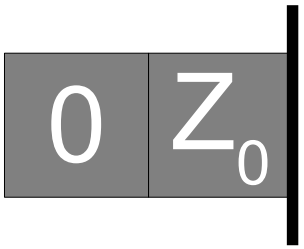
If a transition reads the top symbol of the stack, it always pops that symbol (though it might replace it)

A Simple Pushdown Automaton

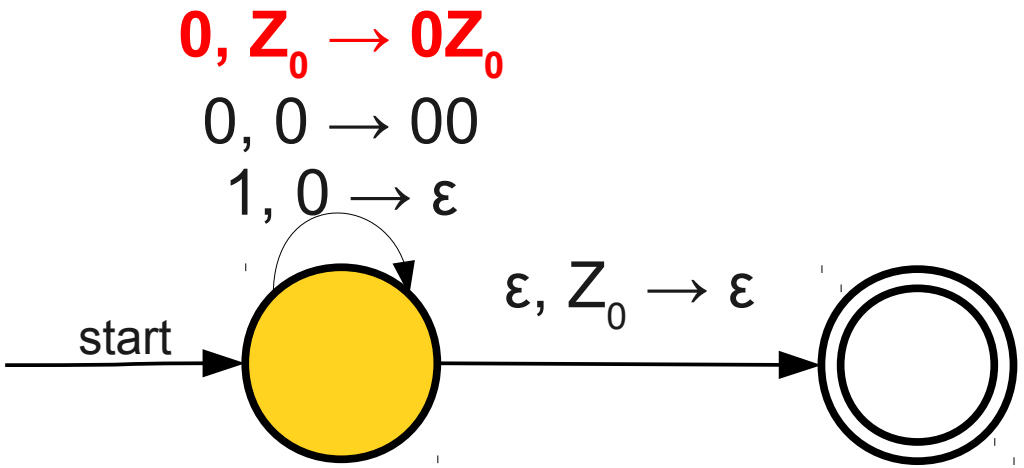


A Simple Pushdown Automaton

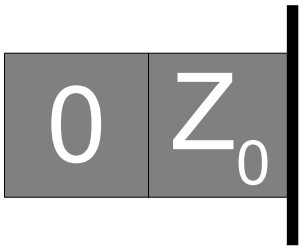




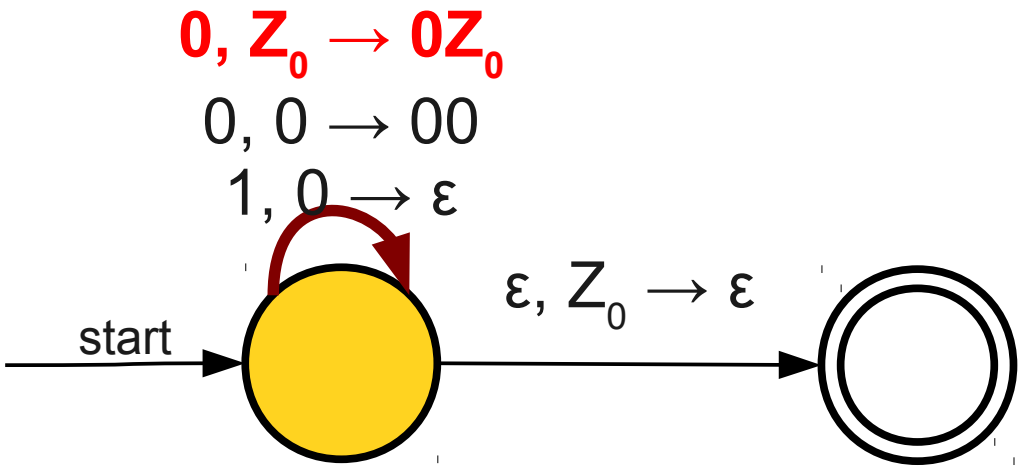
A Simple Pushdown Automaton

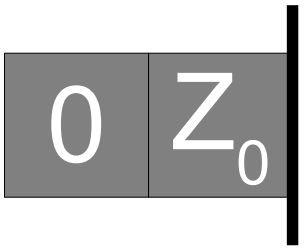


Each transition then pushes some (possibly empty) string back onto the stack. Notice that the leftmost symbol is pushed onto the top.

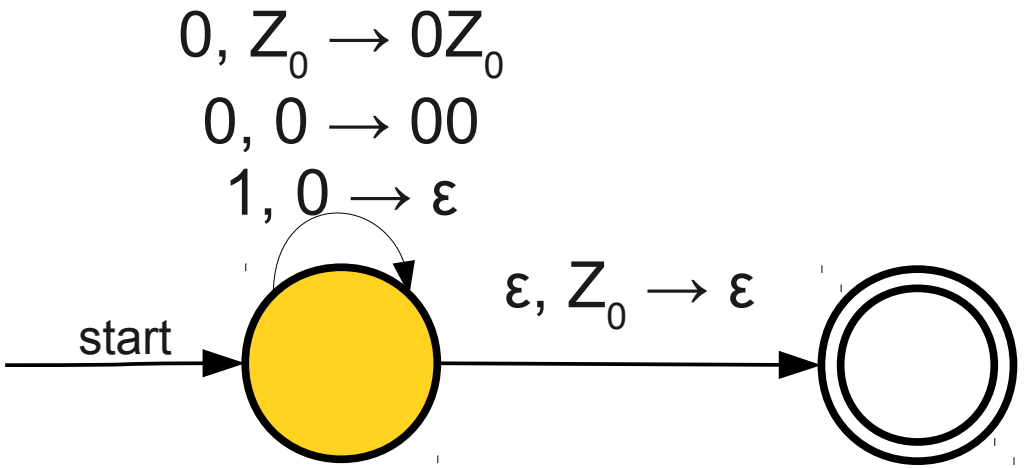


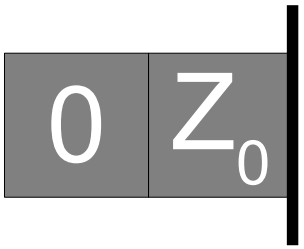
A Simple Pushdown Automaton



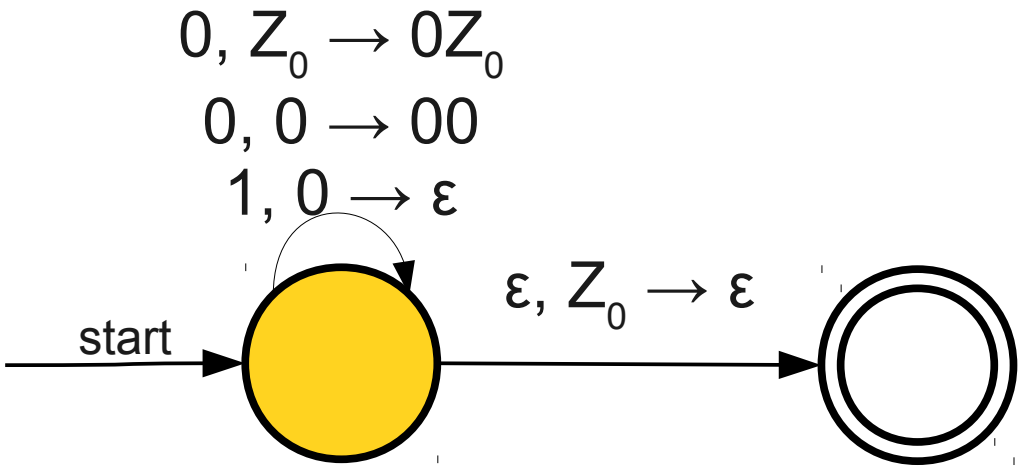


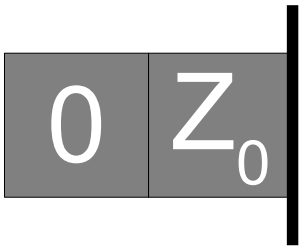
A Simple Pushdown Automaton



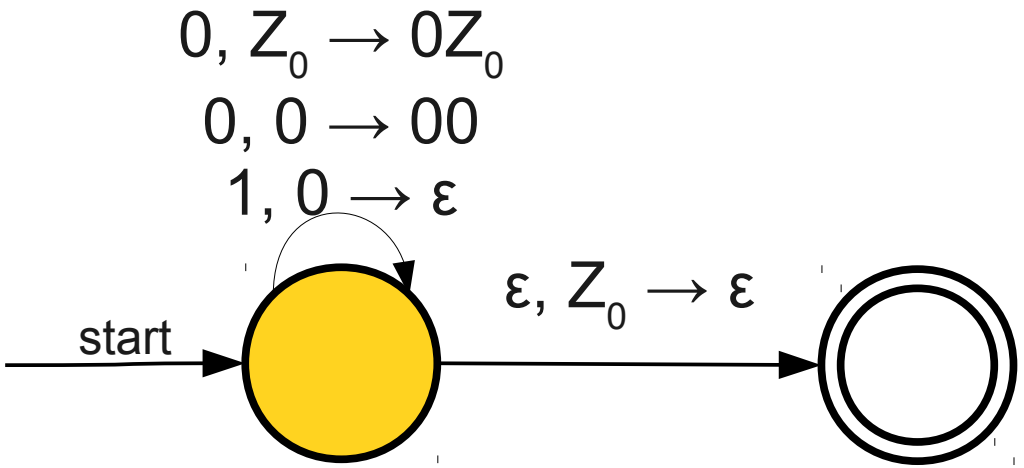


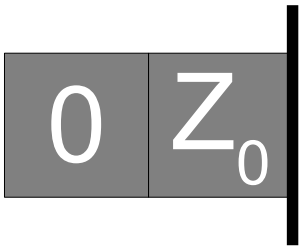
A Simple Pushdown Automaton



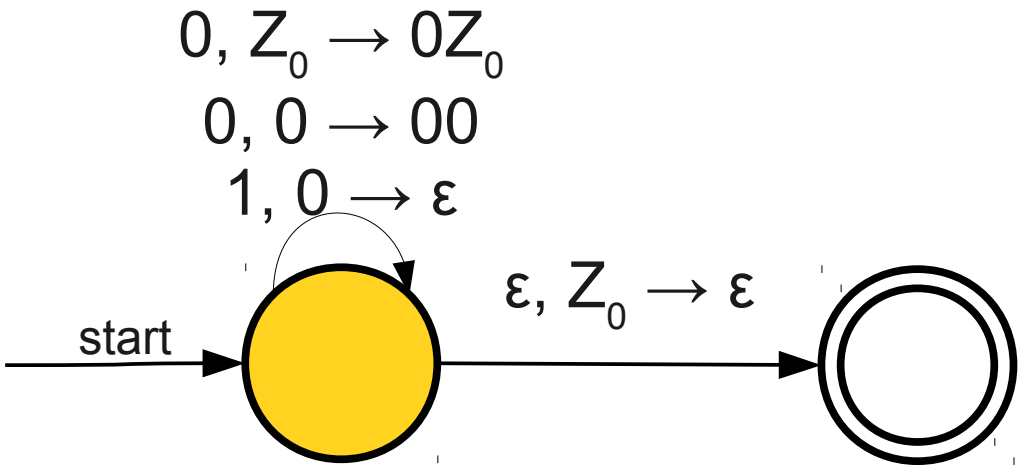


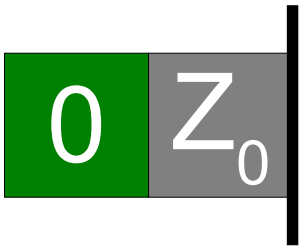
A Simple Pushdown Automaton



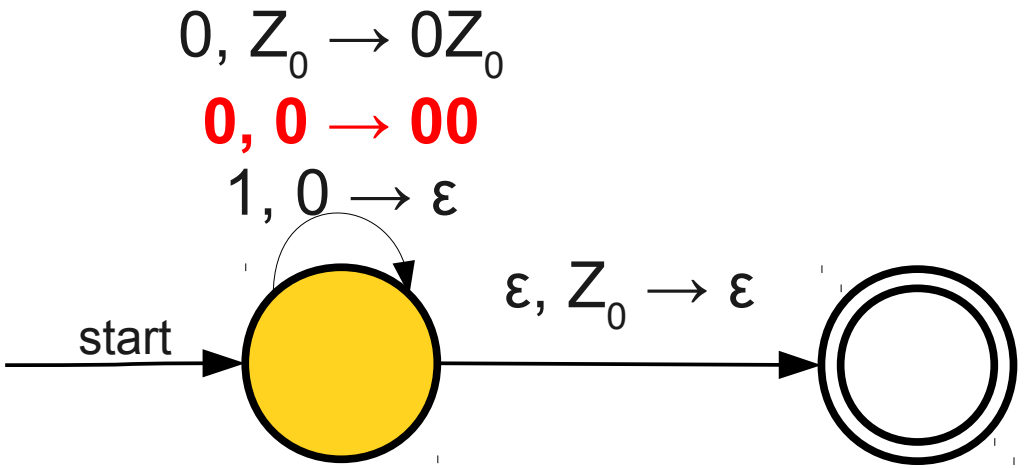


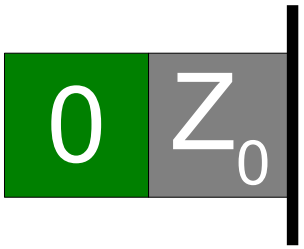
A Simple Pushdown Automaton



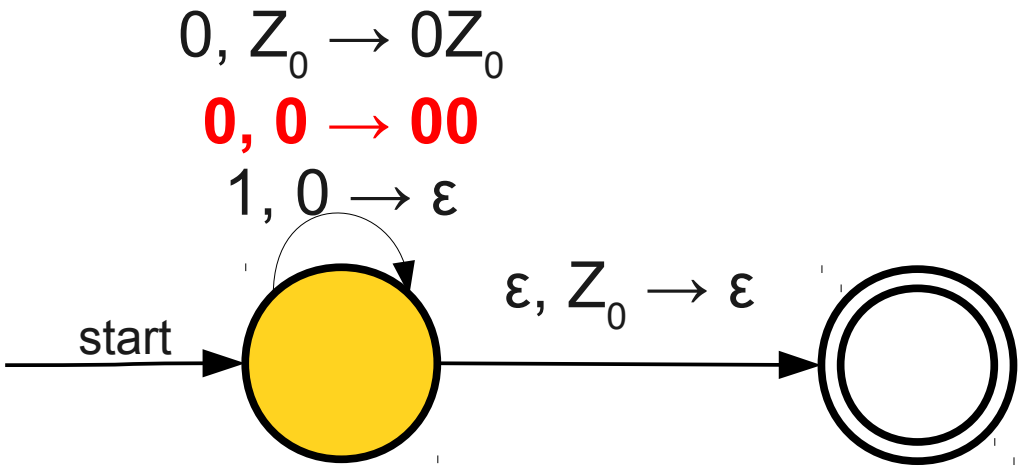


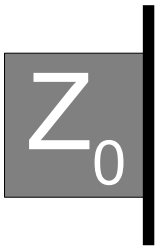
A Simple Pushdown Automaton



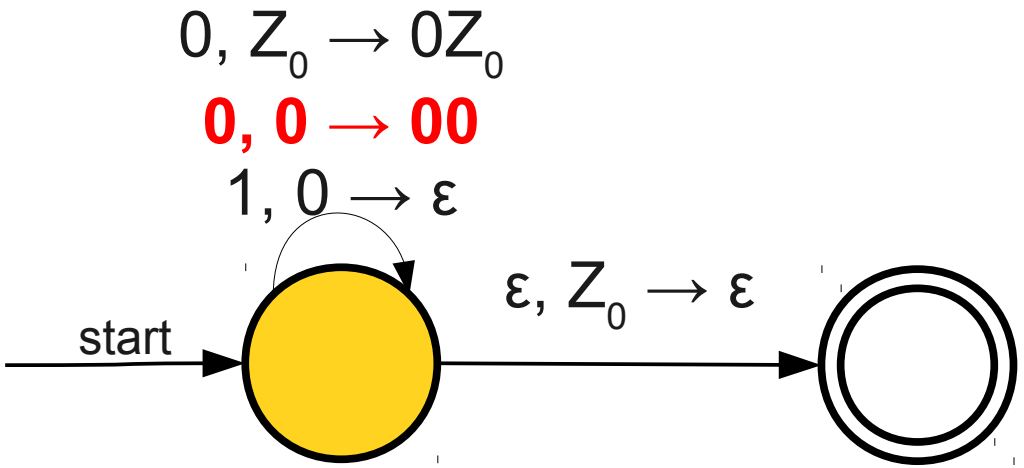


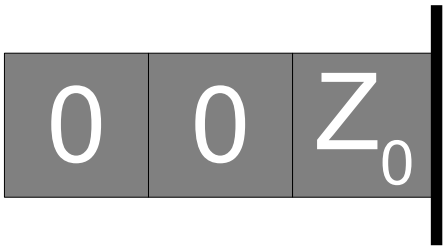
A Simple Pushdown Automaton



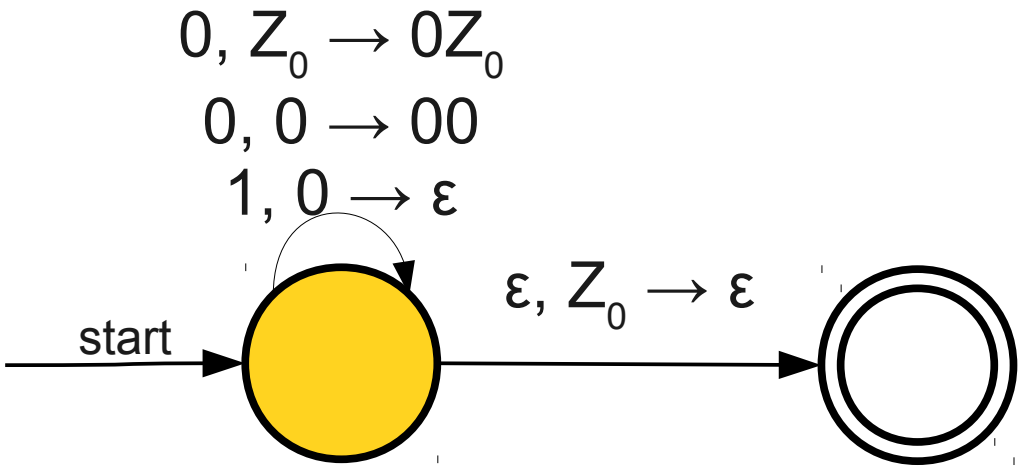


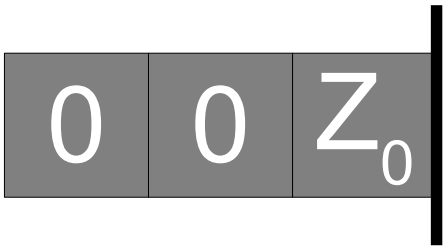
A Simple Pushdown Automaton

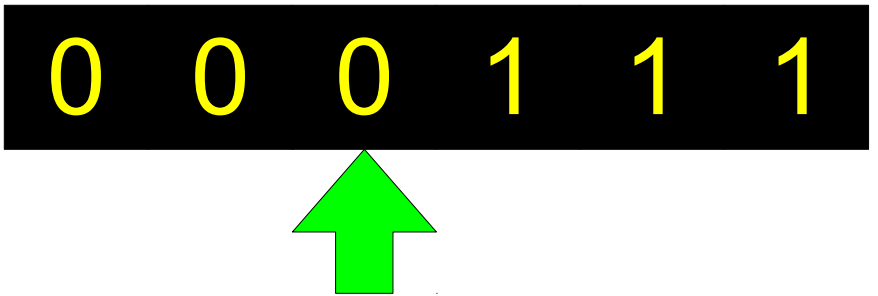




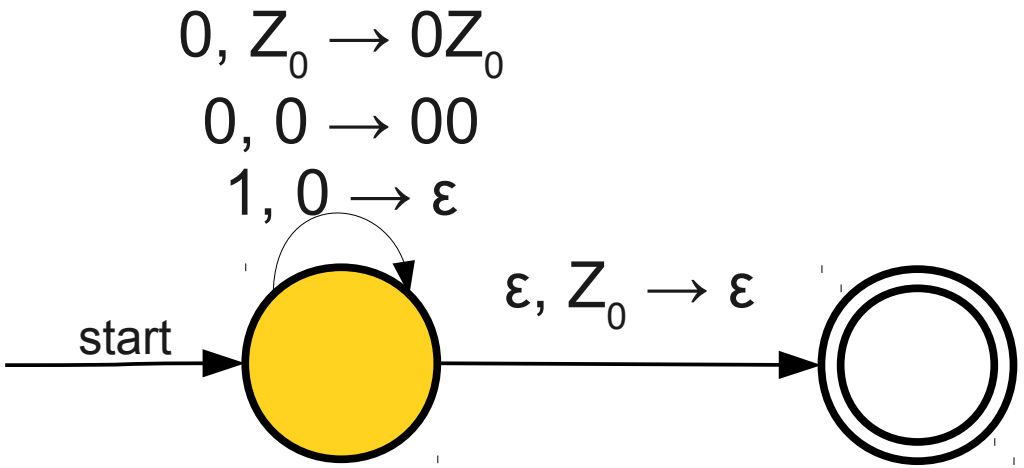
A Simple Pushdown Automaton

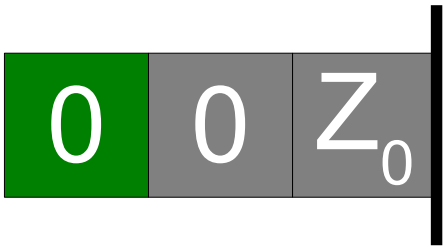


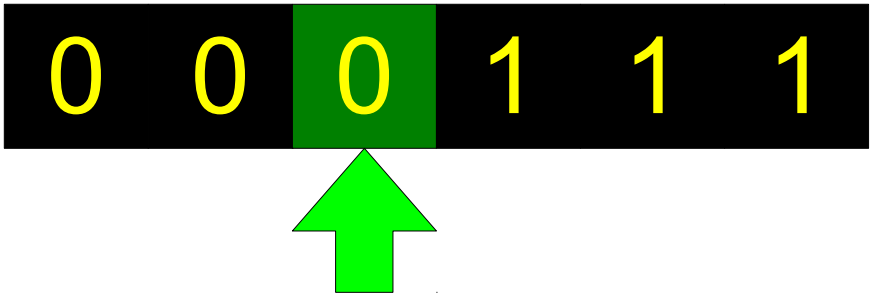




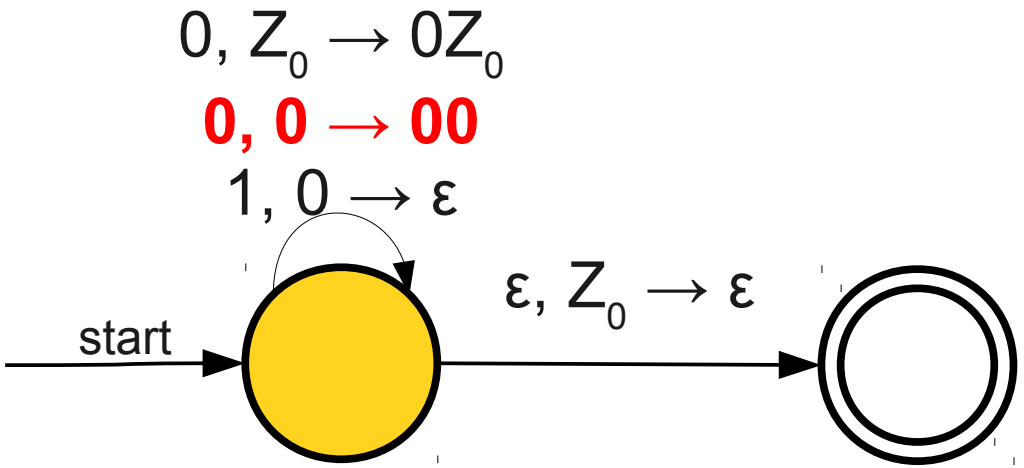
A Simple Pushdown Automaton

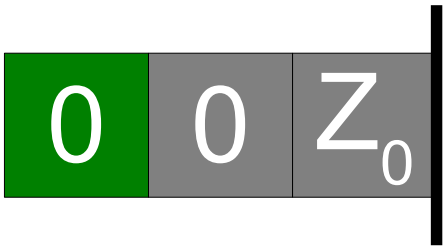


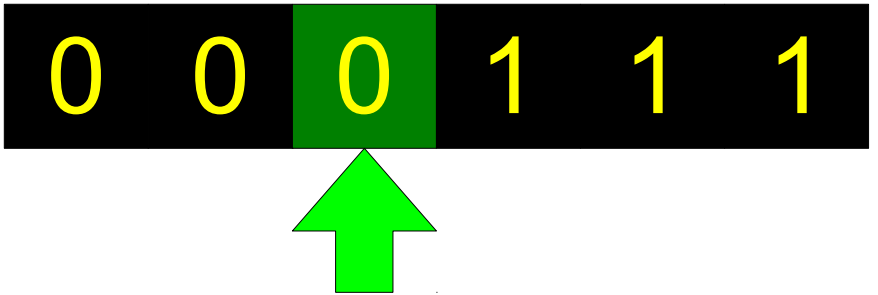




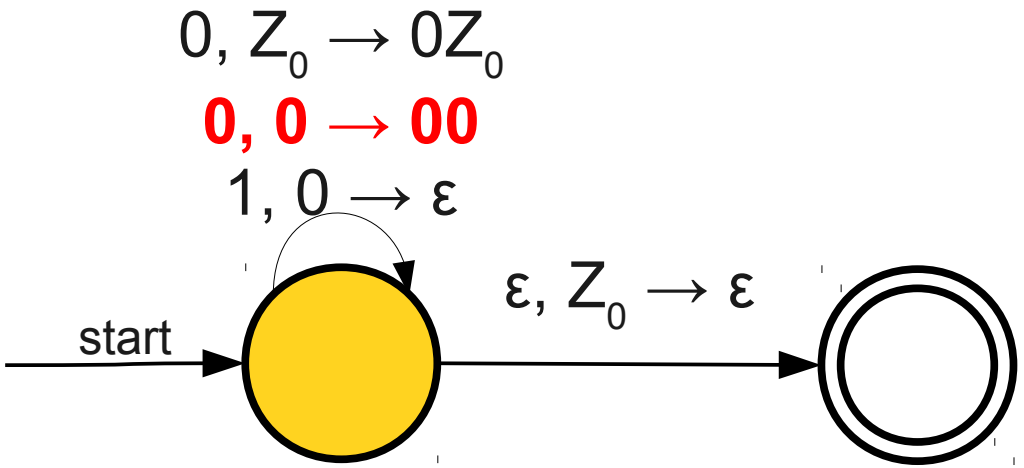
A Simple Pushdown Automaton

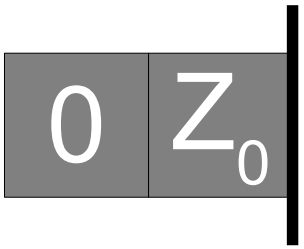


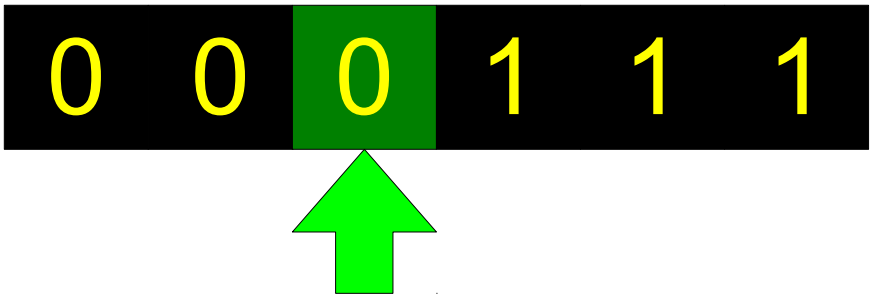




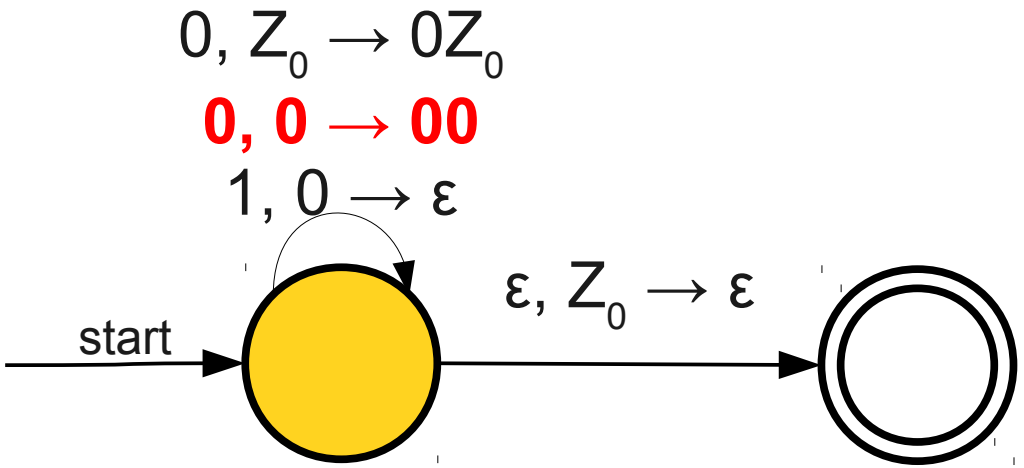
A Simple Pushdown Automaton

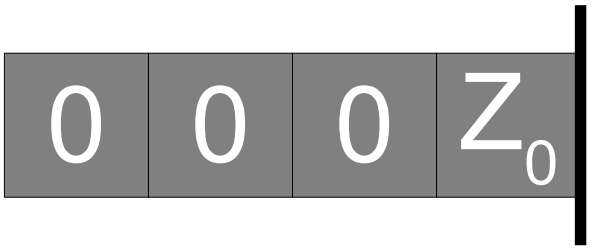


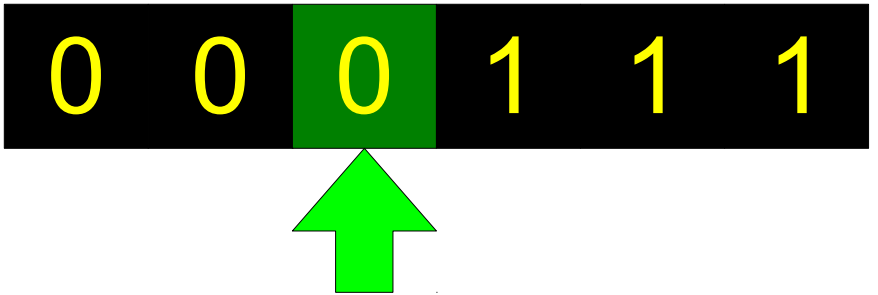




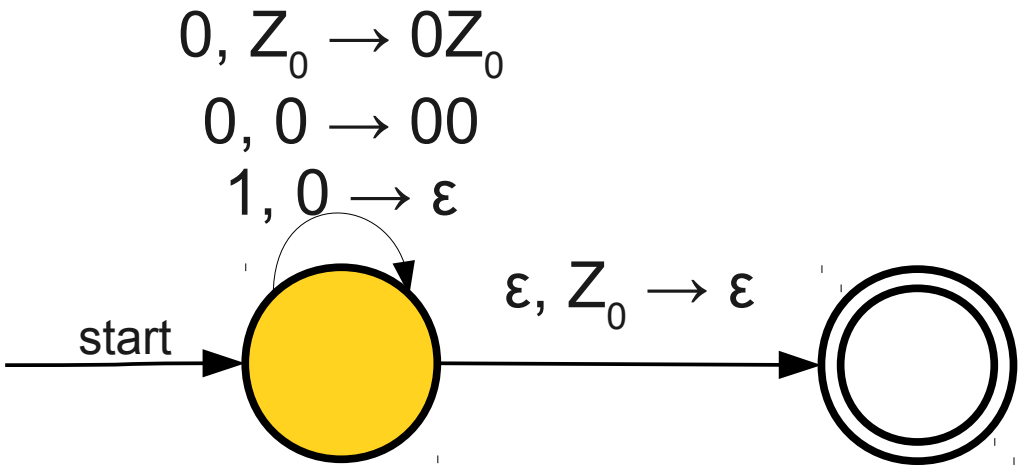
A Simple Pushdown Automaton

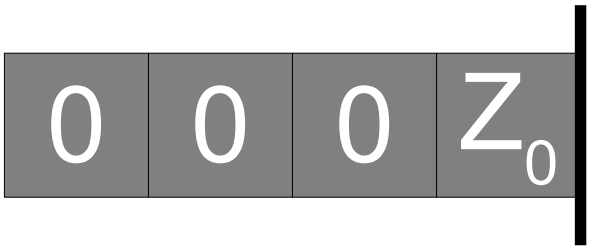


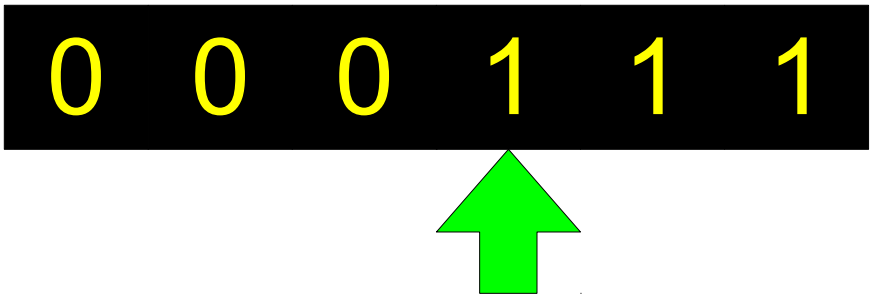




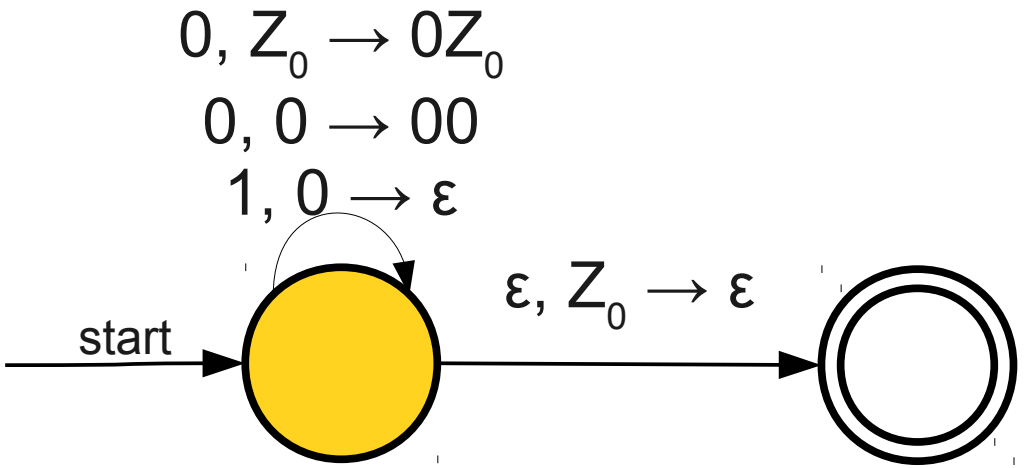
A Simple Pushdown Automaton

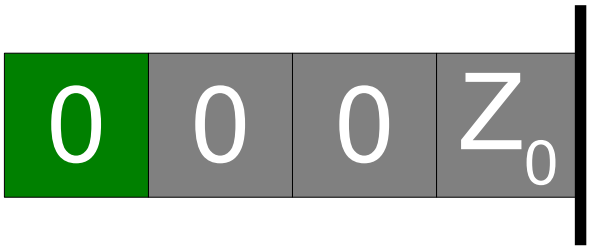


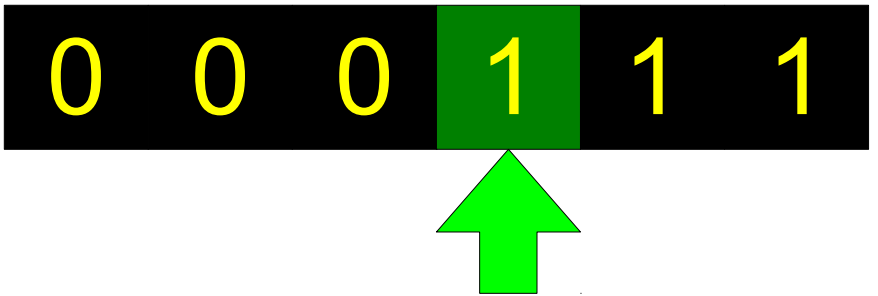




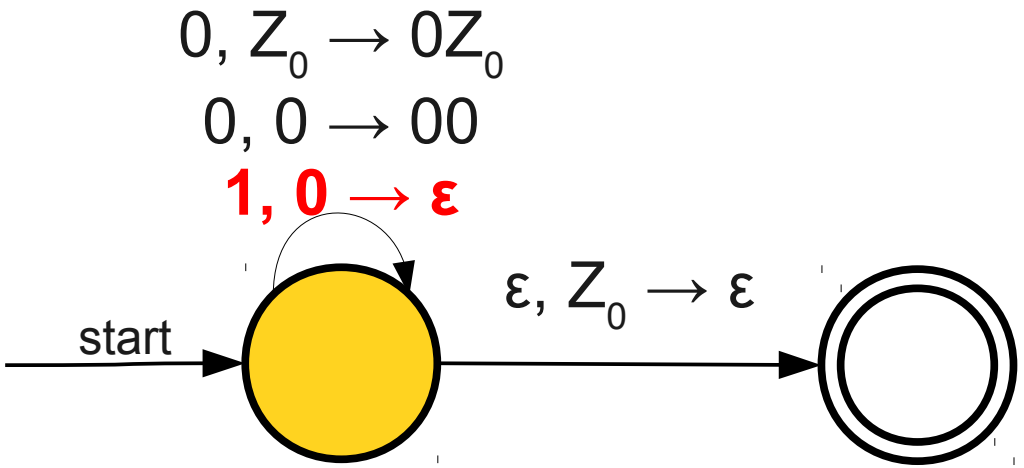
A Simple Pushdown Automaton

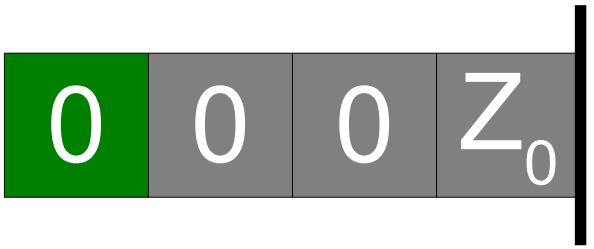


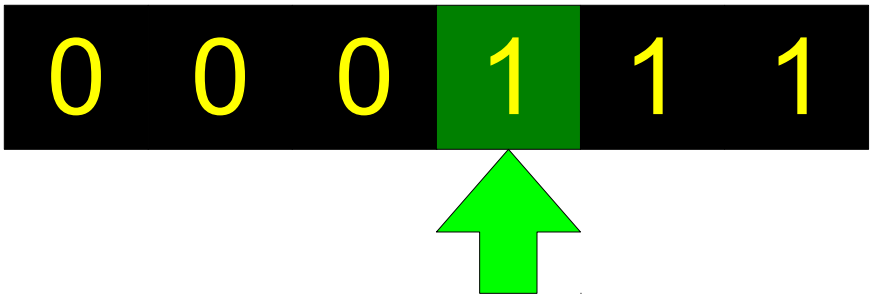




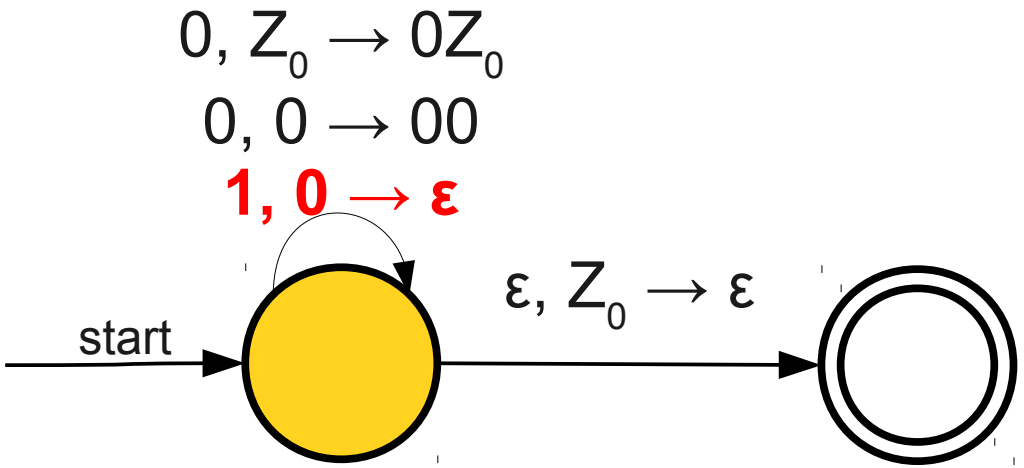
A Simple Pushdown Automaton

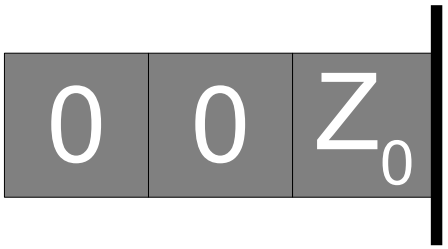


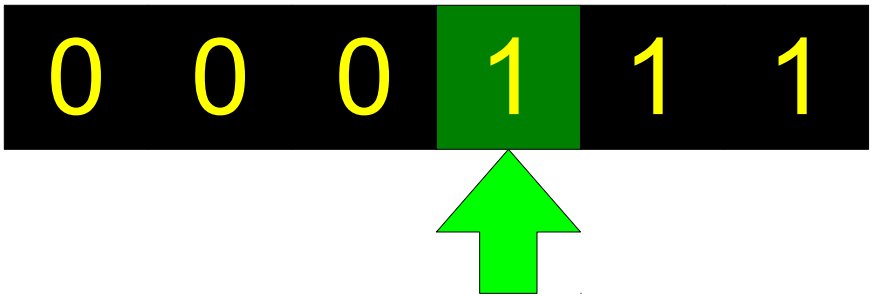




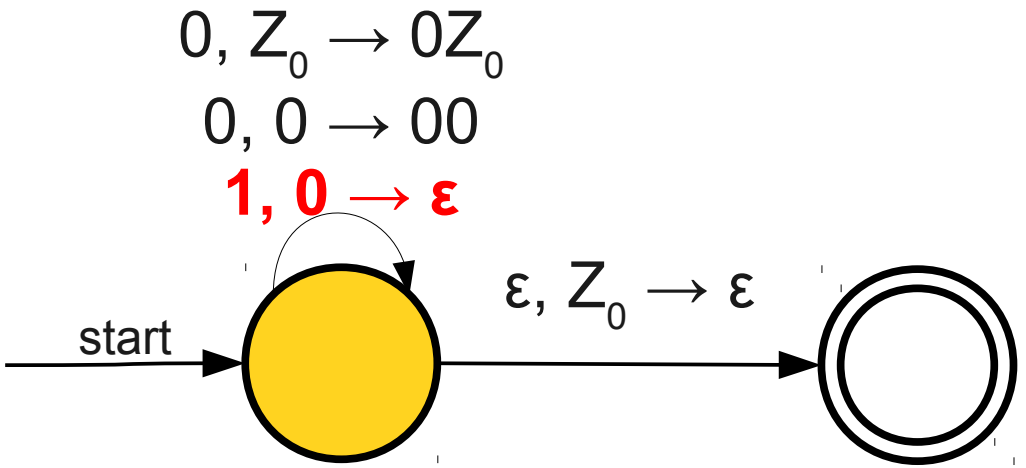
A Simple Pushdown Automaton



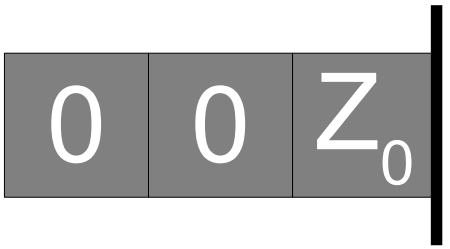


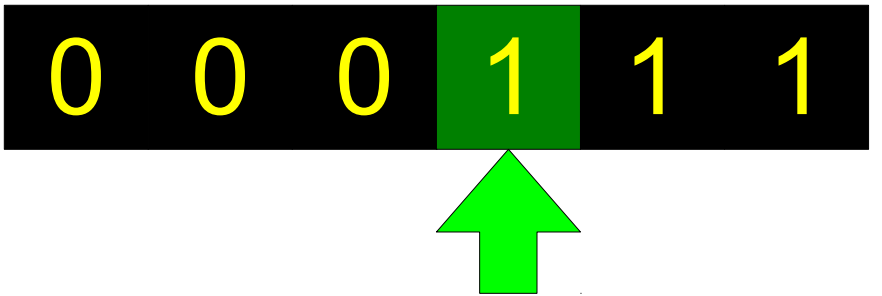


A Simple Pushdown Automaton

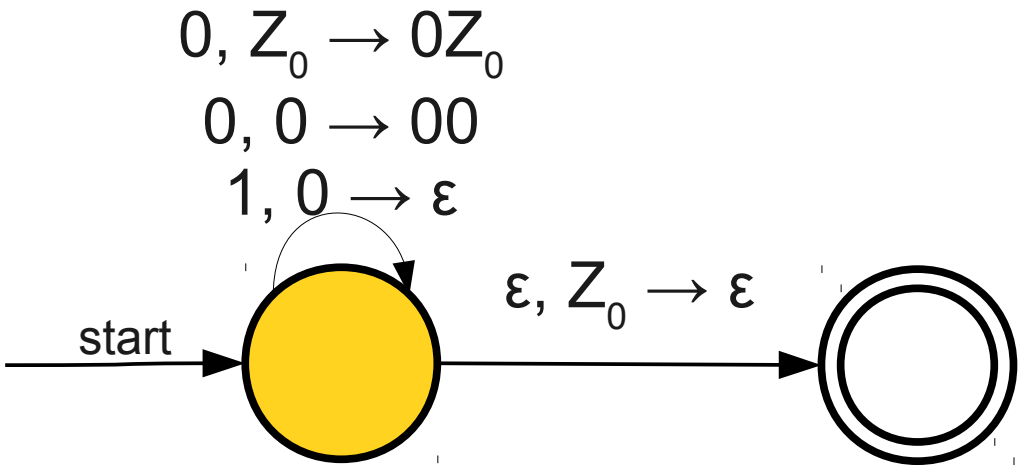


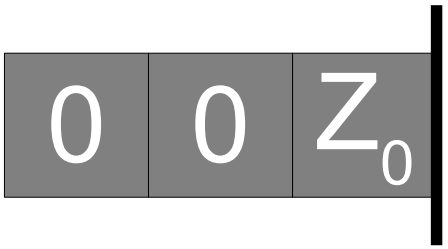
We now push the string ε onto the stack, which adds no new characters. This essentially means "pop the stack."



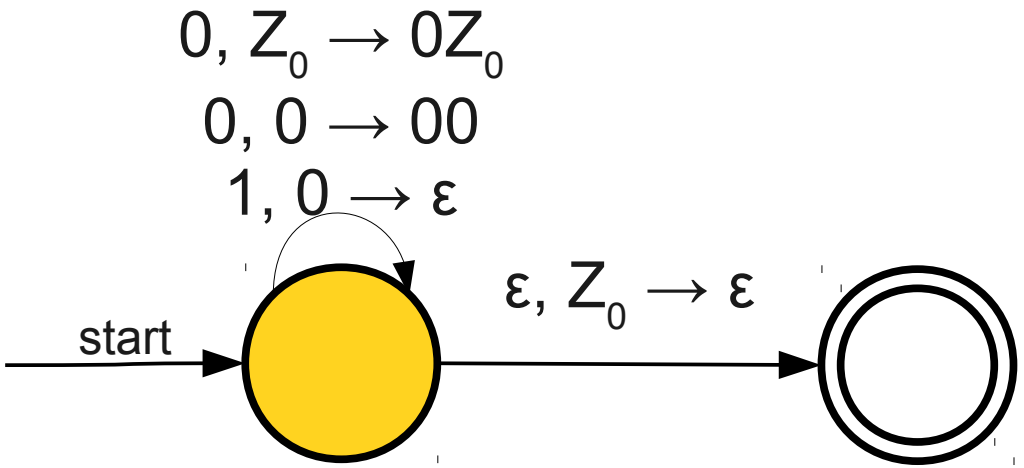


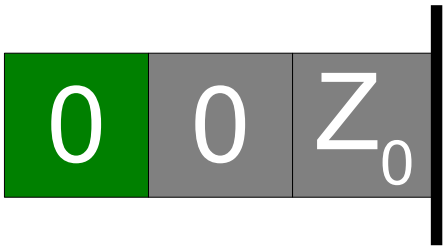
A Simple Pushdown Automaton



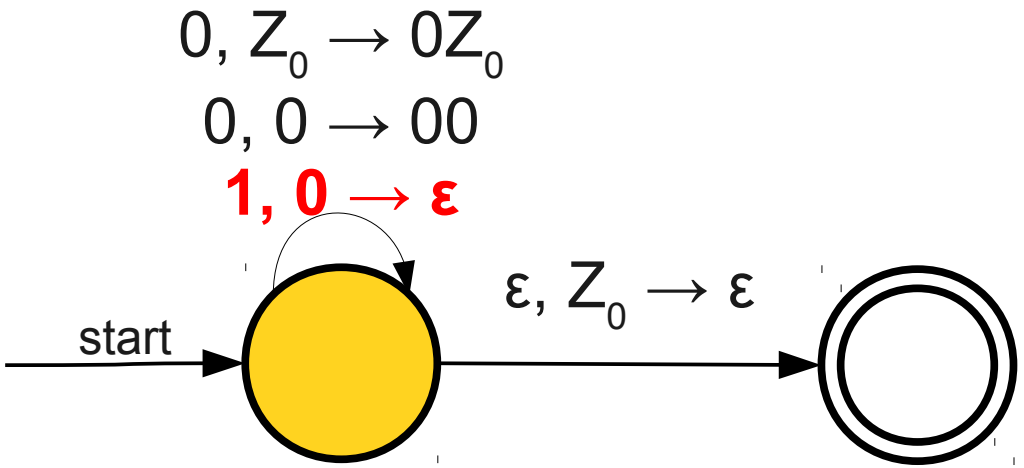


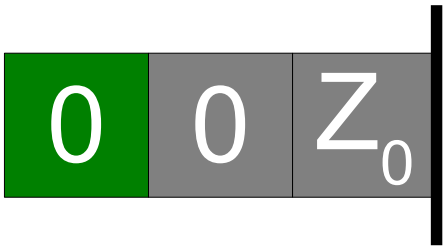
A Simple Pushdown Automaton



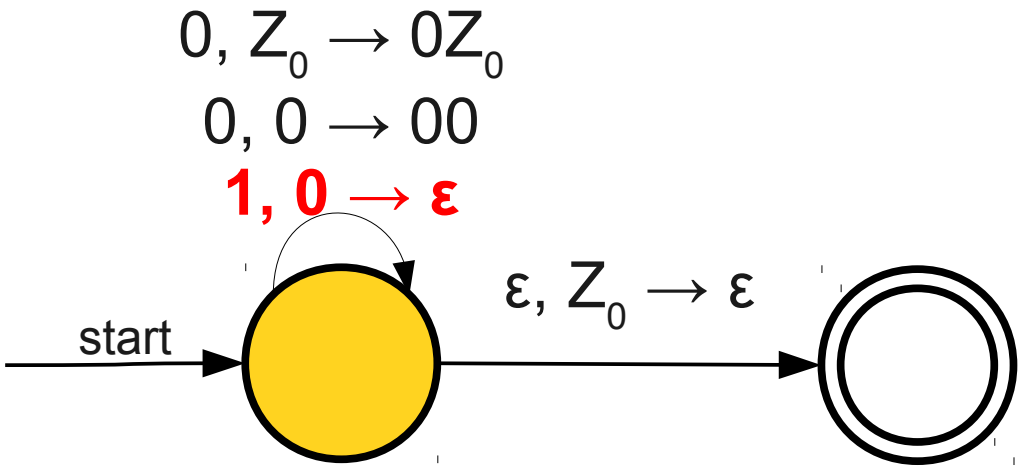


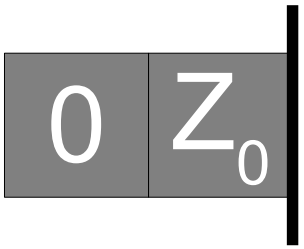
A Simple Pushdown Automaton



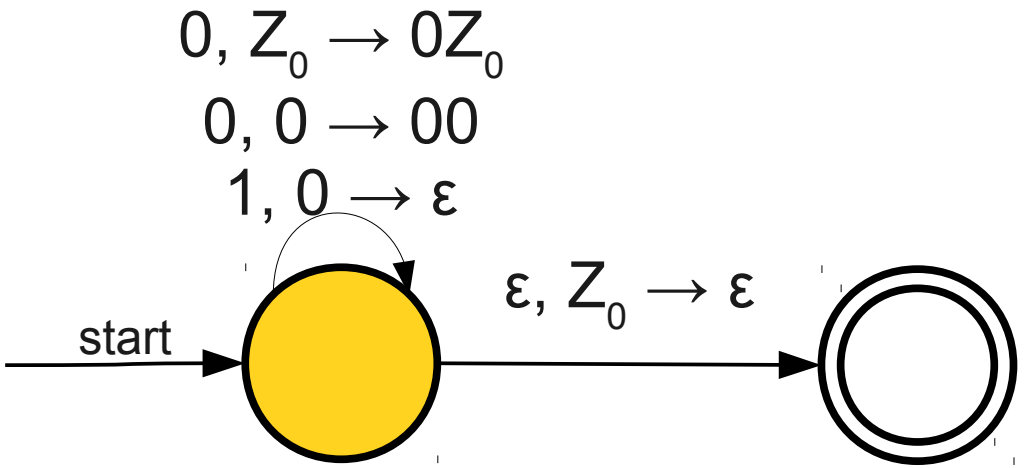


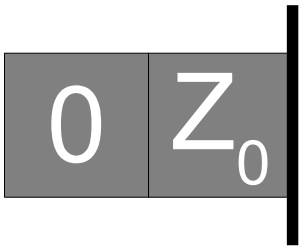
A Simple Pushdown Automaton



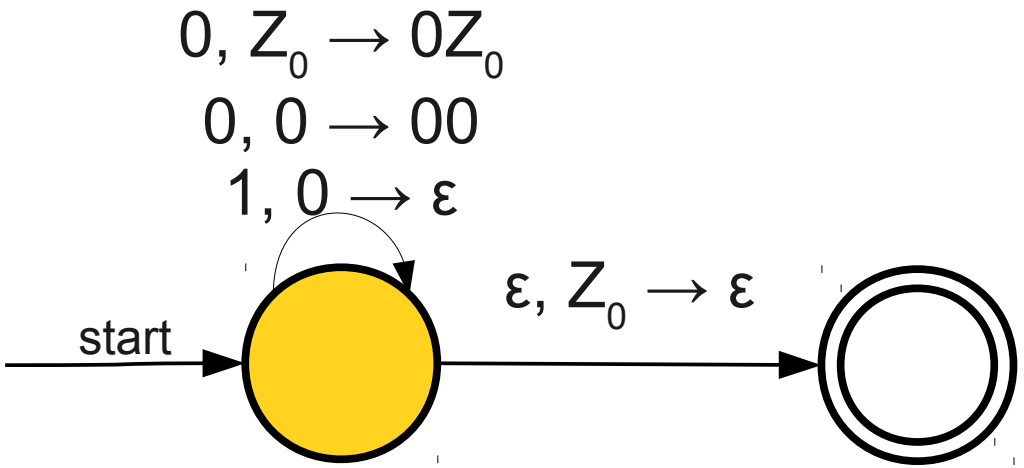


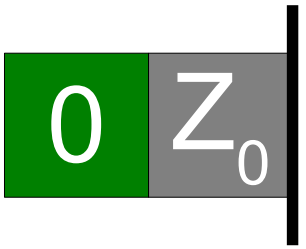
A Simple Pushdown Automaton



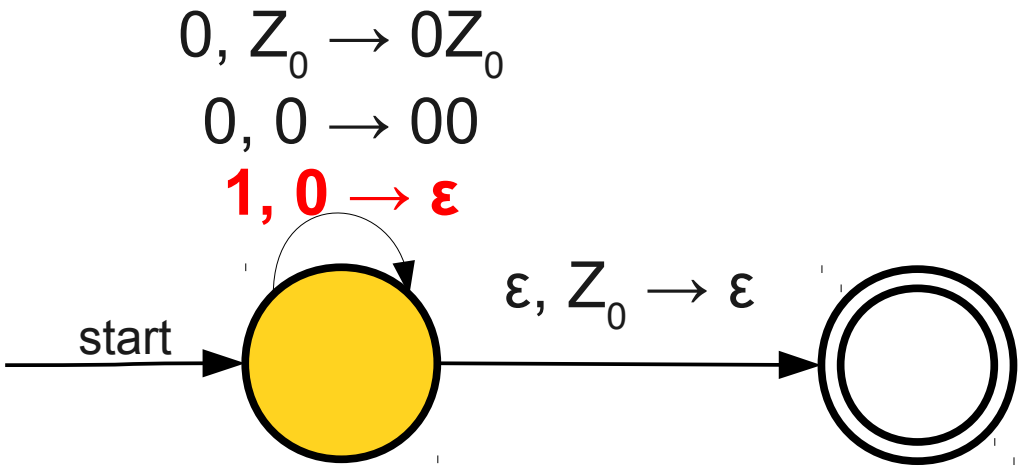


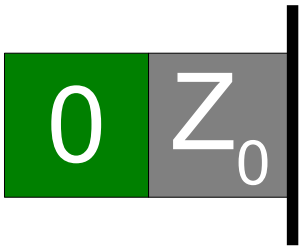
A Simple Pushdown Automaton



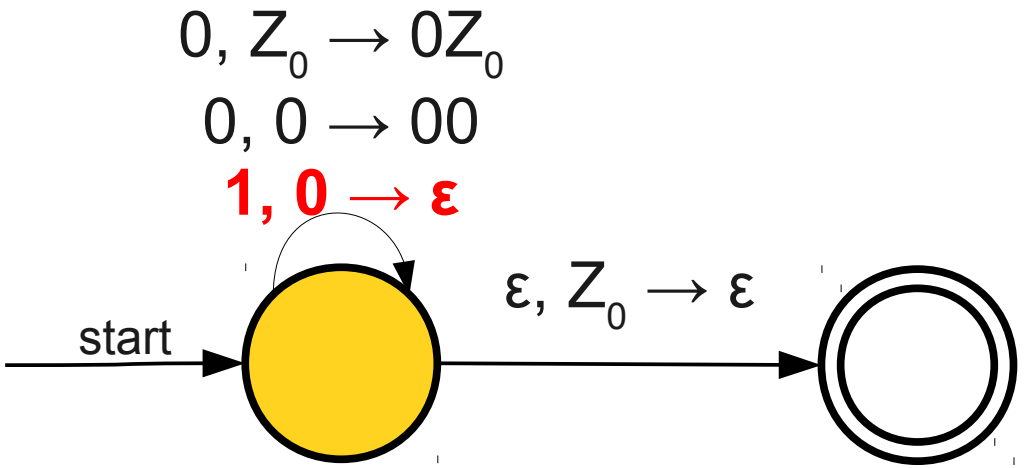


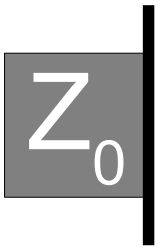
A Simple Pushdown Automaton



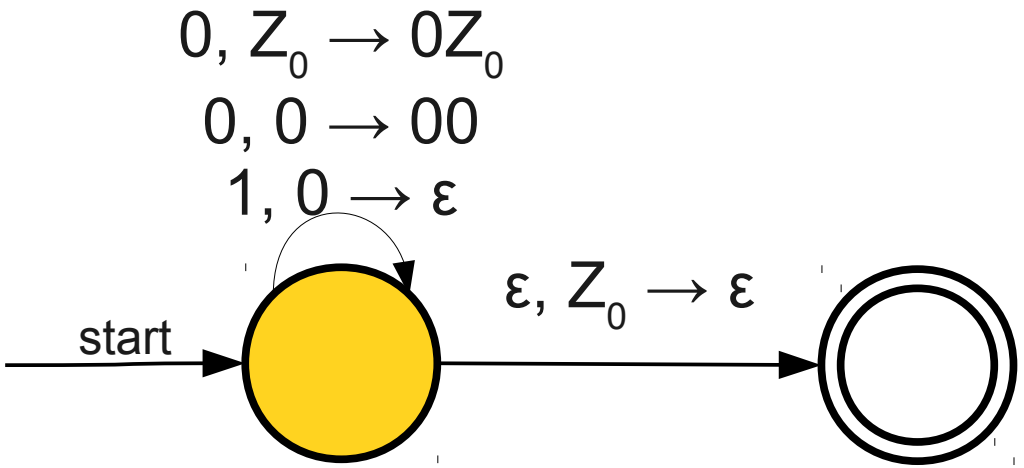


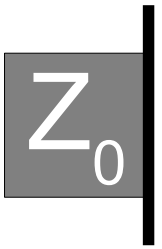
A Simple Pushdown Automaton



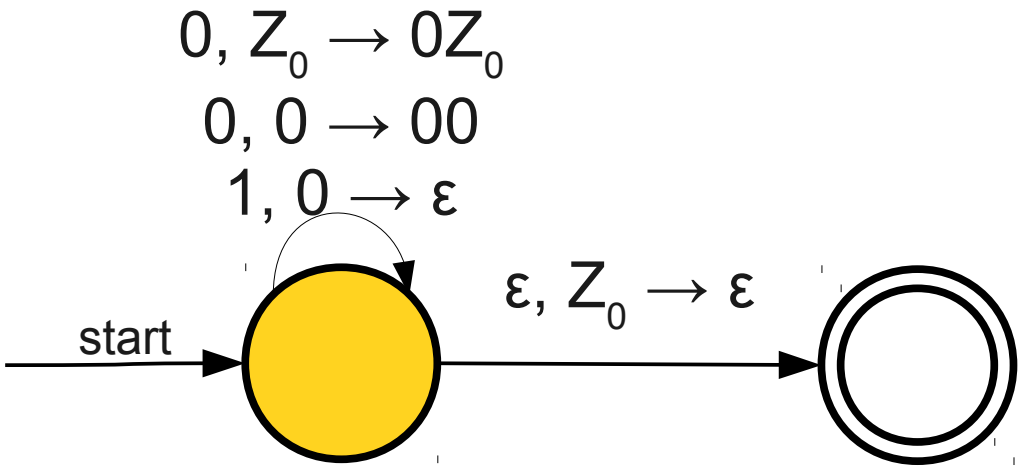


A Simple Pushdown Automaton

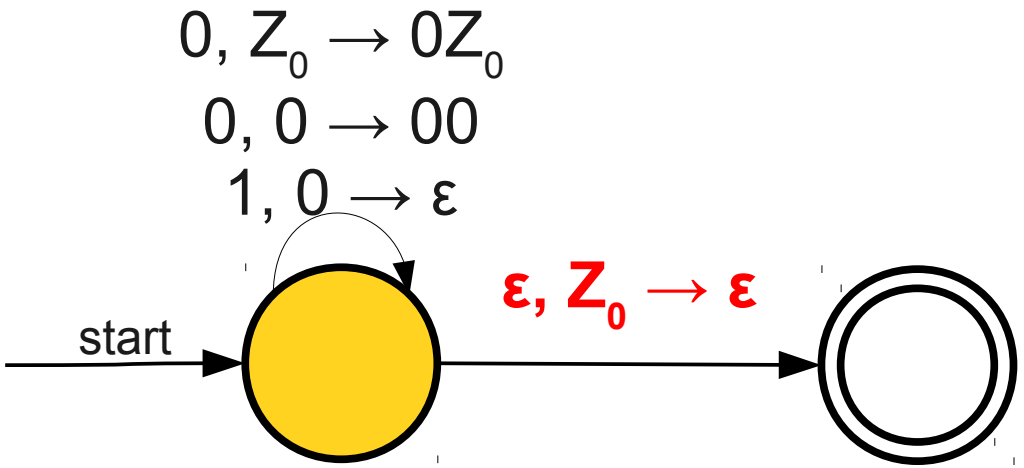




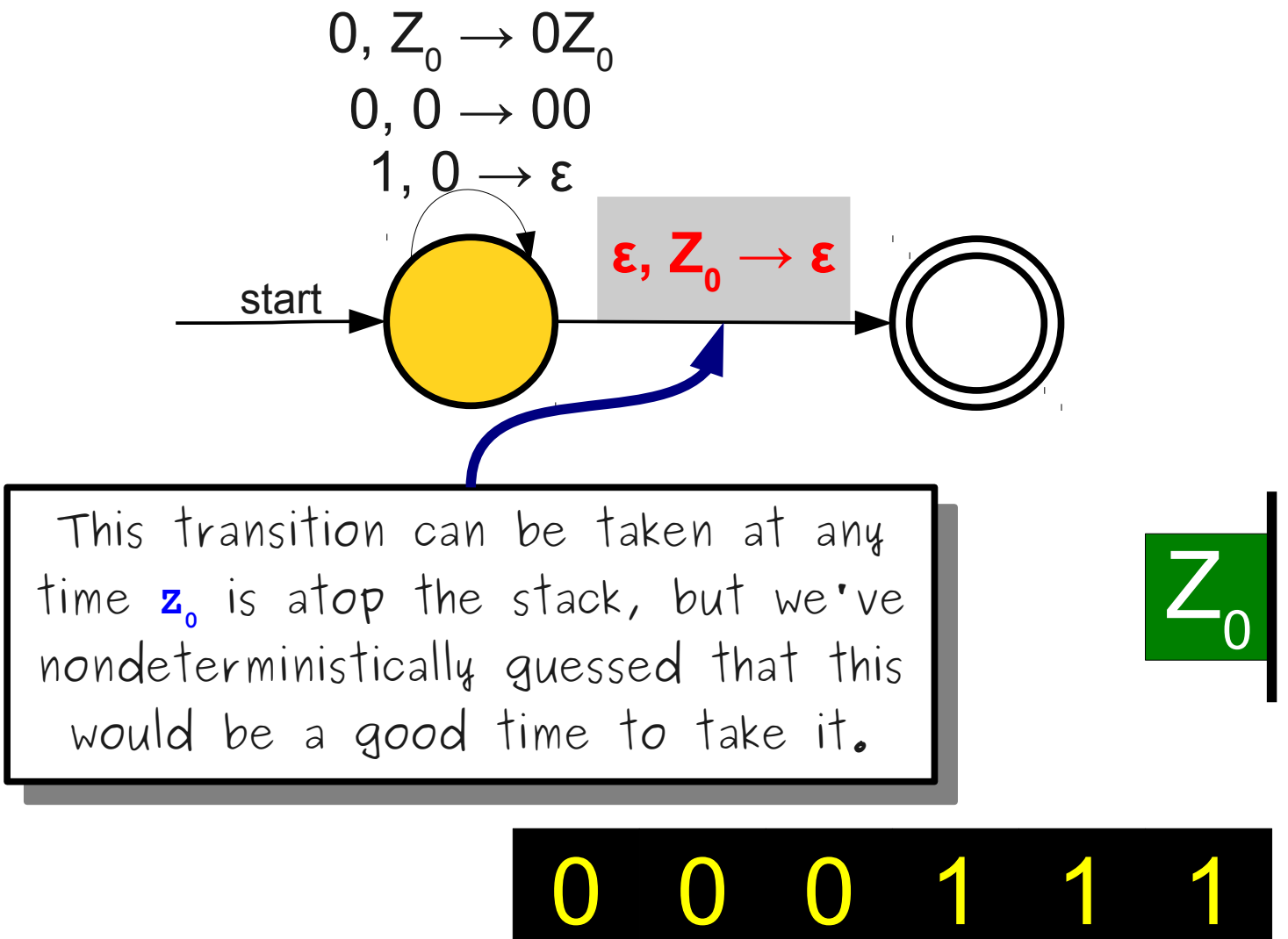
A Simple Pushdown Automaton



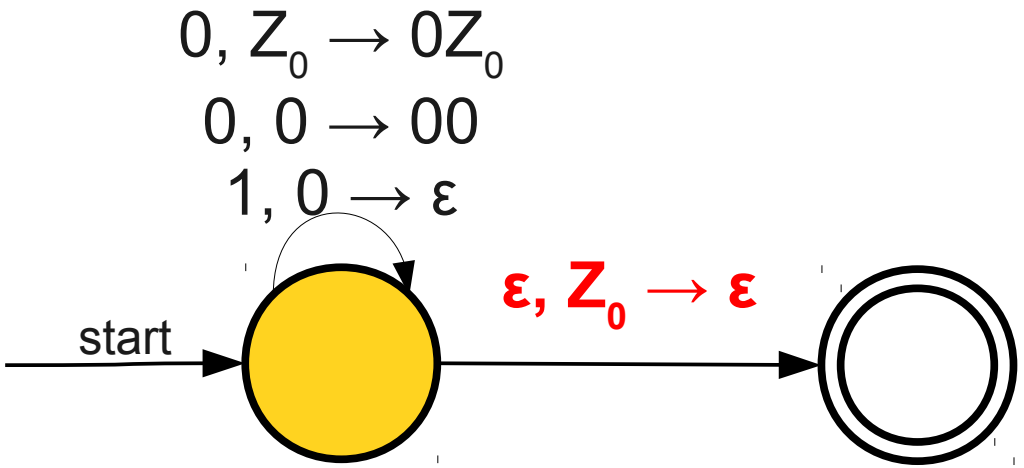
A Simple Pushdown Automaton



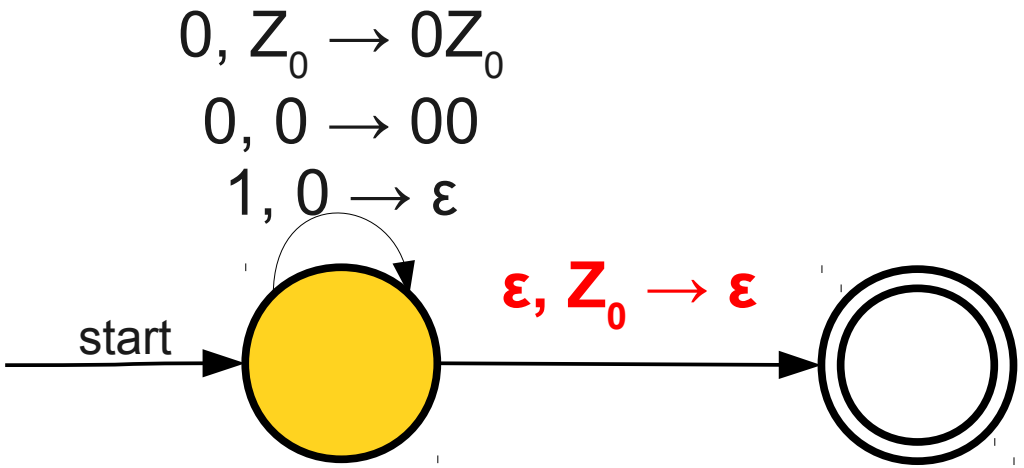
A Simple Pushdown Automaton



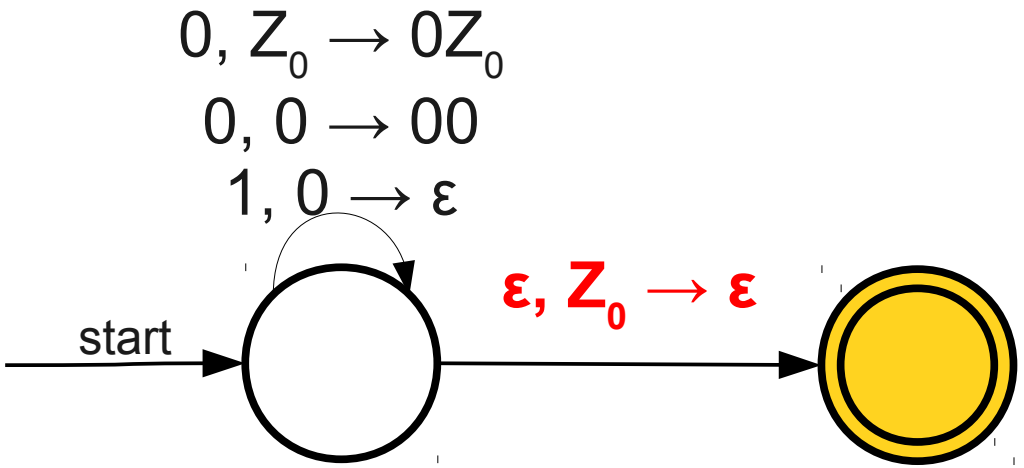
A Simple Pushdown Automaton



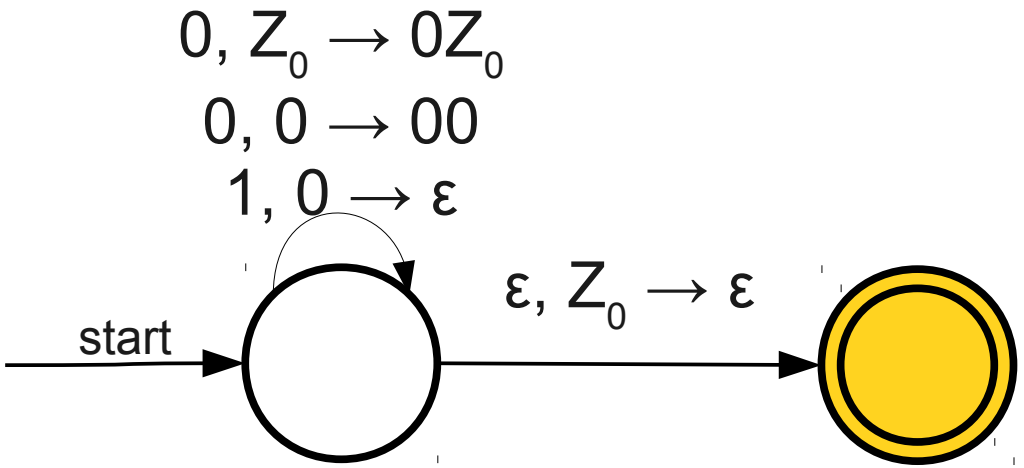
A Simple Pushdown Automaton



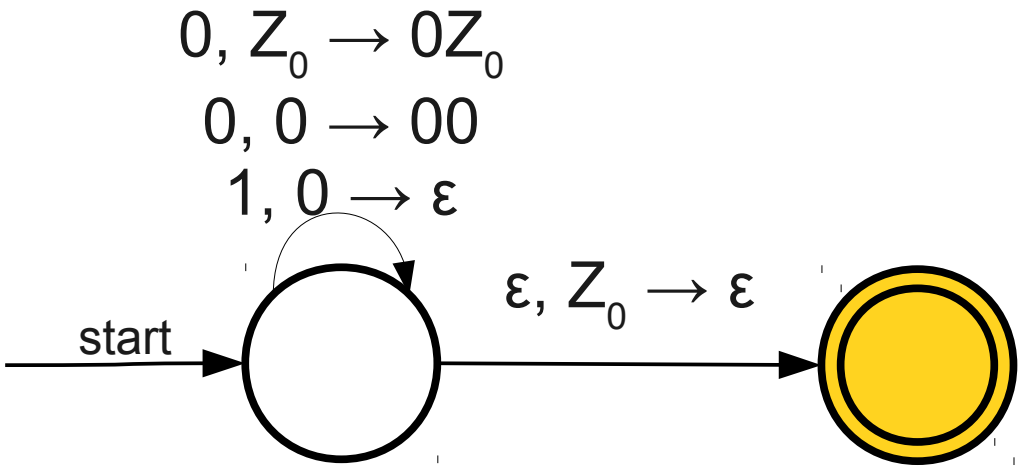
A Simple Pushdown Automaton



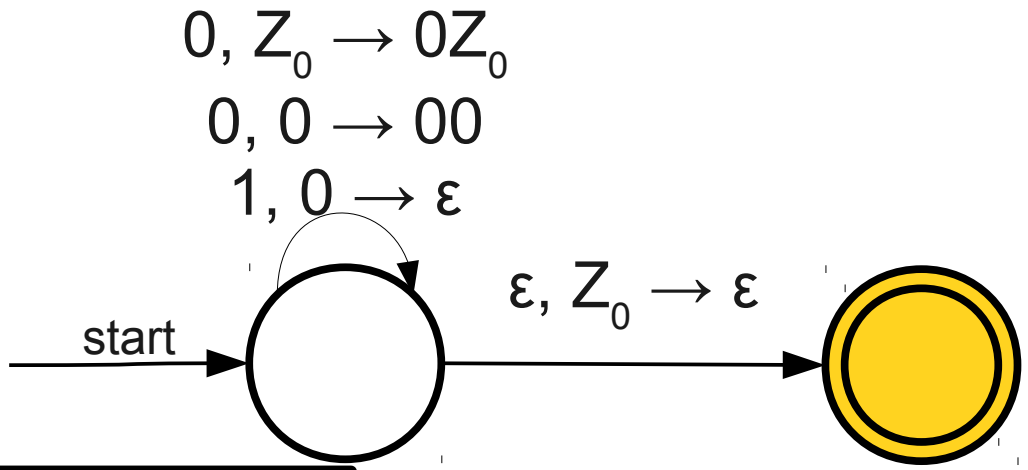
A Simple Pushdown Automaton



A Simple Pushdown Automaton

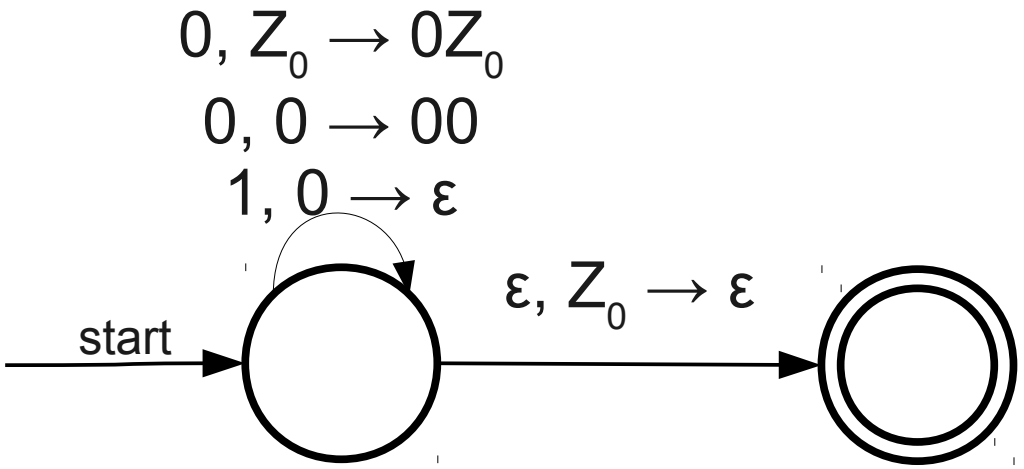


A Simple Pushdown Automaton

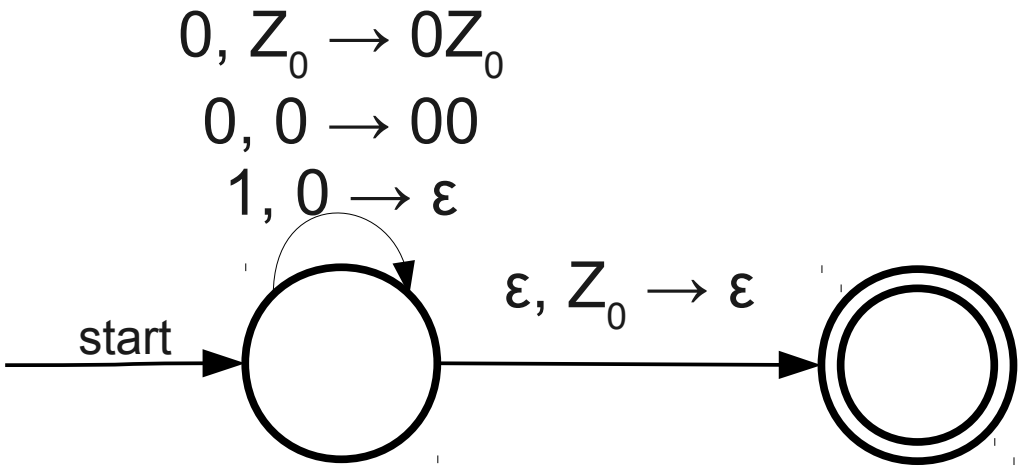


0 0 0 1 1 1

A Simple Pushdown Automaton

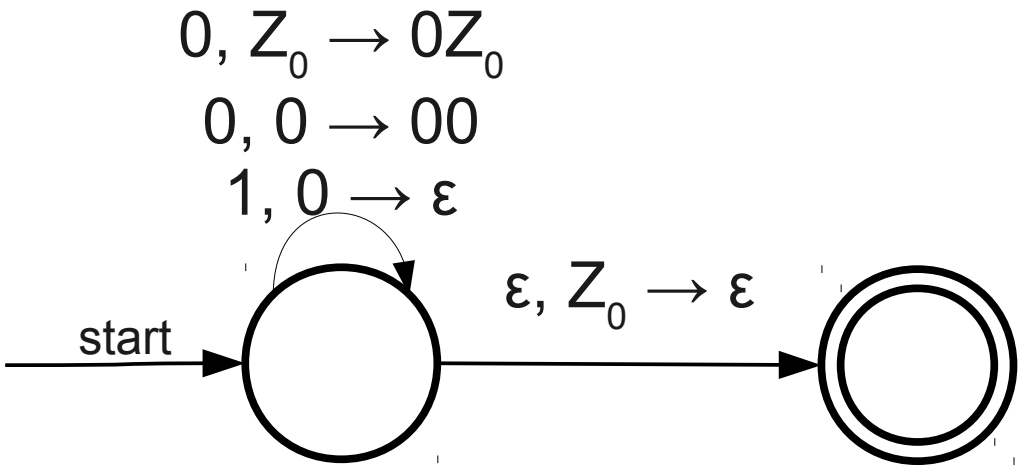


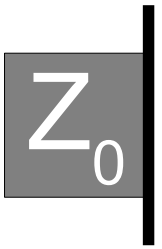
A Simple Pushdown Automaton



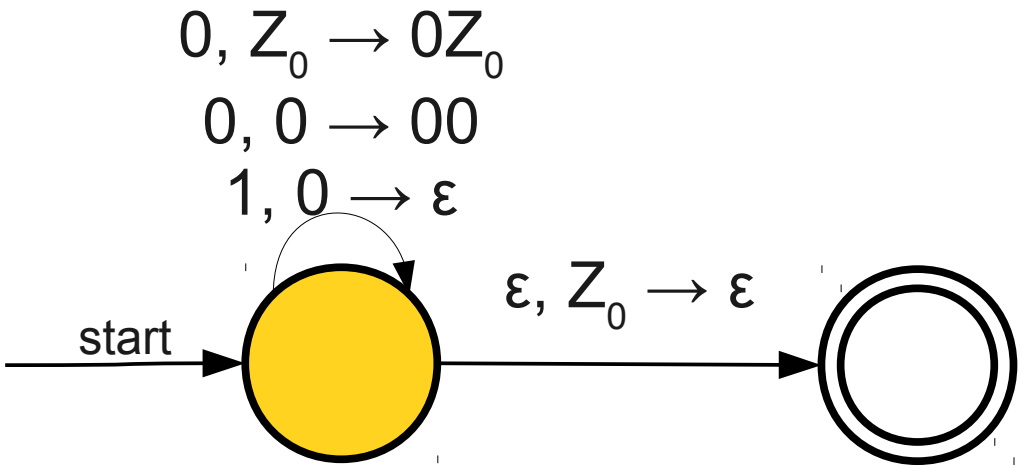
0 1 1 0 0 1

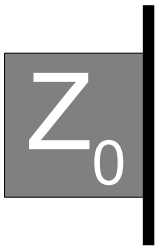
A Simple Pushdown Automaton



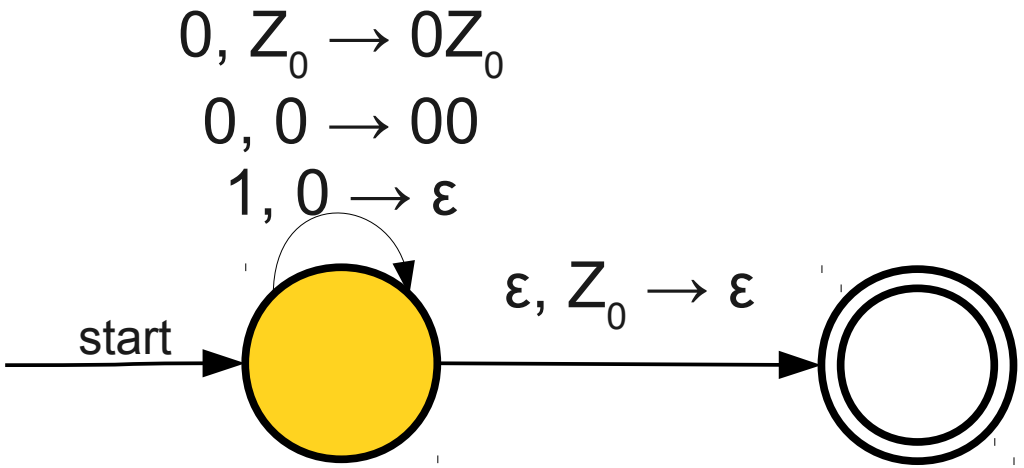


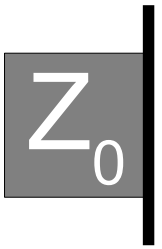
A Simple Pushdown Automaton



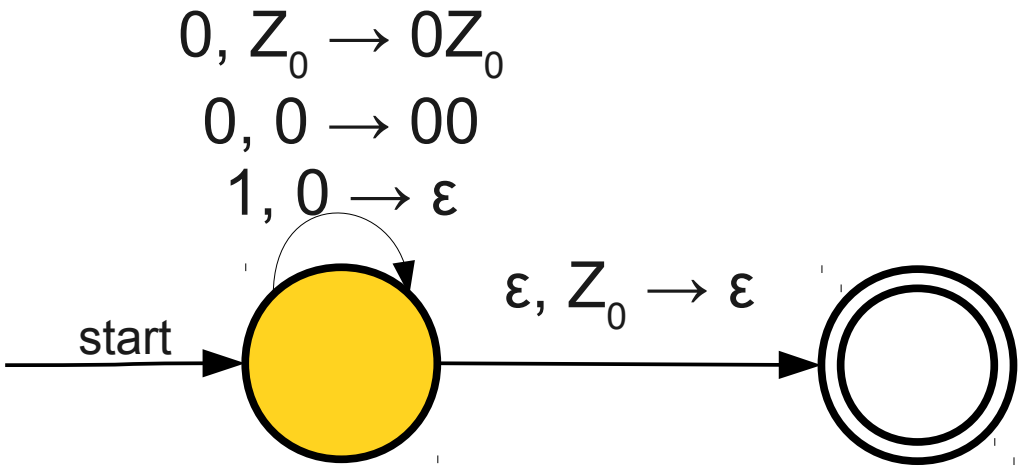


A Simple Pushdown Automaton

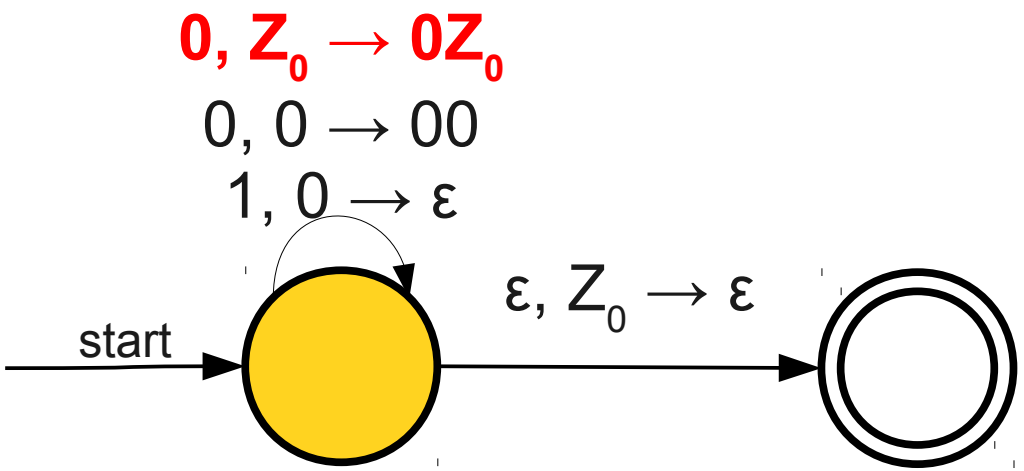




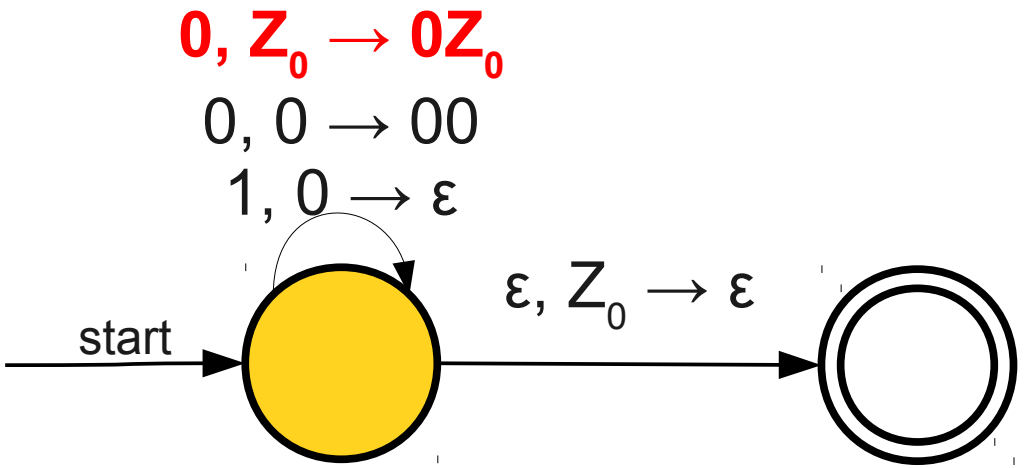
A Simple Pushdown Automaton



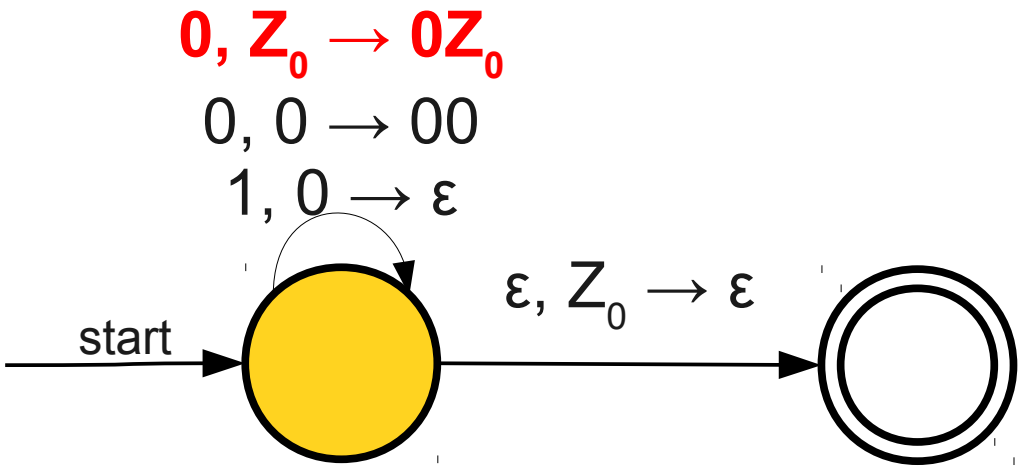
A Simple Pushdown Automaton

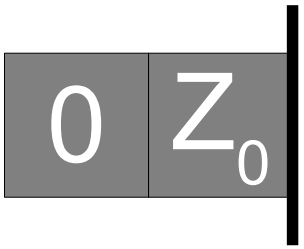


A Simple Pushdown Automaton

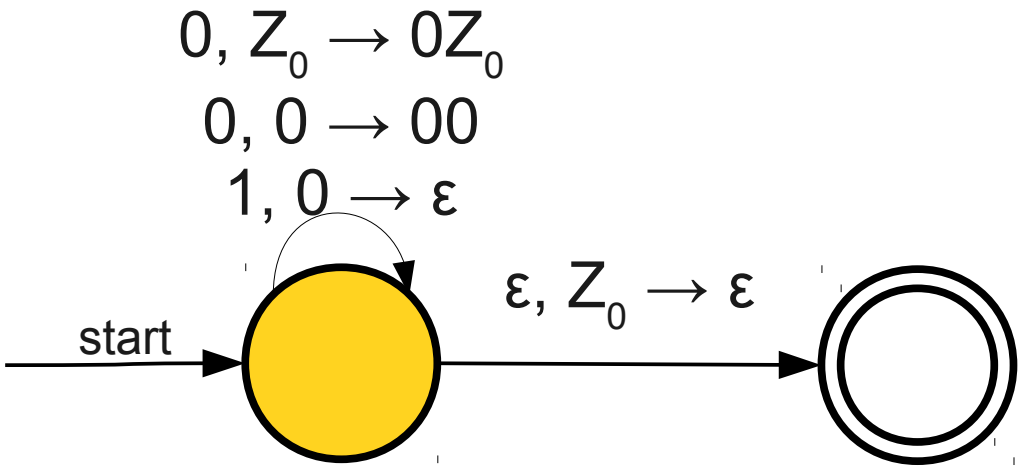


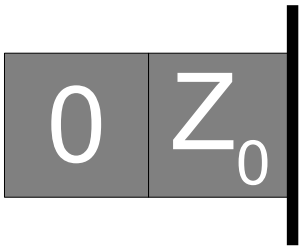
A Simple Pushdown Automaton



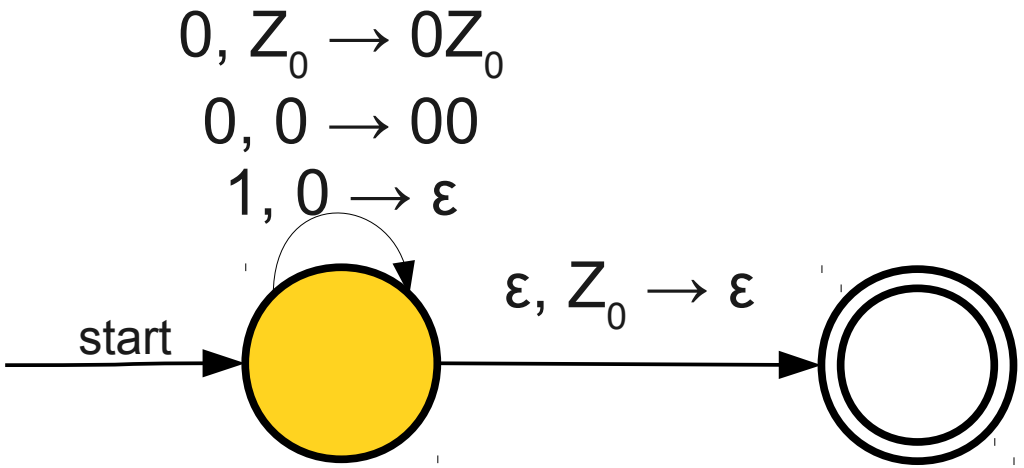


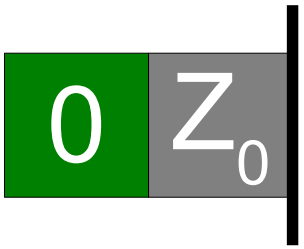
A Simple Pushdown Automaton



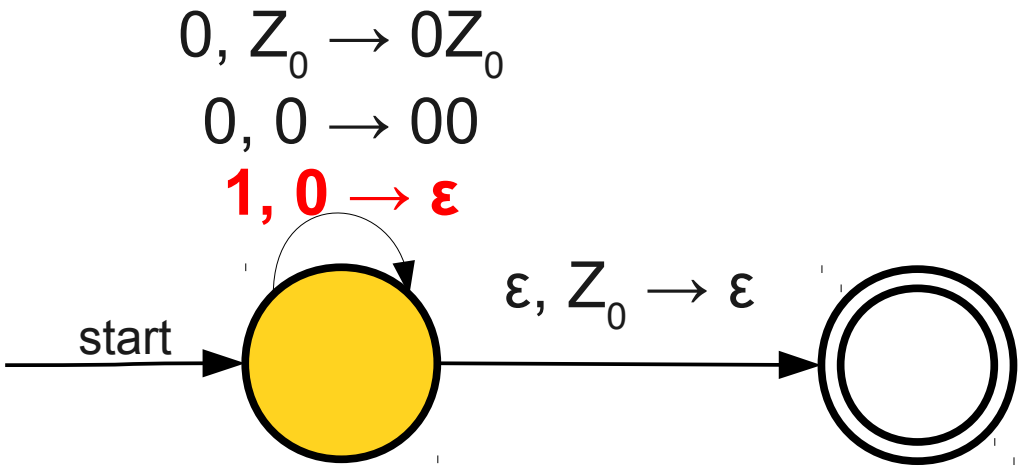


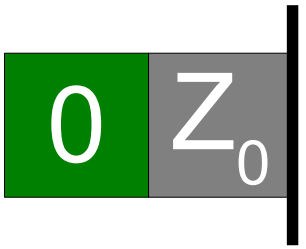
A Simple Pushdown Automaton



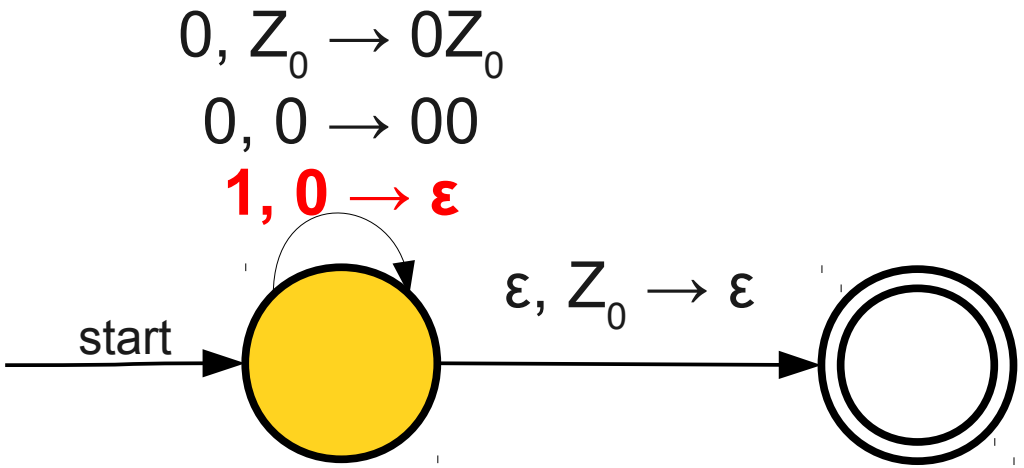


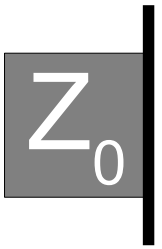
A Simple Pushdown Automaton



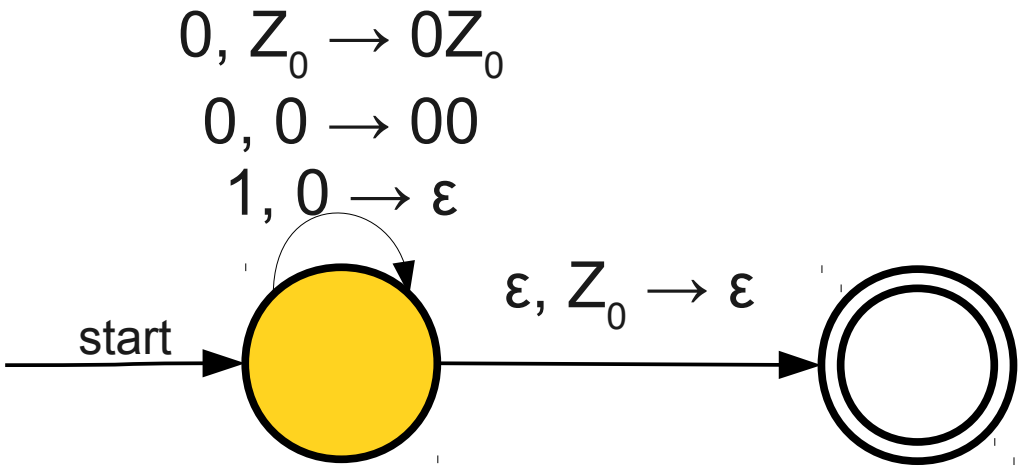


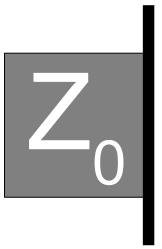
A Simple Pushdown Automaton

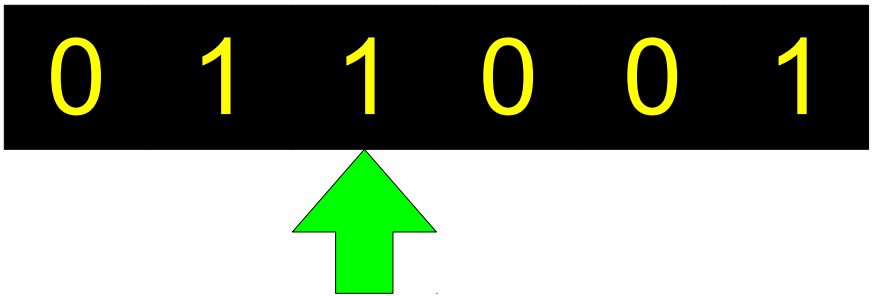




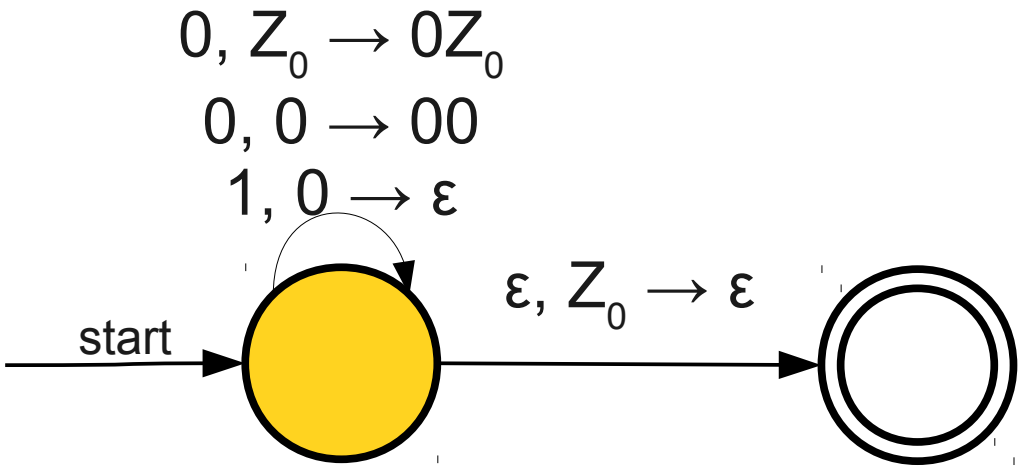
A Simple Pushdown Automaton

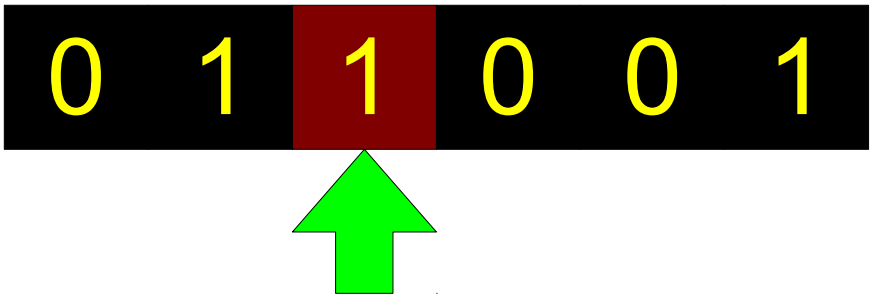




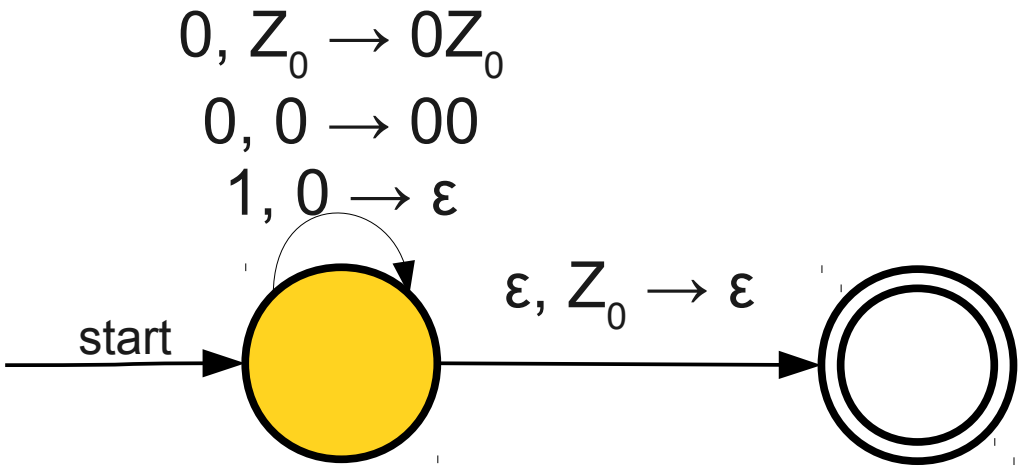


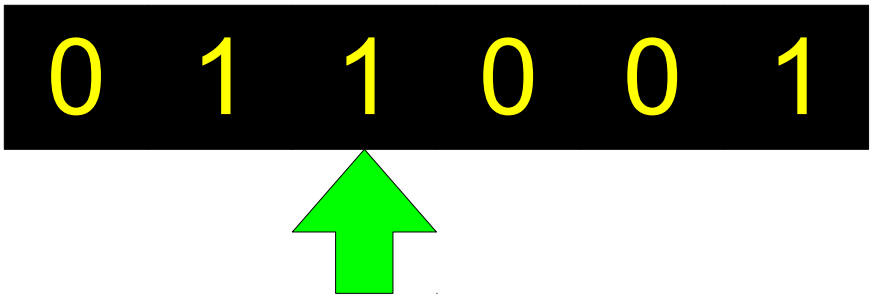
A Simple Pushdown Automaton



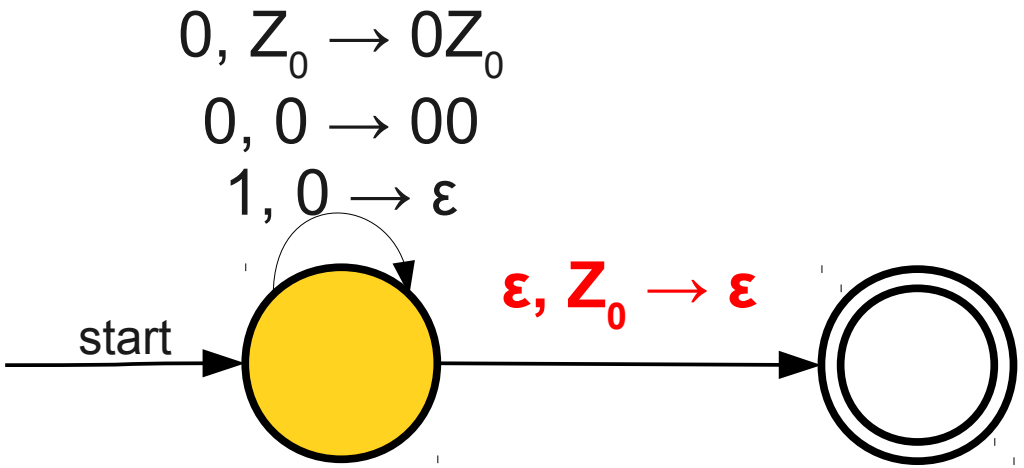


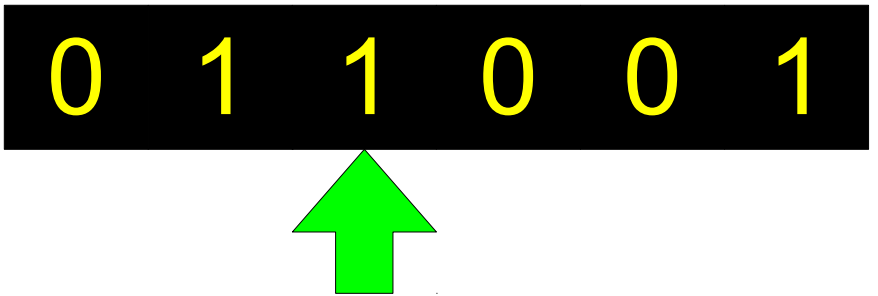
A Simple Pushdown Automaton



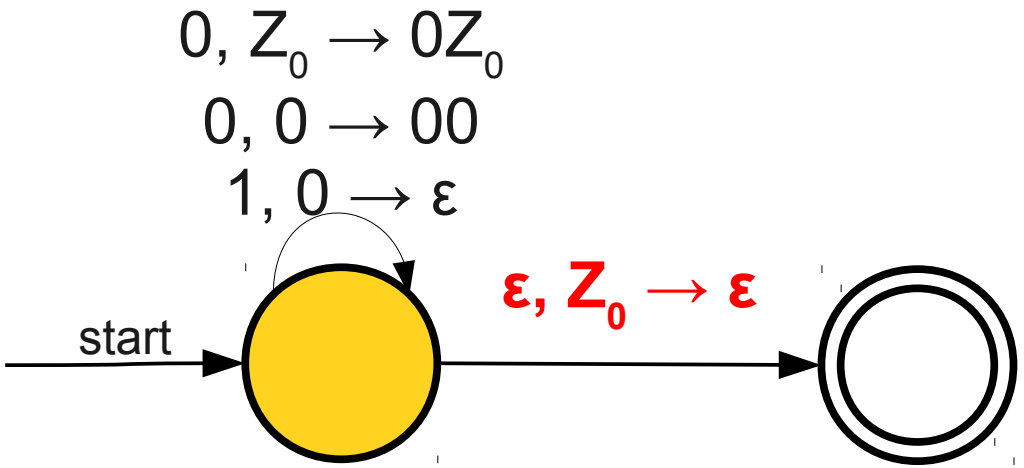


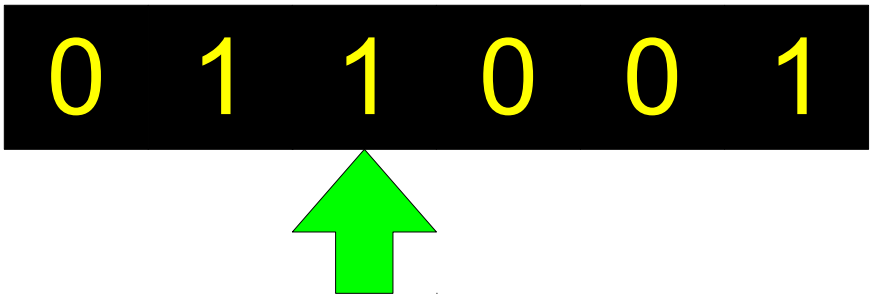
A Simple Pushdown Automaton



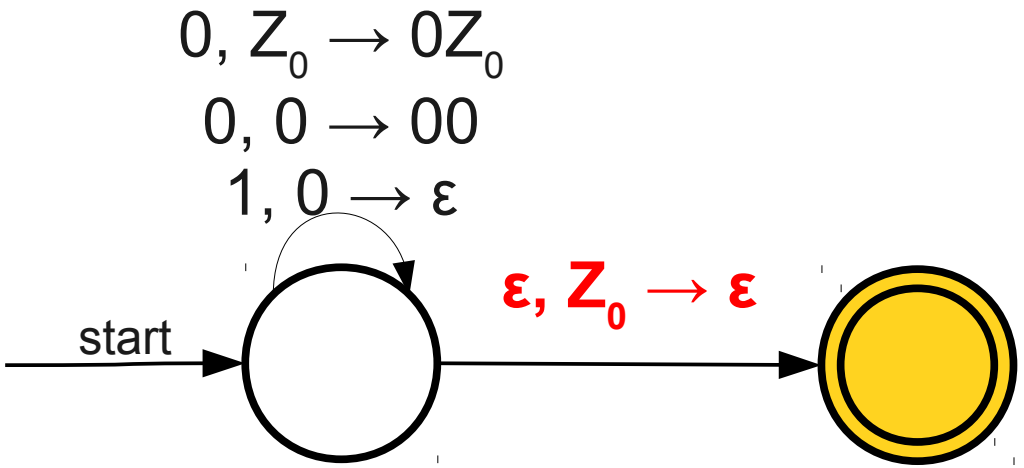


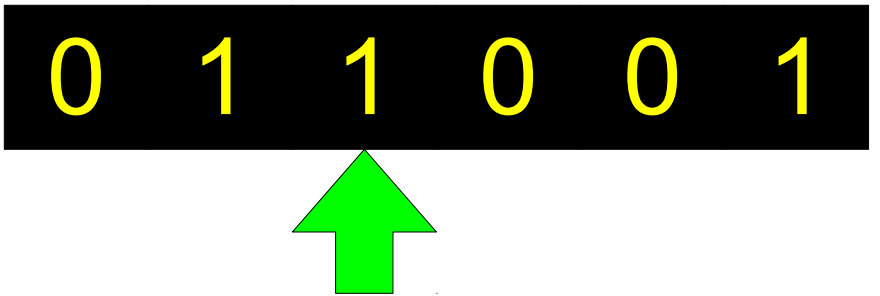
A Simple Pushdown Automaton



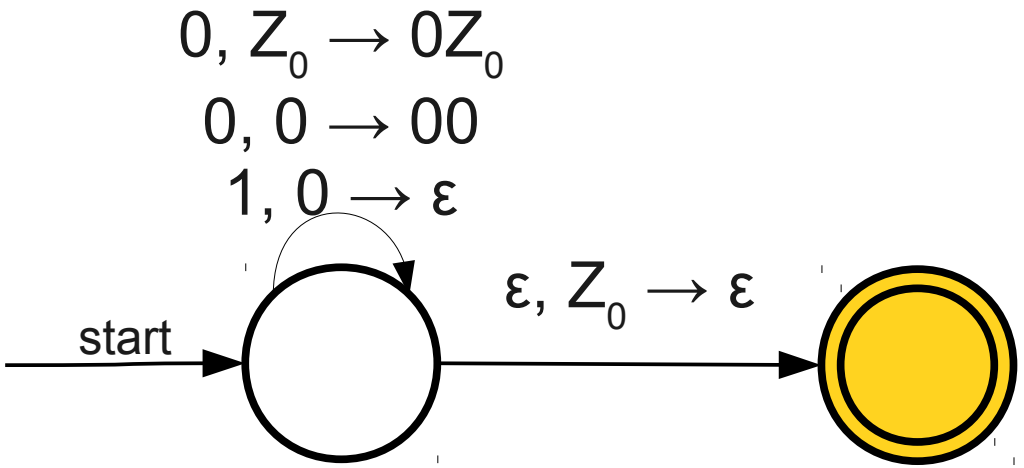


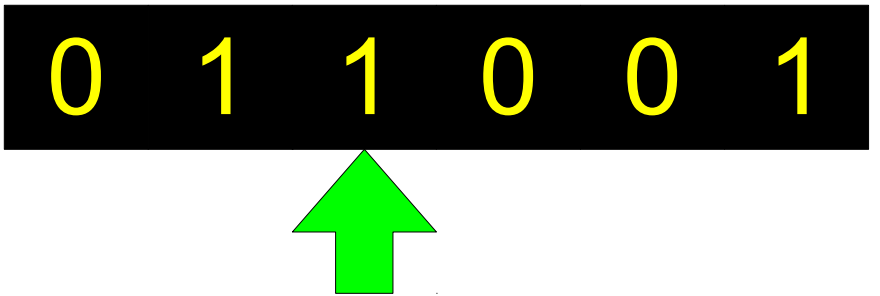
A Simple Pushdown Automaton



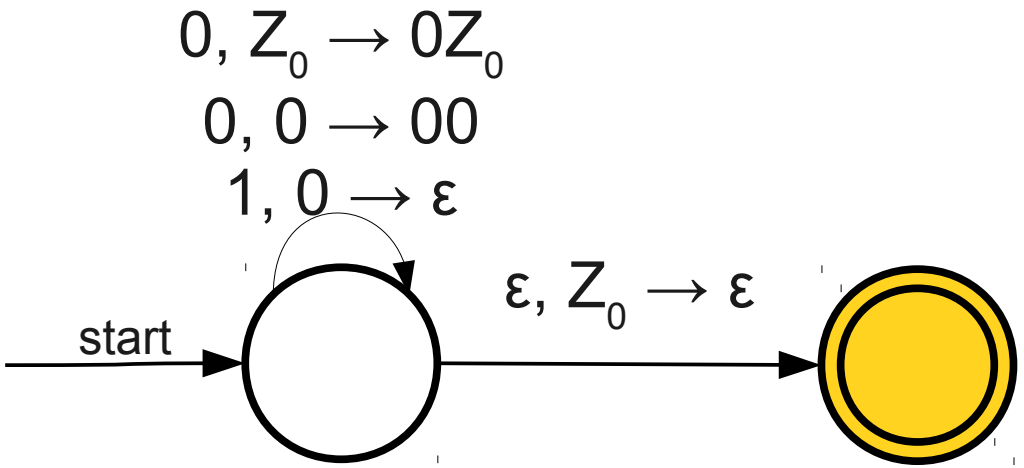


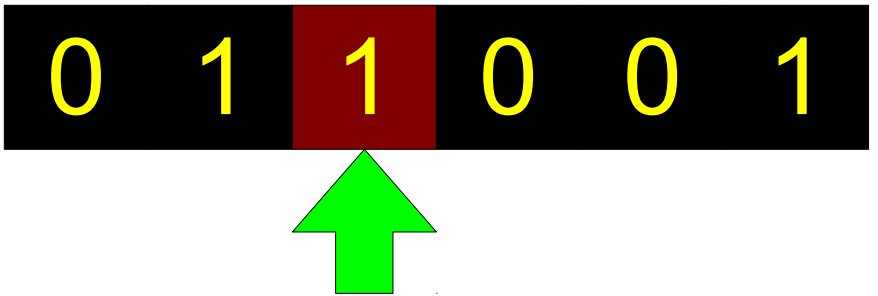
A Simple Pushdown Automaton



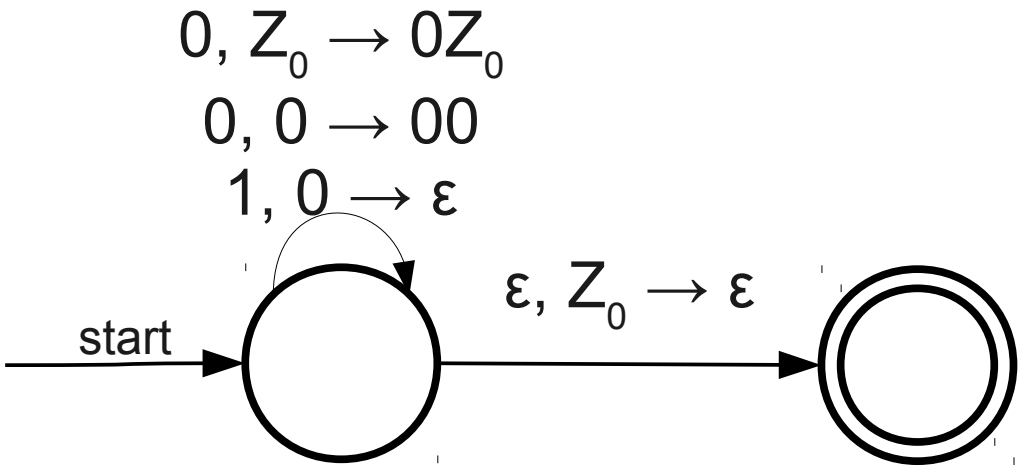


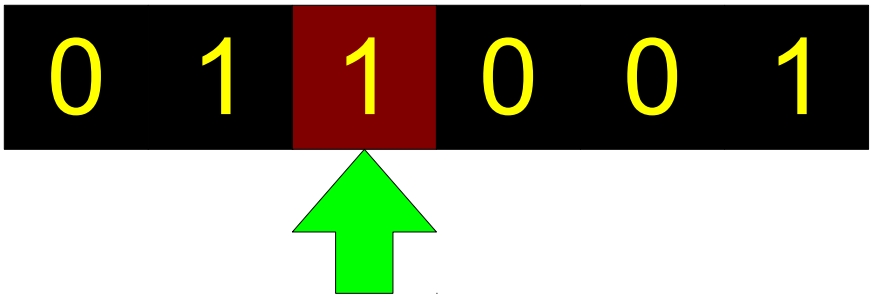
A Simple Pushdown Automaton



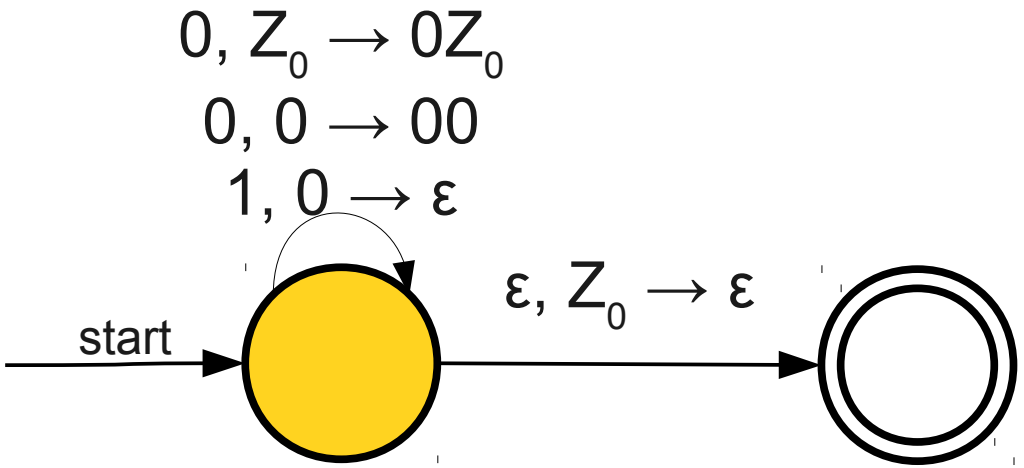


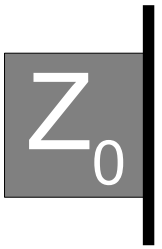
A Simple Pushdown Automaton



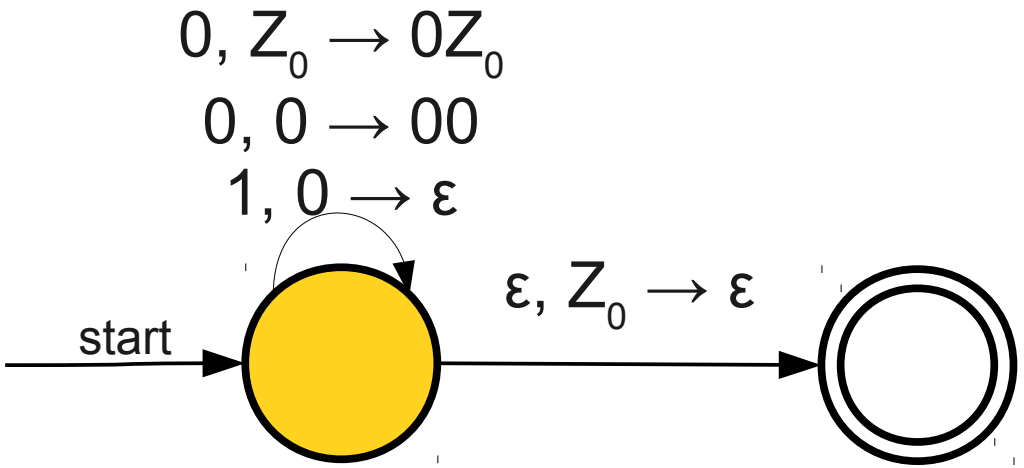


A Simple Pushdown Automaton

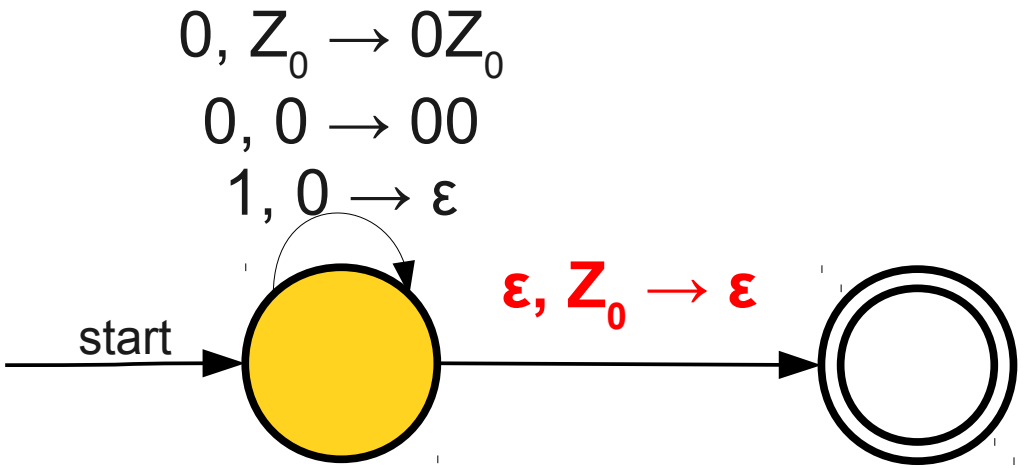




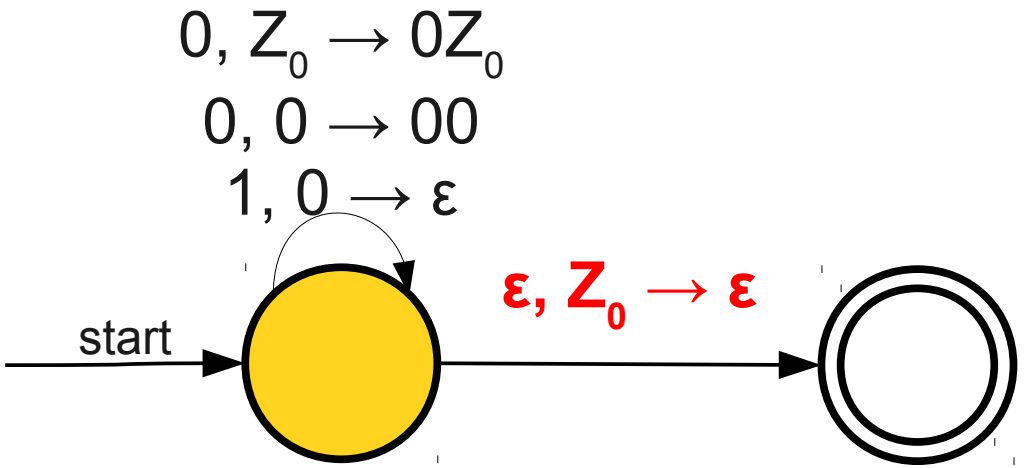
A Simple Pushdown Automaton



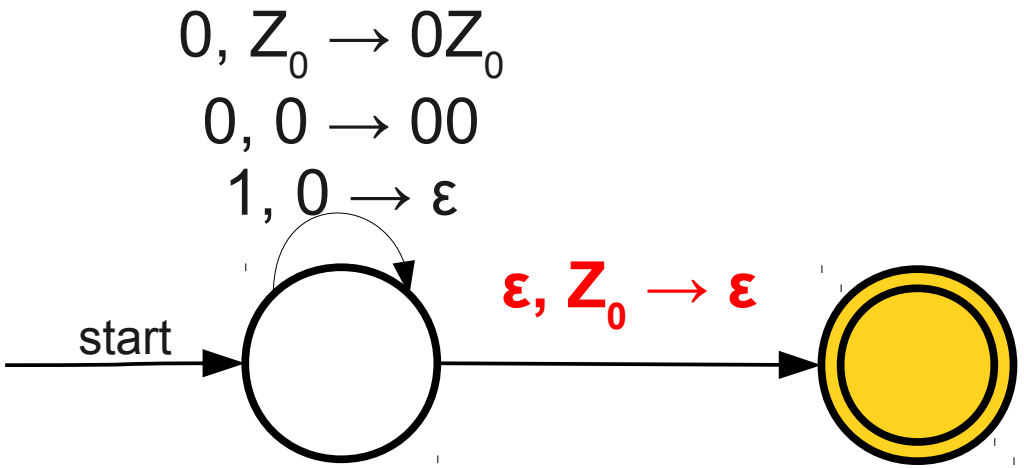
A Simple Pushdown Automaton



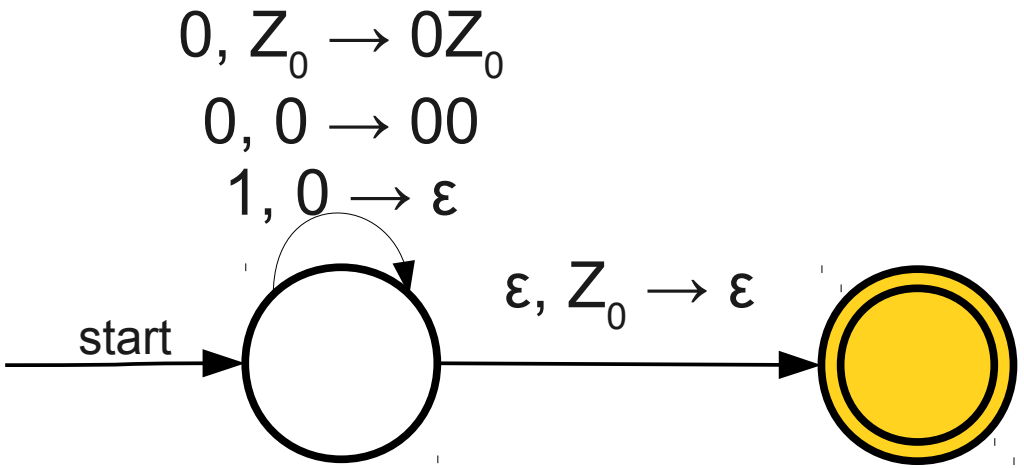
A Simple Pushdown Automaton



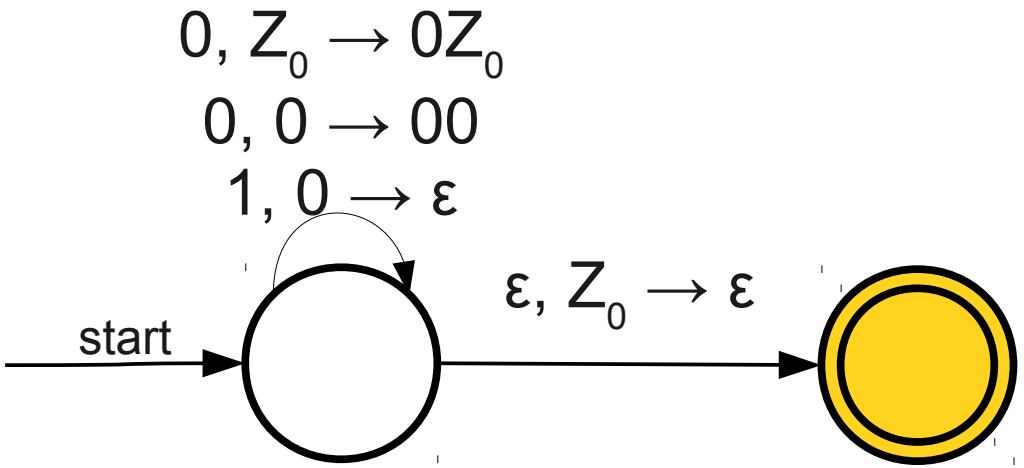
A Simple Pushdown Automaton



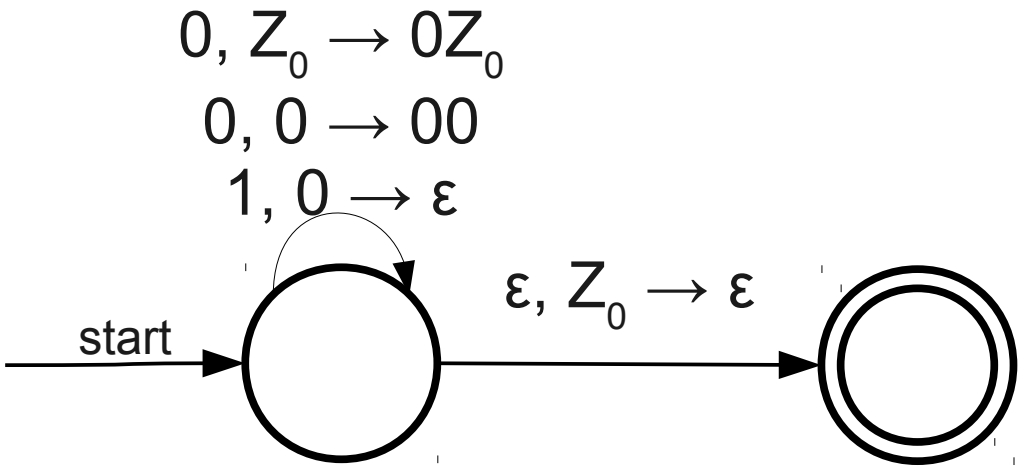
A Simple Pushdown Automaton



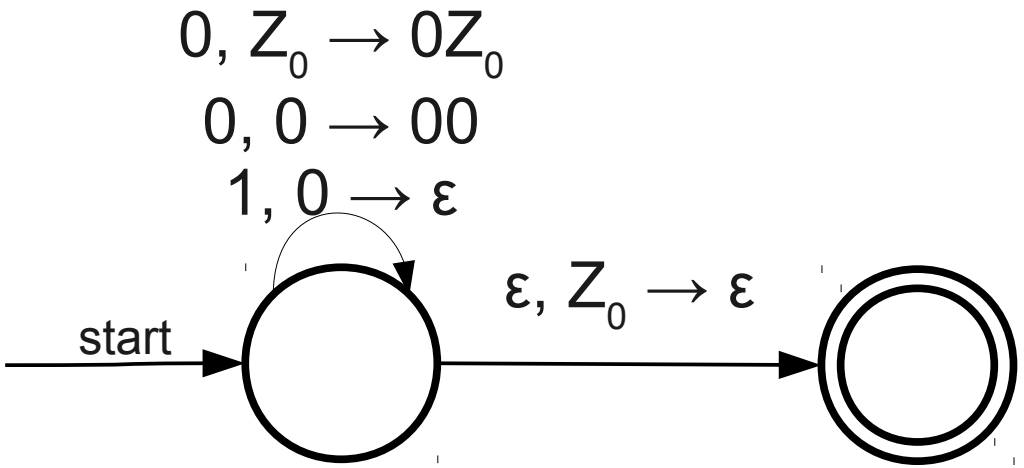
A Simple Pushdown Automaton



A Simple Pushdown Automaton

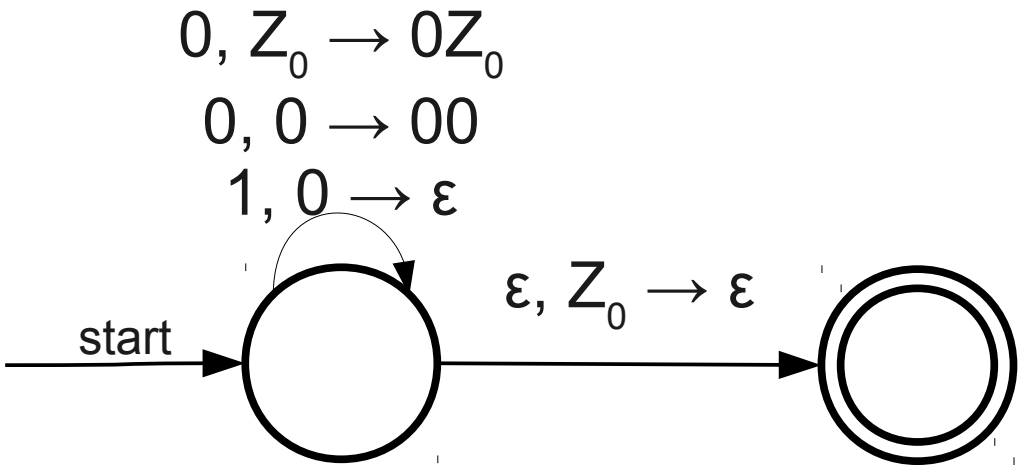


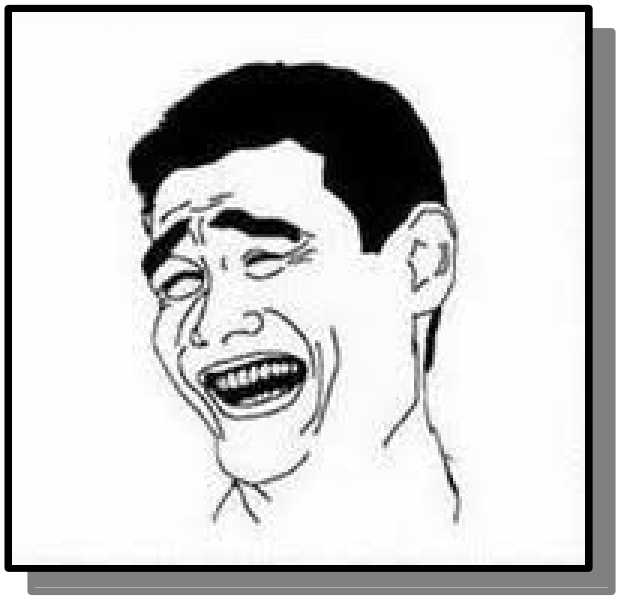
A Simple Pushdown Automaton



0 1 1 0 0 1

A Simple Pushdown Automaton





0 1 1 0 0 1

Exemplu de parser LL cu automat push-down

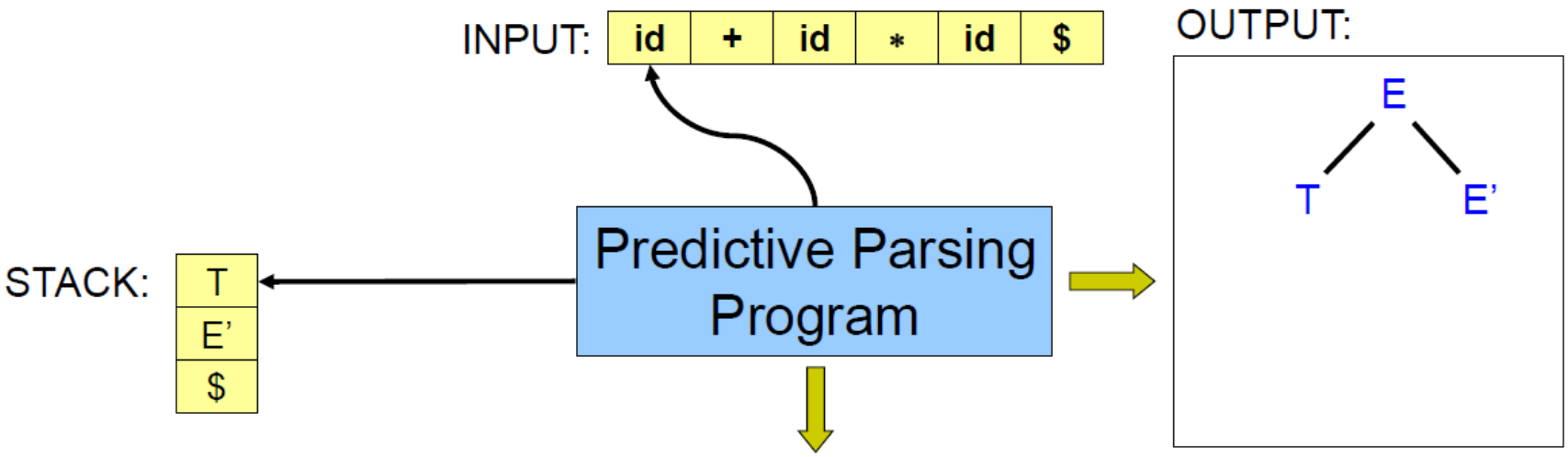
Gramatica:

$$
\begin{array}{lcl}
E & \rightarrow & T\ E' \\
E' & \rightarrow & +\ T\ E' \mid \epsilon \\
T & \rightarrow & F\ T' \\
T' & \rightarrow & *\ F\ T' \mid \epsilon \\
F & \rightarrow & (\ E\ ) \mid \mathbf{id}
\end{array}
$$

Tabelul de parsare:

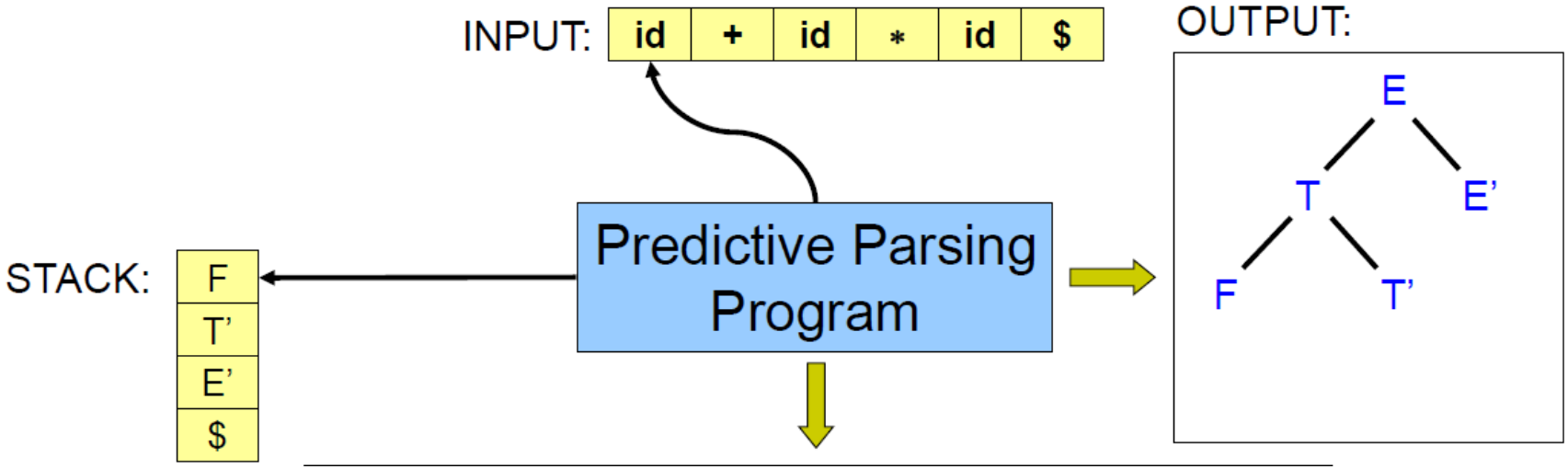
NON-TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow \mathbf{id}$			$F \rightarrow (E)$		

Exemplu de parser LL cu automat push-down



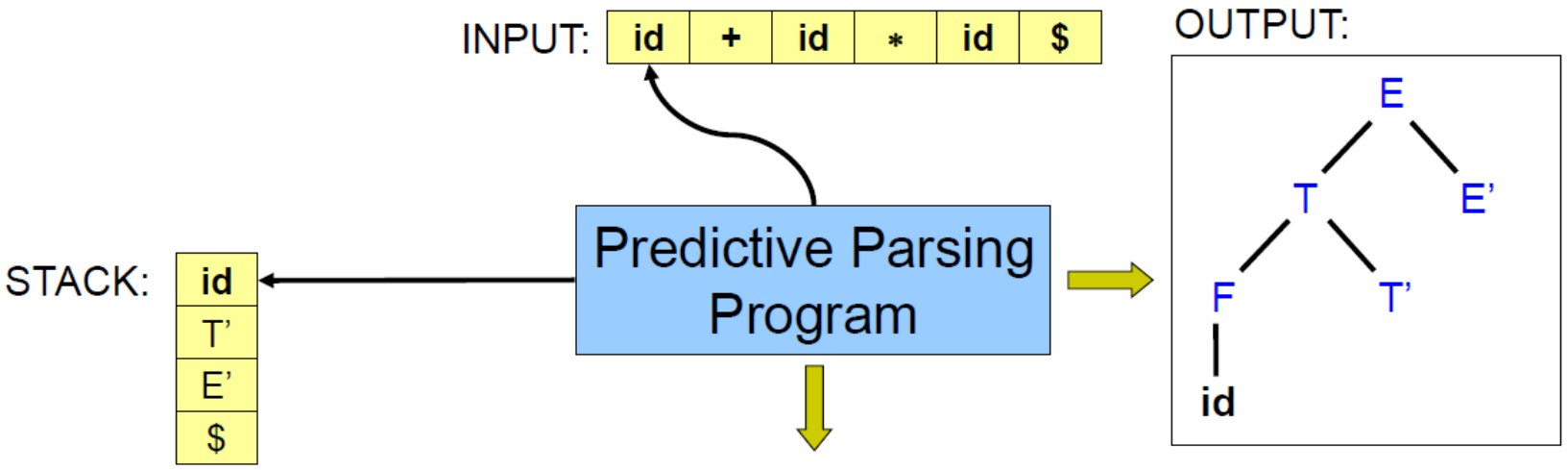
NON - TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \epsilon$	$T' \to *FT'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to \mathbf{id}$			$F \to (E)$		

Exemplu de parser LL cu automat push-down



NON - TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \epsilon$	$T' \to *FT'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to \mathbf{id}$			$F \to (E)$		

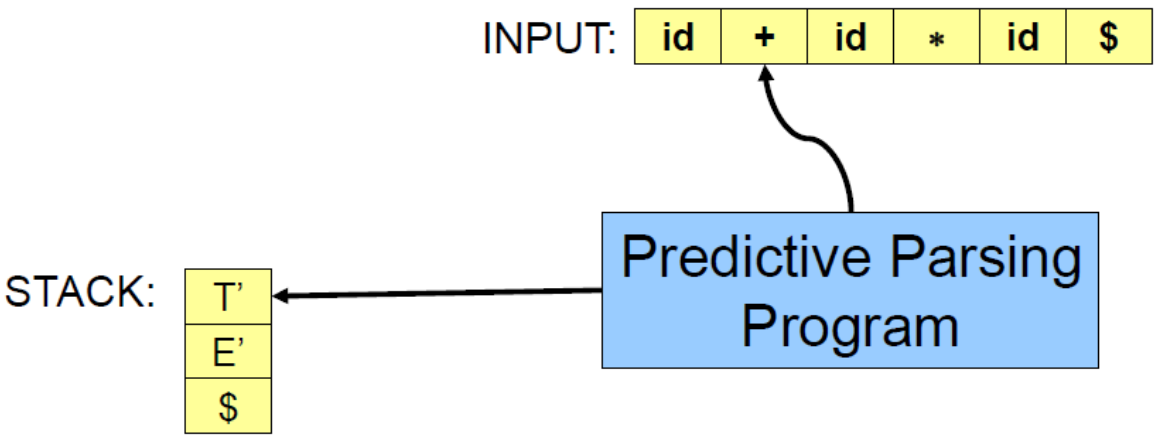
Exemplu de parser LL cu automat push-down

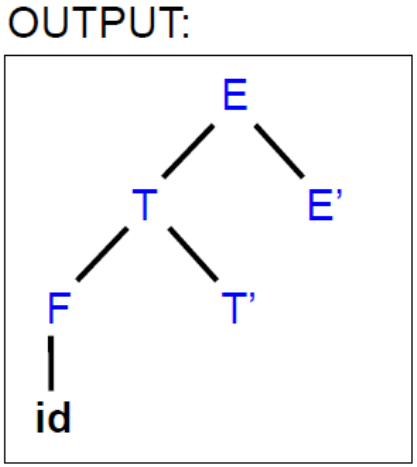


NON - TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \epsilon$	$T' \to *FT'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to \mathbf{id}$			$F \to (E)$		

Exemplu de parser LL cu automat push-down

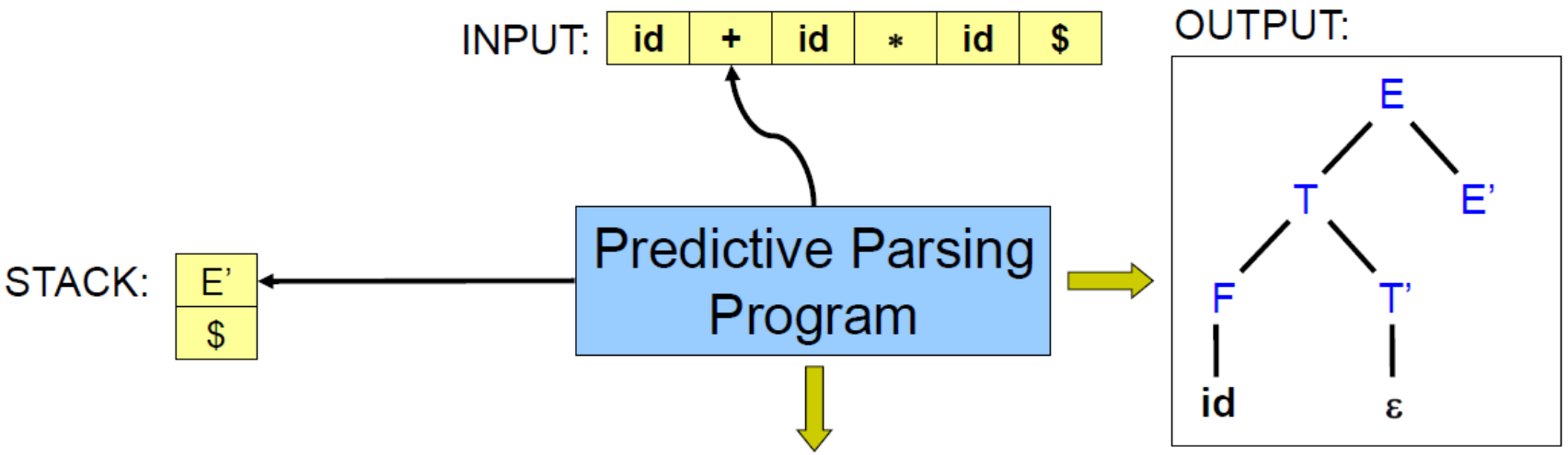
Actiunea cand Top(Stack) = input ≠ $: 'Pop' din stiva, avanseaza pe banda de intrare.





NON-TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \epsilon$	$T' \to *FT'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to \mathbf{id}$			$F \to (E)$		

Exemplu de parser LL cu automat push-down



NON - TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \epsilon$	$T' \to *FT'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to \mathbf{id}$			$F \to (E)$		

Exemplu de parser LL cu automat push-down

Si tot asa, se construieste urmatorul arbore de derivare:

$E' \rightarrow +TE'$

$T \rightarrow FT'$

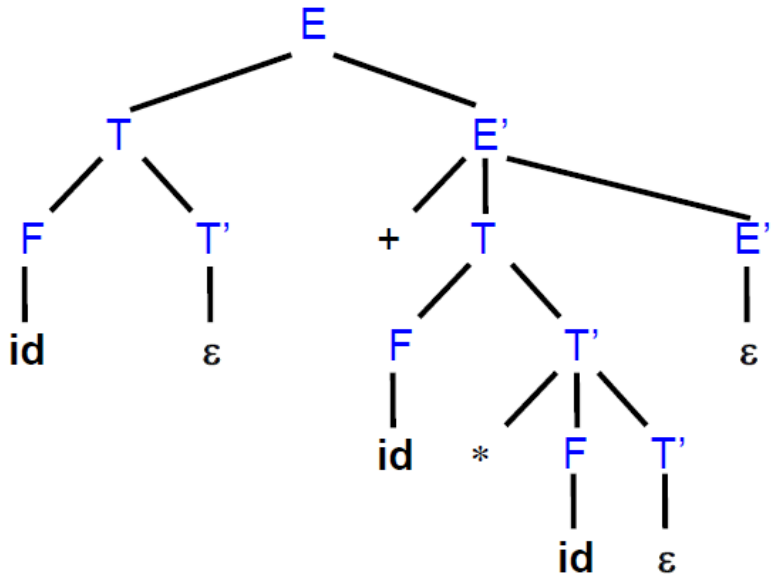
$F \rightarrow \mathbf{id}$

$T' \rightarrow * FT'$

$F \rightarrow \mathbf{id}$

$T' \rightarrow \varepsilon$

$E' \rightarrow \varepsilon$



Cand Top(Stack) = input = $

Parserul se opreste si accepta intrarea.

(Aho,Sethi, Ullman, pp. 188)

Terminologie: LL vs LR

- LL(k)
  - Scaneaza intrarea "Left-to-right"
  - "Left-most derivation" – deriveaza mereu cel mai din stanga neterminal
  - k simboluri de lookahead
  - Face o traversare in pre-ordine a arborelui de parsare
- LR(k)
  - Scaneaza intrarea "Left-to-right"
  - "Right-most derivation" – 'deriveaza' cel mai din dreapta neterminal
  - k simboluri de lookahead
  - Face o traversare in post-ordine a arborelui de parsare

Parserul ascendent (LR)

Un parser ascendent, sau “parser shift-reduce”, incepe de la ‘frunze’ si construieste spre varf arborele de derivare

Pasii de reducere urmaresc o derivare dreapta in ordine inversa

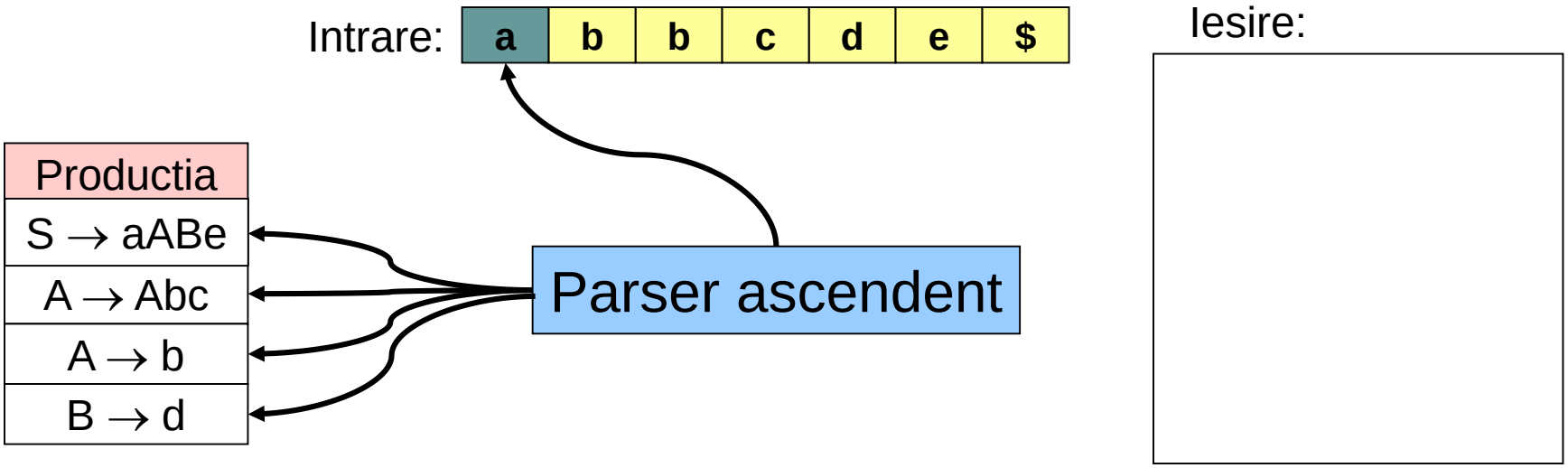
Sa consideram GRAMATICA:

$S \rightarrow aABe$
$A \rightarrow Abc \mid b$
$B \rightarrow d$

Vrem sa parsam sirul de Intrare abbcde.

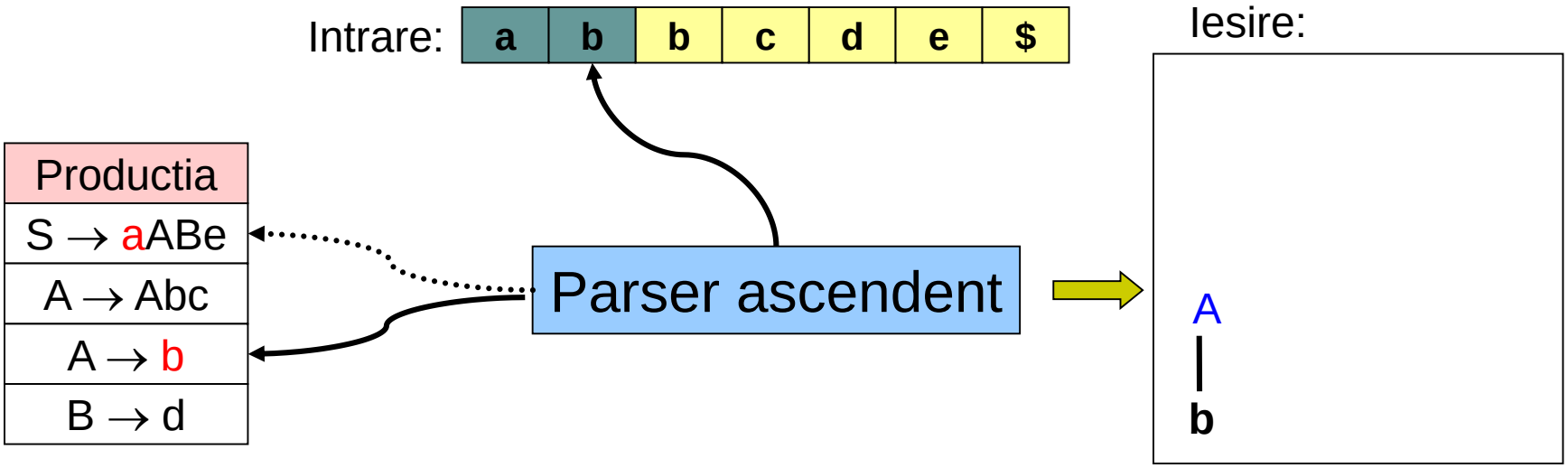
(Aho,Sethi,Ullman, pp. 195)

Exemplu de parser ascendent



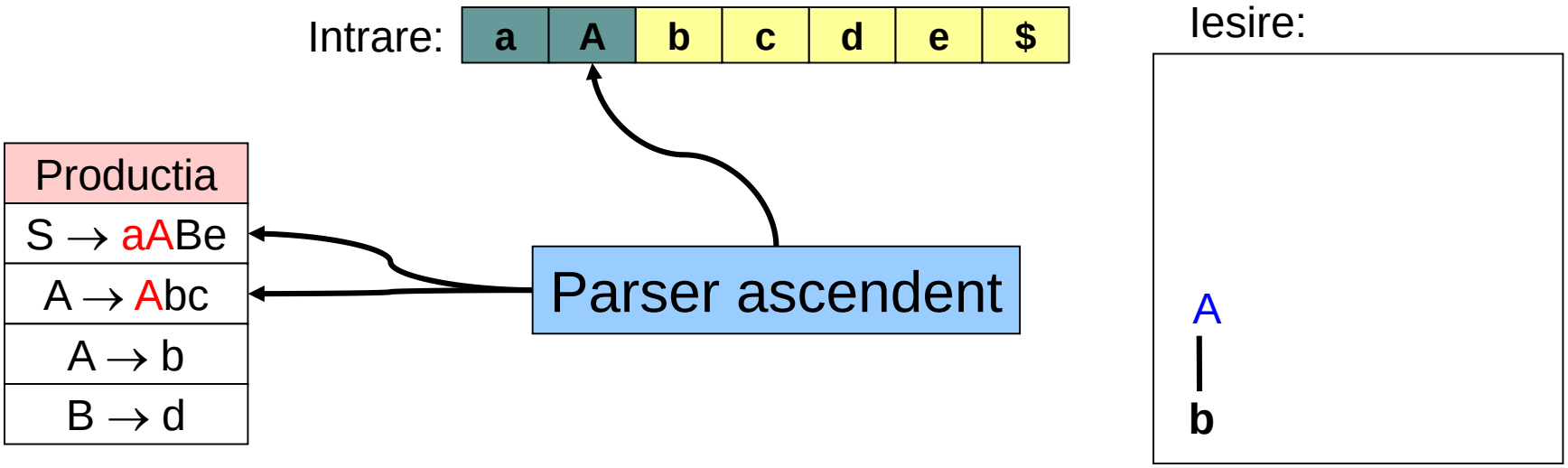
(Aho,Sethi,Ullman, pp. 195)

Exemplu de parser ascendent



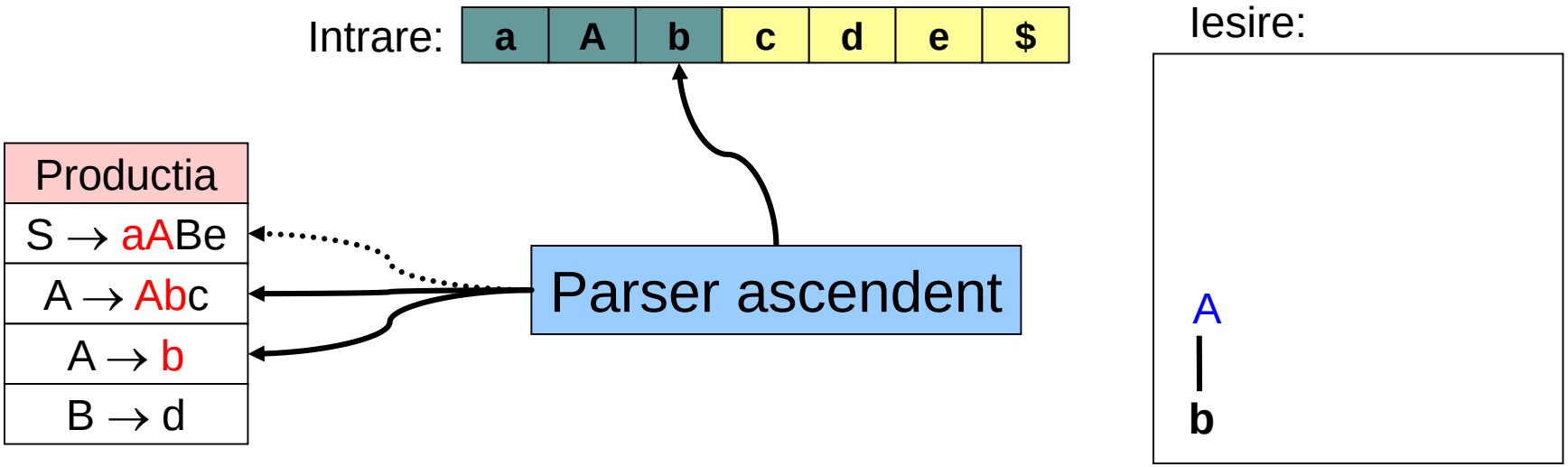
(Aho,Sethi,Ullman, pp. 195)

Exemplu de parser ascendent



(Aho,Sethi,Ullman, pp. 195)

Exemplu de parser ascendent

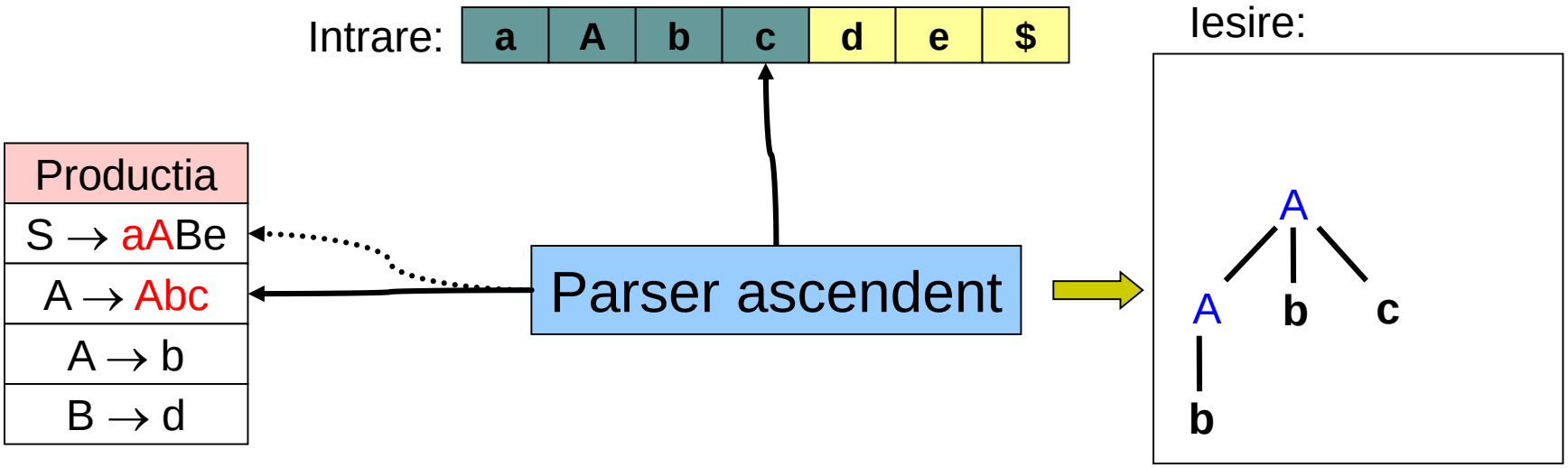


Nu reducem in acest exemplu. Un parser

ar reduce, s-ar impotmoli si ar trebui sa faca backtracking!

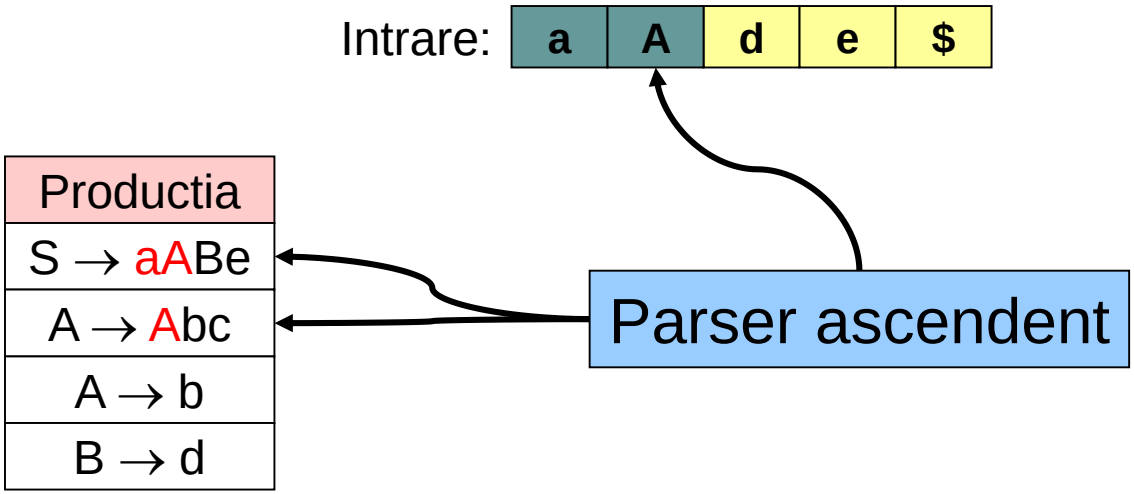
(Aho,Sethi,Ullman, pp. 195)

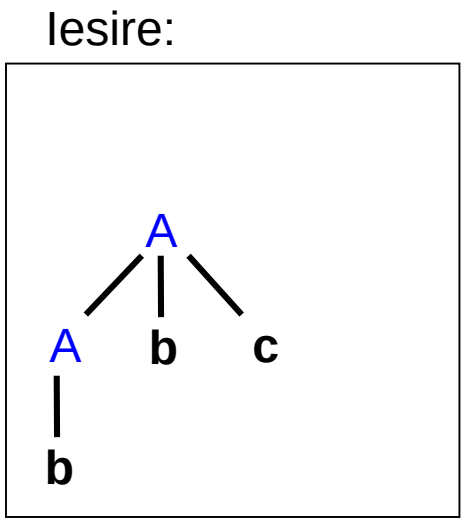
Exemplu de parser ascendent



(Aho,Sethi,Ullman, pp. 195)

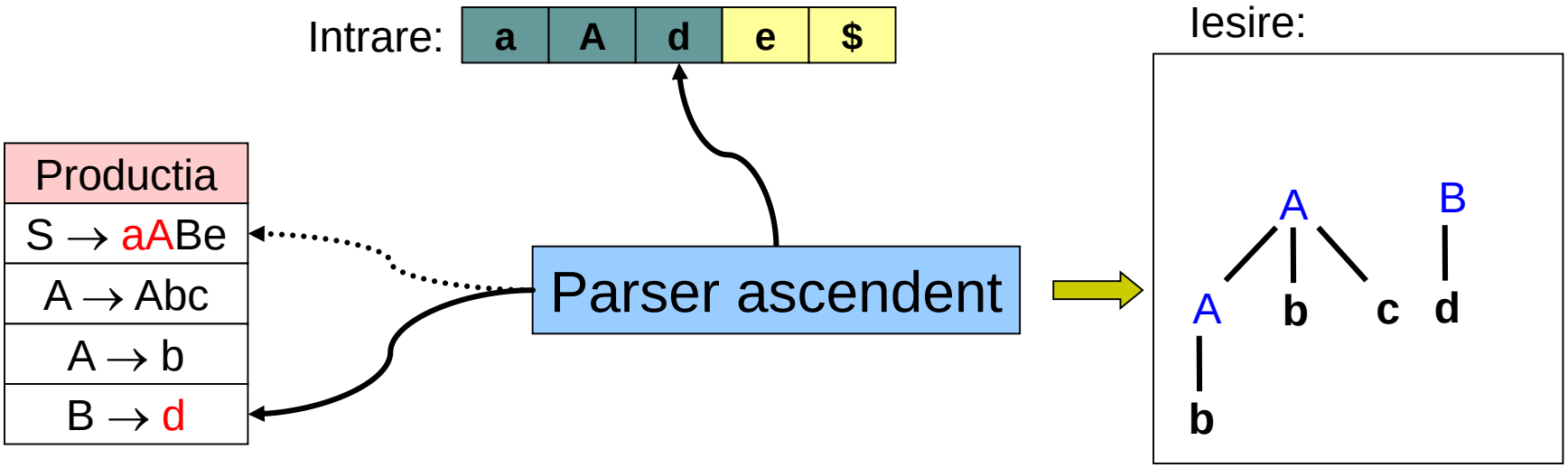
Exemplu de parser ascendent





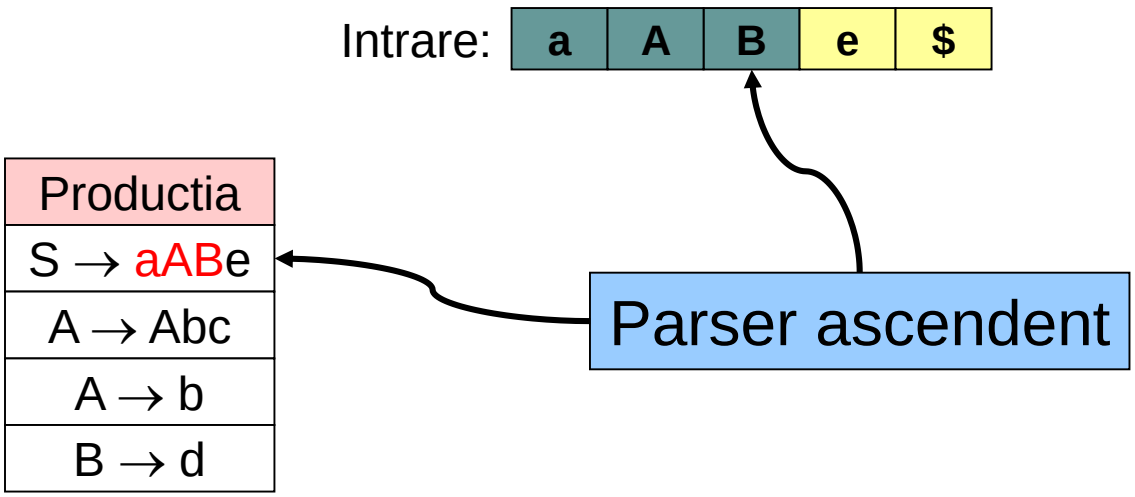
(Aho,Sethi,Ullman, pp. 195)

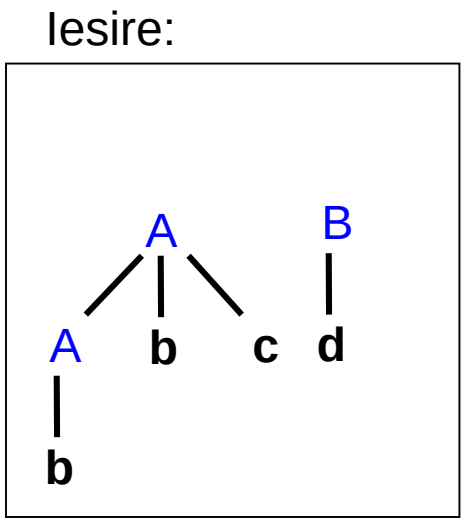
Exemplu de parser ascendent



(Aho,Sethi,Ullman, pp. 195)

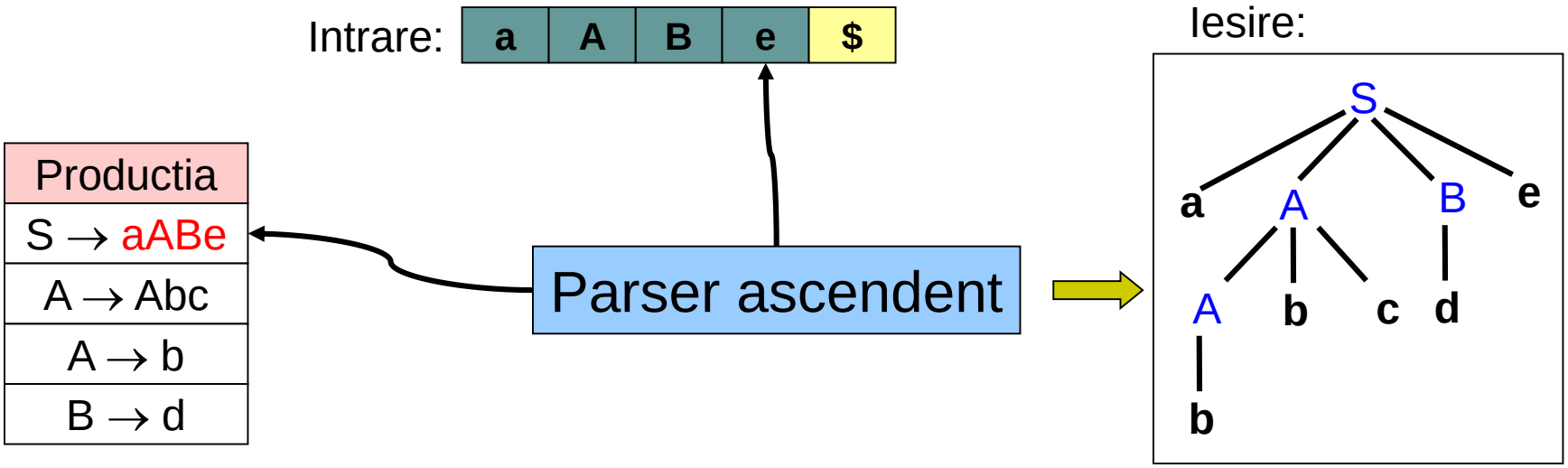
Exemplu de parser ascendent





(Aho,Sethi,Ullman, pp. 195)

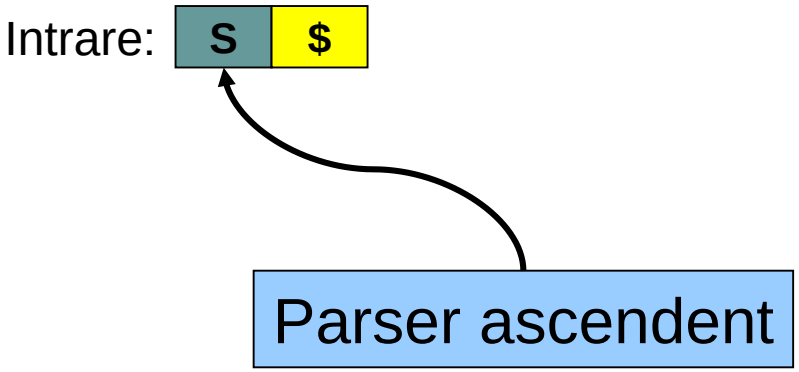
Exemplu de parser ascendent



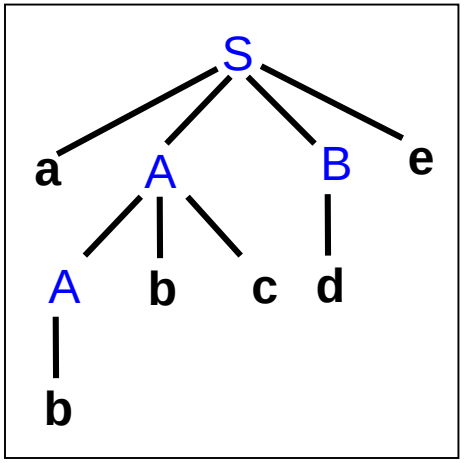
(Aho,Sethi,Ullman, pp. 195)

Exemplu de parser ascendent

Productia
S → aABe
A → Abc
A → b
B → d



Iesire:



Acest parser este cunoscut ca Parser LR fiindca scaneaza Intrarea "Left to right", si construieste "Rightmost derivation" in ordine inversa.

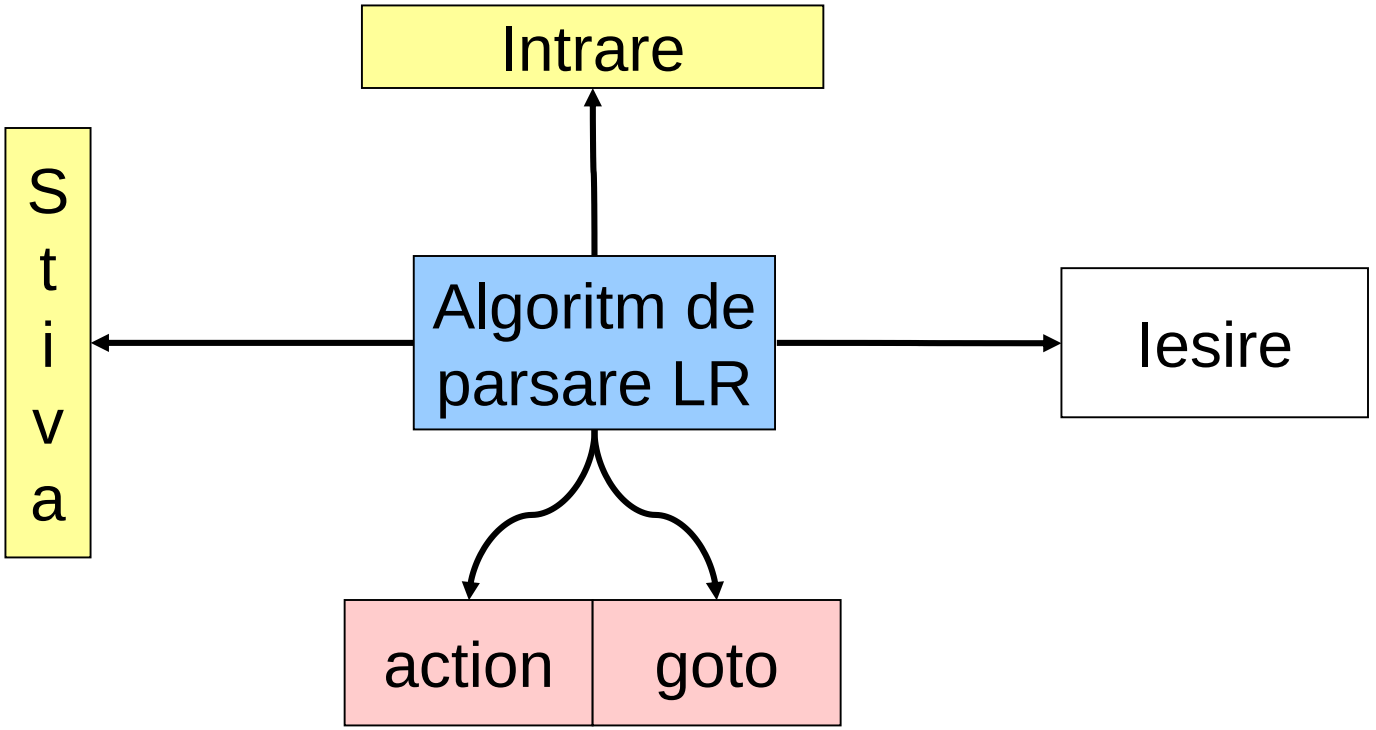
(Aho,Sethi,Ullman, pp. 195)

Exemplu de parser ascendent

Scanarea Productiilor pentru a detecta potrivirea cu subsiruri din Intrare, si backtrackingul face metoda din exemplul precedent foarte ineficienta.

Se poate mai bine?

Exemplu de parser LR



(Aho,Sethi,Ullman, pp. 217)

Exemplu de parser LR

Urmatoarea GRAMATICA:

(1) E → E + T
(2) E → T
(3) T → T * F
(4) T → F
(5) F → (E)
(6) F → **id**

Poate fi parsata cu urmatoarele tabele ‘action’ si ‘goto’

State	action						goto		
	id	**+**	*****	**(**	**)**	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

(Aho,Sethi,Ullman, pp. 219)

GRAMATICA:

(1) E → E + T
(2) E → T
(3) T → T * F
(4) T → F
(5) F → (E)
(6) F → **id**

Exemplu de parser LR

Intrare: | **id** | + | **id** | * | **id** | **$** |

Iesire:

Stiva: 0

LR Parsing Program

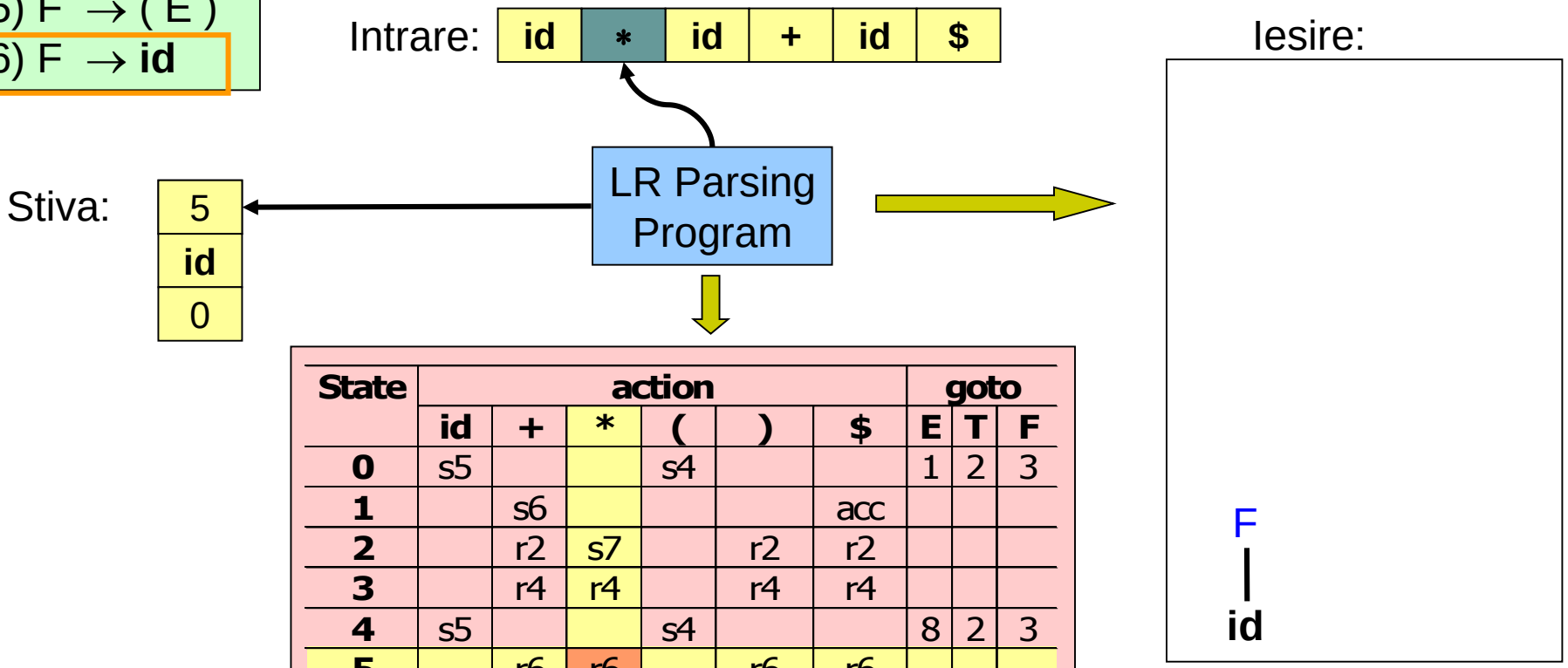
State	action						goto		
	id	**+**	*****	**(**	**)**	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

(Aho,Sethi,Ullman, pp. 220)

GRAMATICA:

(1) E → E + T
(2) E → T
(3) T → T * F
(4) T → F
(5) F → (E)
(6) F → **id**

Exemplu de parser LR



State	action						goto		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

(Aho,Sethi,Ullman, pp. 220)

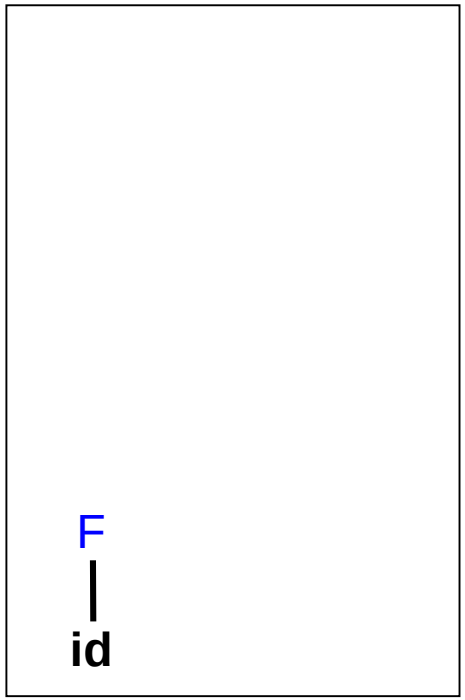
GRAMATICA:

(1) E → E + T
(2) E → T
(3) T → T * F
(4) T → F
(5) F → (E)
(6) F → **id**

Exemplu de parser LR

Intrare: | **id** | * | **id** | + | **id** | **$** |

Iesire:



Stiva: 0

LR Parsing Program

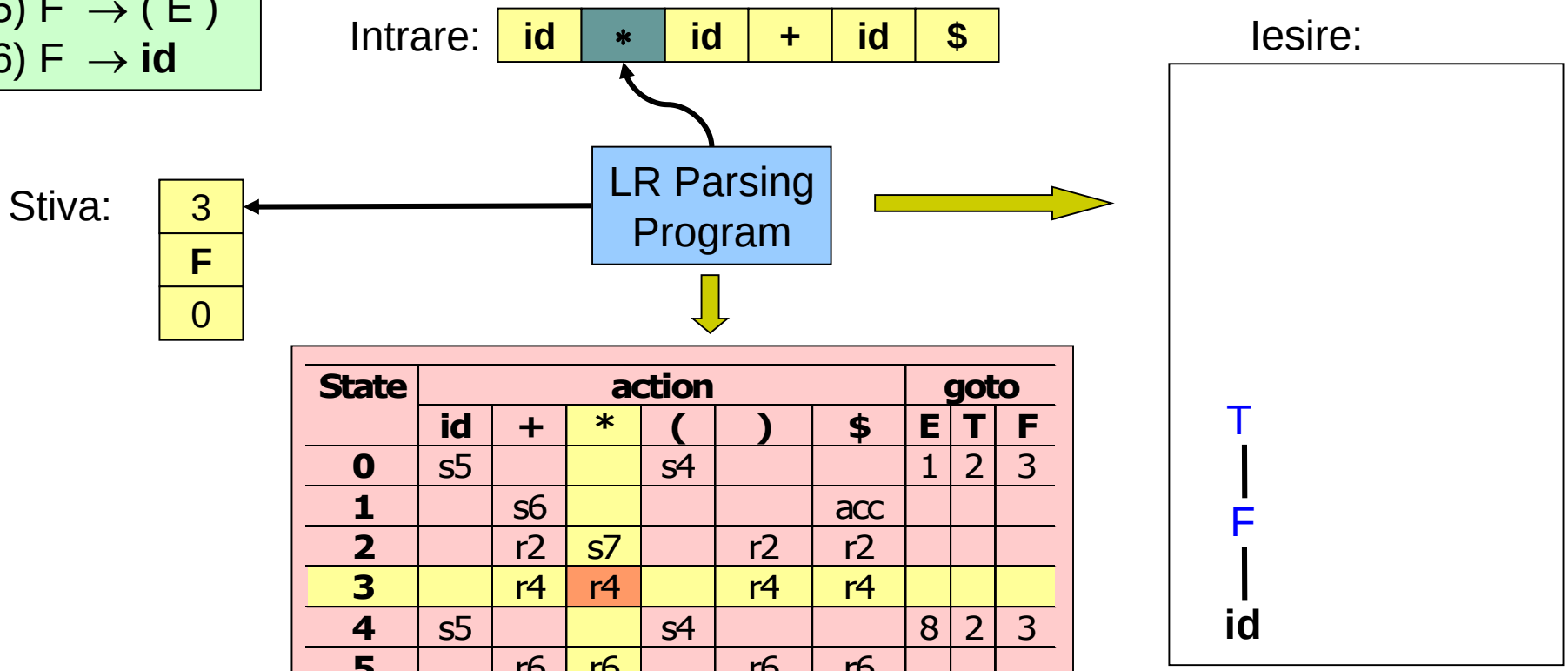
State	action						goto		
	id	**+**	*	**(**	**)**	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

(Aho,Sethi,Ullman, pp. 220)

GRAMATICA:

(1) E → E + T
(2) E → T
(3) T → T * F
(4) T → F
(5) F → (E)
(6) F → **id**

Exemplu de parser LR



State	action						goto		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

(Aho,Sethi,Ullman, pp. 220)

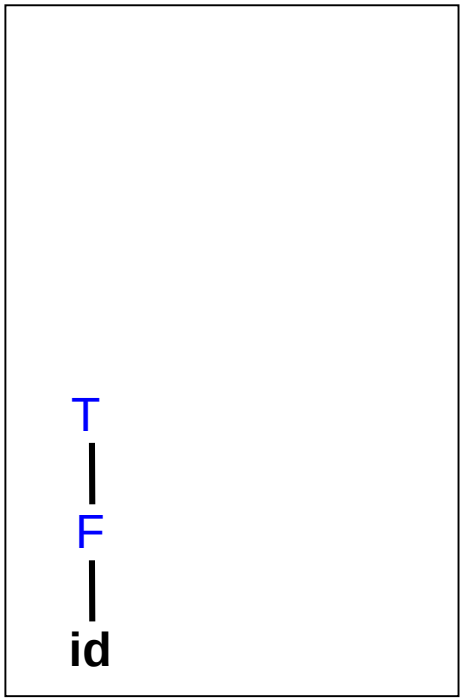
GRAMATICA:

(1) E → E + T
(2) E → T
(3) T → T * F
(4) T → F
(5) F → (E)
(6) F → **id**

Exemplu de parser LR

Intrare: | **id** | * | **id** | + | **id** | **$** |

Iesire:



Stiva: 0

LR Parsing Program

State	action						goto		
	id	**+**	*****	**(**	**)**	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

(Aho,Sethi,Ullman, pp. 220)

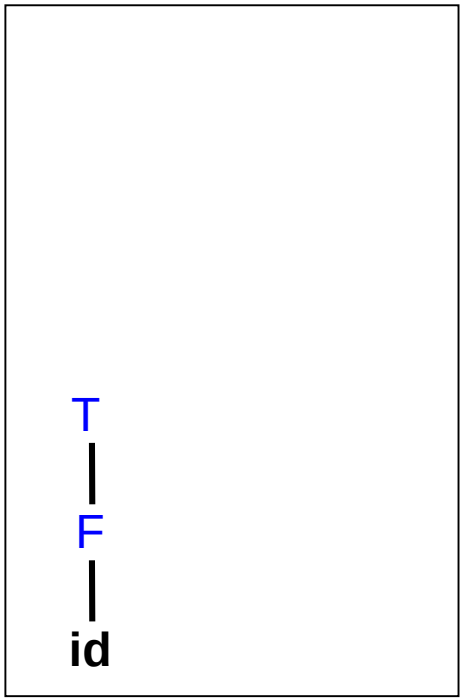
GRAMATICA:

(1) E → E + T
(2) E → T
(3) T → T * F
(4) T → F
(5) F → (E)
(6) F → **id**

Exemplu de parser LR

Intrare: | **id** | * | **id** | + | **id** | **$** |

Ieșire:



Stiva:

2
T
0

LR Parsing Program

State	action						goto		
	id	**+**	*	**(**	**)**	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

(Aho,Sethi,Ullman, pp. 220)

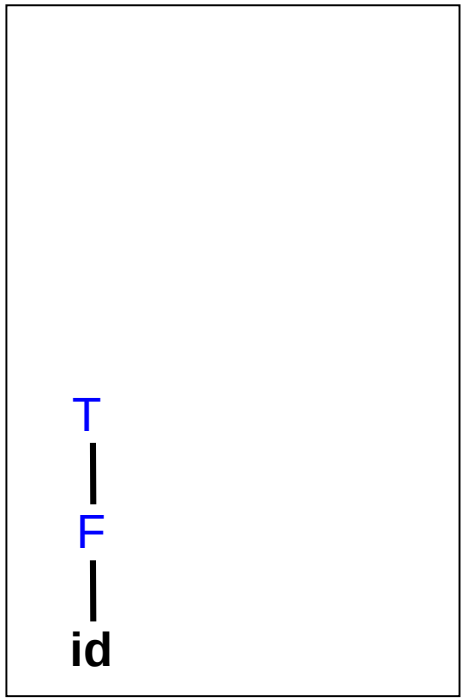
GRAMATICA:

(1) E → E + T
(2) E' → T
(3) T → T * F
(4) T → F
(5) F → (E)
(6) F → **id**

Exemplu de parser LR

Intrare: | **id** | * | **id** | + | **id** | **$** |

Iesire:



Stiva:

7
*
2
T
0

LR Parsing Program

State	action						goto		
	id	**+**	*****	**(**	**)**	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

(Aho,Sethi,Ullman, pp. 220)

GRAMATICA:

(1) E → E + T
(2) E' → T
(3) T → T * F
(4) T → F
(5) F → (E)
(6) F → **id**

Exemplu de parser LR

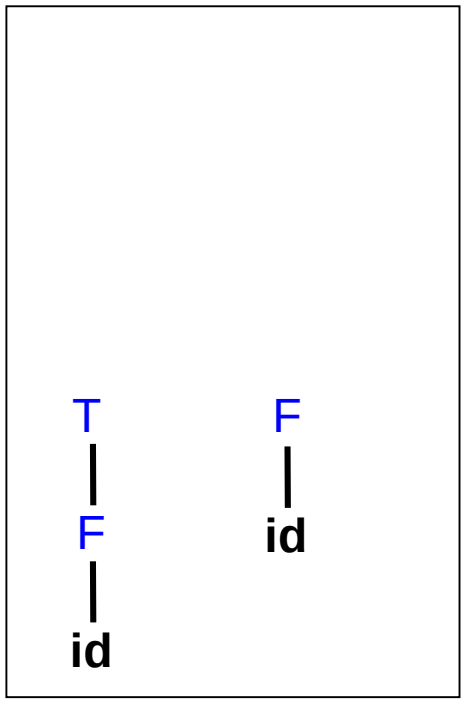
Intrare: | **id** | * | **id** | + | **id** | **$** |

Iesire:

Stiva:

5
id
7
*
2
T
0

LR Parsing Program



State	action						goto		
	id	**+**	*	**(**	**)**	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

(Aho,Sethi,Ullman, pp. 220)

GRAMATICA:

(1) E → E + T
(2) E' → T
(3) T → T * F
(4) T → F
(5) F → (E)
(6) F → **id**

Exemplu de parser LR

Intrare: | **id** | * | **id** | + | **id** | **$** |

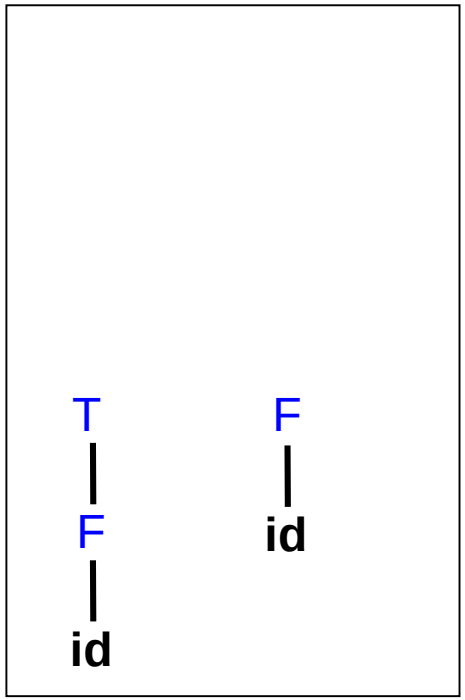
Stiva:

7
*
2
T
0

LR Parsing Program

State	action						goto		
	id	**+**	*****	**(**	**)**	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Iesire:



(Aho,Sethi,Ullman, pp. 220)

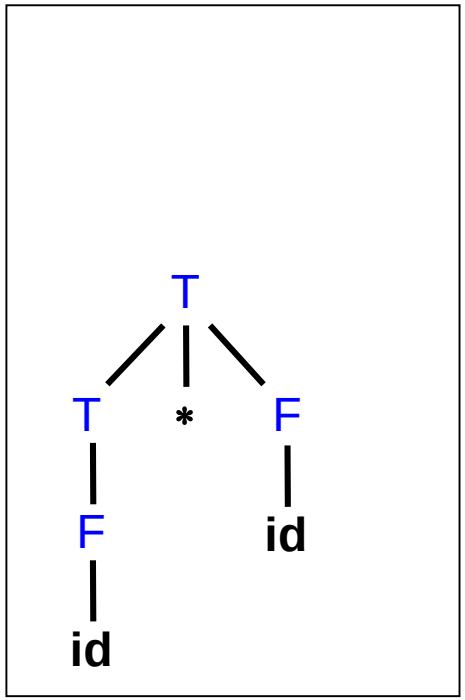
GRAMATICA:

(1) E → E + T
(2) E' → T
(3) T → T * F
(4) T → F
(5) F → (E)
(6) F → **id**

Exemplu de parser LR

Intrare: | **id** | * | **id** | + | **id** | **$** |

Ieșire:



Stiva:

10
F
7
*
2
T
0

LR Parsing Program

State	action						goto		
	id	**+**	*	(	)	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

(Aho,Sethi,Ullman, pp. 220)

GRAMATICA:

(1) E → E + T
(2) E → T
(3) T → T * F
(4) T → F
(5) F → (E)
(6) F → **id**

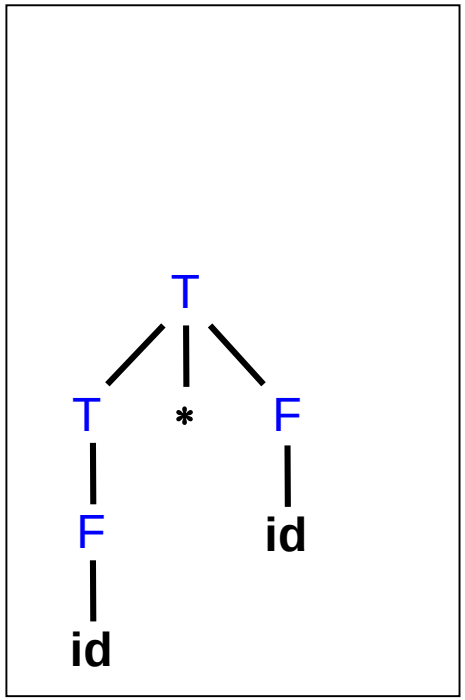
Exemplu de parser LR

Intrare: | **id** | * | **id** | + | **id** | **$** |

Stiva: 0

LR Parsing Program

Iesire:



State	action						goto		
	id	**+**	*****	**(**	**)**	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

(Aho,Sethi,Ullman, pp. 220)

GRAMATICA:

(1) E → E + T
(2) E → T
(3) T → T * F
(4) T → F
(5) F → (E)
(6) F → **id**

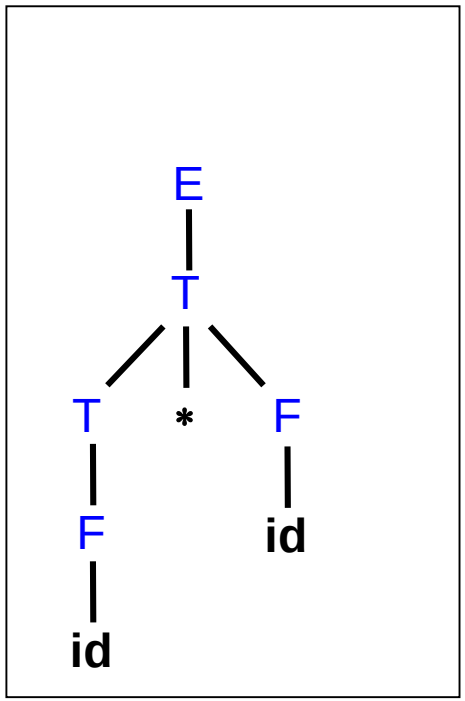
Exemplu de parser LR

Intrare: | **id** | * | **id** | + | **id** | **$** |

Stiva: | 2 | **T** | 0 |

LR Parsing Program

Iesire:



State	action						goto		
	id	**+**	*	**(**	**)**	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

(Aho,Sethi,Ullman, pp. 220)

GRAMATICA:

(1) E → E + T
(2) E → T
(3) T → T * F
(4) T → F
(5) F → (E)
(6) F → **id**

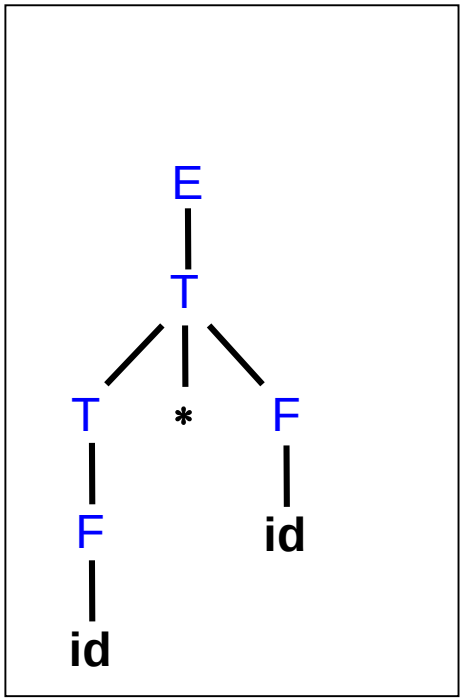
Exemplu de parser LR

Intrare: | **id** | * | **id** | + | **id** | **$** |

Stiva: 0

LR Parsing Program

Iesire:



State	action						goto		
	id	**+**	*****	**(**	**)**	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

(Aho,Sethi,Ullman, pp. 220)

GRAMATICA:

(1) E → E + T
(2) E' → T
(3) T → T * F
(4) T → F
(5) F → (E)
(6) F → **id**

Exemplu de parser LR

Intrare: | **id** | * | **id** | **+** | **id** | **$** |

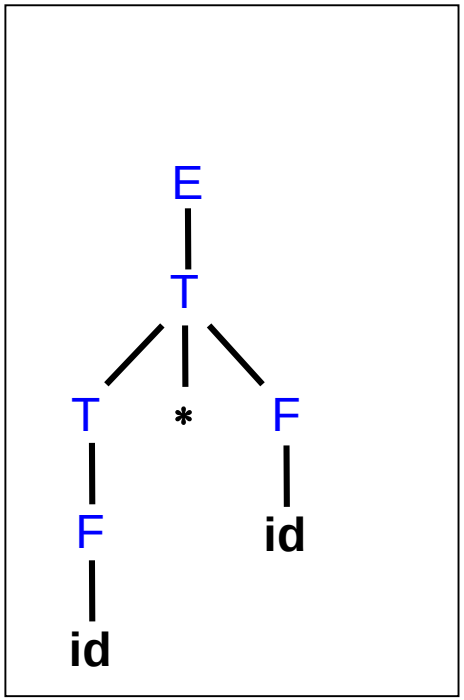
Stiva:

1
E
0

LR Parsing Program

State	action						goto		
	id	**+**	*	**(**	**)**	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Iesire:



(Aho,Sethi,Ullman, pp. 220)

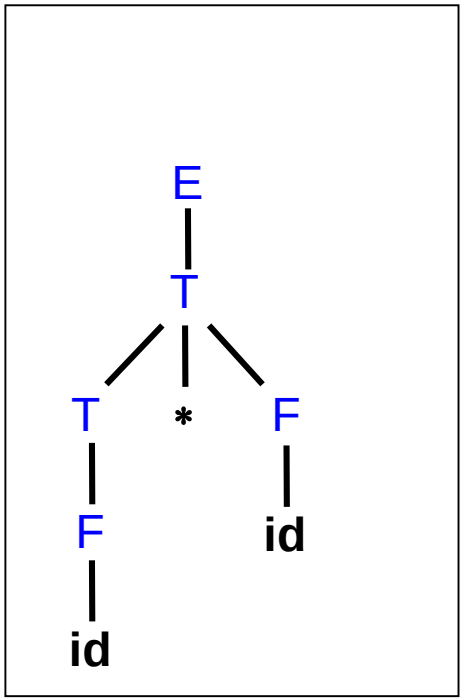
GRAMATICA:

(1) E → E + T
(2) E' → T
(3) T → T * F
(4) T → F
(5) F → (E)
(6) F → **id**

Exemplu de parser LR

Intrare: | **id** | * | **id** | + | **id** | **$** |

Ieșire:



Stiva:

6
+
1
E
0

LR Parsing Program

State	action						goto		
	id	**+**	*****	**(**	**)**	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

(Aho,Sethi,Ullman, pp. 220)

GRAMATICA:

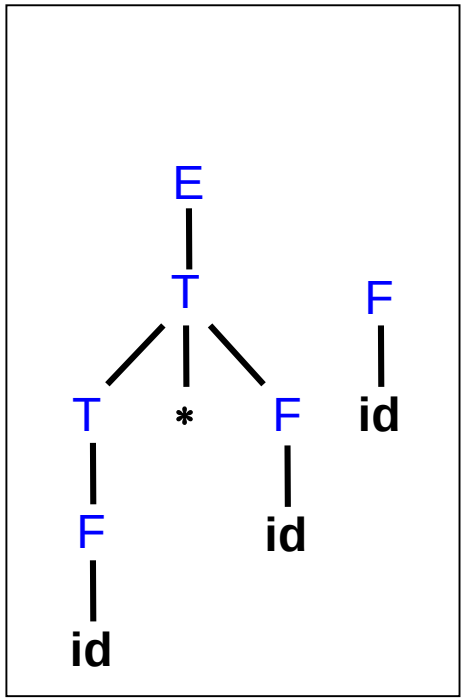
(1) E → E + T
(2) E' → T
(3) T → T * F
(4) T → F
(5) F → (E)
(6) F → **id**

Exemplu de parser LR

Intrare:

id	*	**id**	+	**id**	**$**

Iesire:



Stiva:

5
id
6
+
1
E
0

LR Parsing Program

State	action						goto		
	id	**+**	*	**(**	**)**	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

(Aho,Sethi,Ullman, pp. 220)

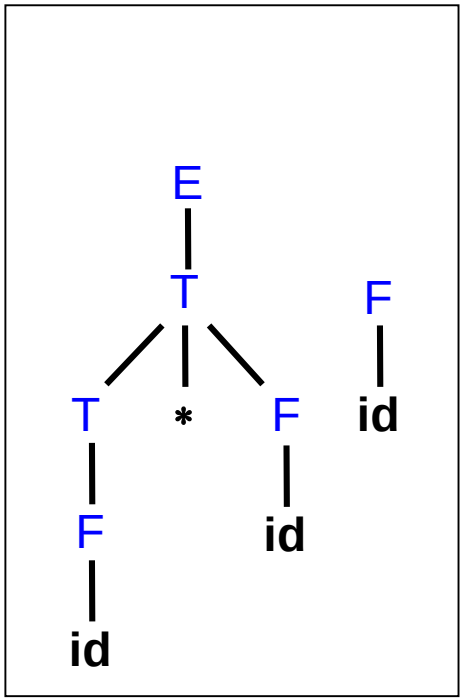
GRAMATICA:

(1) E → E + T
(2) E' → T
(3) T → T * F
(4) T → F
(5) F → (E)
(6) F → **id**

Exemplu de parser LR

Intrare: | **id** | * | **id** | + | **id** | **$** |

Iesire:



Stiva:

6
+
1
E
0

LR Parsing Program

State	action						goto		
	id	**+**	*****	**(**	**)**	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

(Aho,Sethi,Ullman, pp. 220)

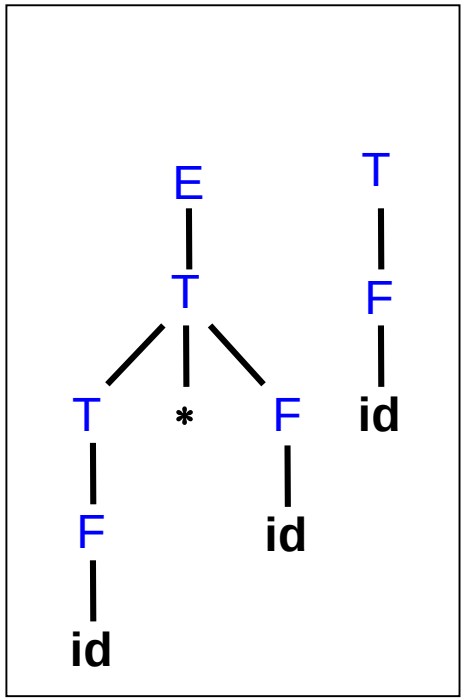
GRAMATICA:

(1) E → E + T
(2) E' → T
(3) T → T * F
(4) T → F
(5) F → (E)
(6) F → **id**

Exemplu de parser LR

Intrare: | **id** | * | **id** | + | **id** | **$** |

Ieșire:



Stiva:

3
F
6
+
1
E
0

LR Parsing Program

State	action						goto		
	id	**+**	*	**(**	**)**	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

(Aho,Sethi,Ullman, pp. 220)

GRAMATICA:

(1) E → E + T
(2) E' → T
(3) T → T * F
(4) T → F
(5) F → (E)
(6) F → **id**

Exemplu de parser LR

Intrare: | **id** | * | **id** | + | **id** | **$** |

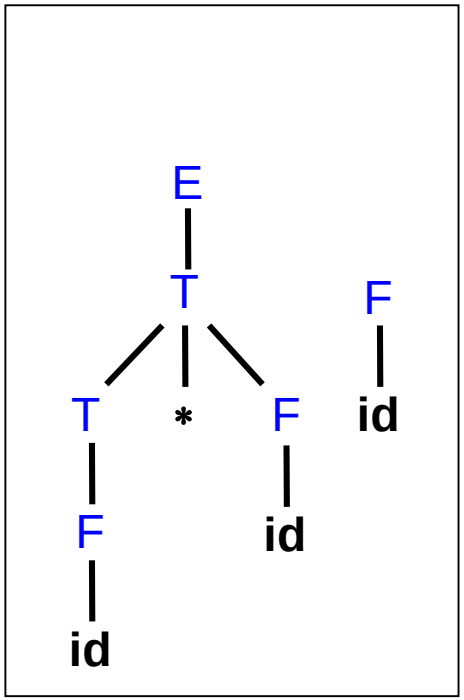
Stiva:

6
+
1
E
0

LR Parsing Program

State	action						goto		
	id	**+**	*****	**(**	**)**	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Iesire:



(Aho,Sethi,Ullman, pp. 220)

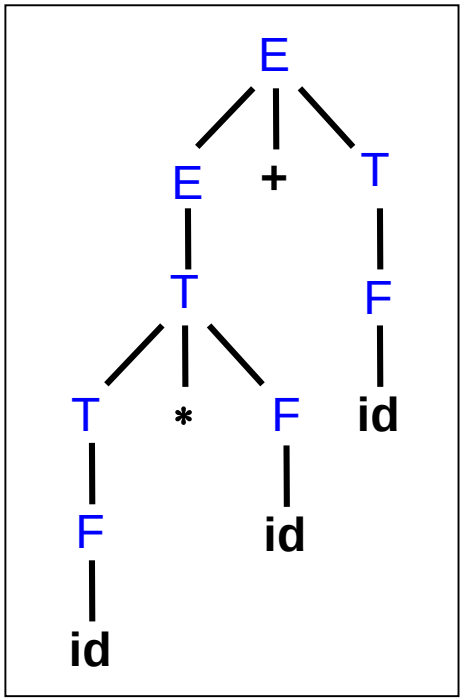
GRAMATICA:

(1) E → E + T
(2) E' → T
(3) T → T * F
(4) T → F
(5) F → (E)
(6) F → **id**

Exemplu de parser LR

Intrare: | **id** | * | **id** | + | **id** | **$** |

Iesire:



Stiva:

9
T
6
+
1
E
0

LR Parsing Program

State	action						goto		
	id	**+**	*	**(**	**)**	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

(Aho,Sethi,Ullman, pp. 220)

GRAMATICA:

(1) E → E + T
(2) E → T
(3) T → T * F
(4) T → F
(5) F → (E)
(6) F → **id**

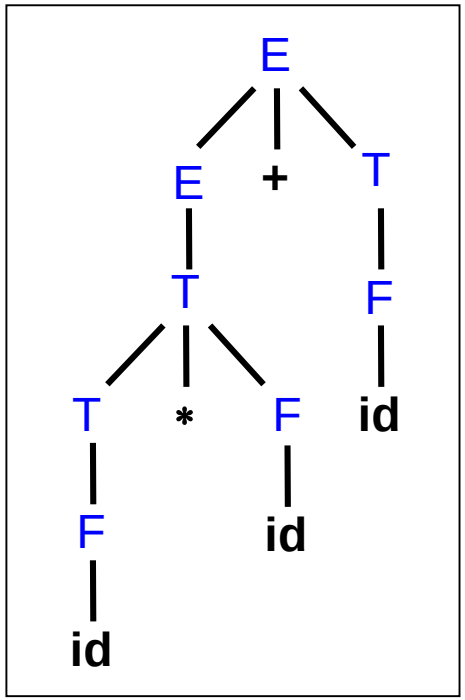
Exemplu de parser LR

Intrare: | **id** | * | **id** | + | **id** | **$** |

Stiva: 0

LR Parsing Program

Iesire:



State	action						goto		
	id	**+**	*****	**(**	**)**	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

(Aho,Sethi,Ullman, pp. 220)

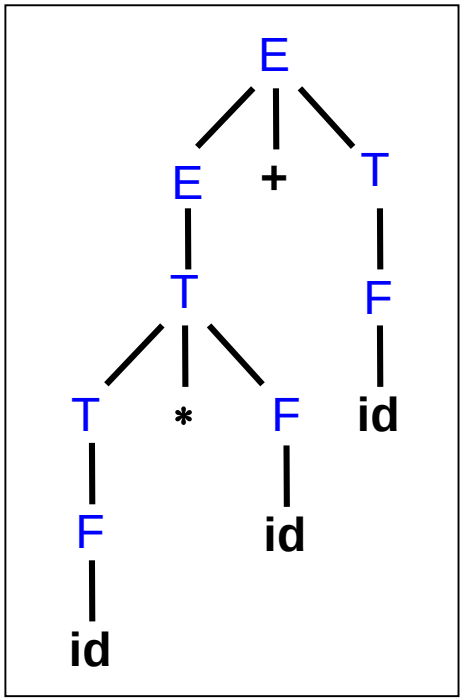
GRAMATICA:

(1) E → E + T
(2) E' → T
(3) T → T * F
(4) T → F
(5) F → (E)
(6) F → **id**

Exemplu de parser LR

Intrare: | **id** | * | **id** | + | **id** | **$** |

Ieșire:



Stiva:

1
E
0

LR Parsing Program

State	action						goto		
	id	**+**	*****	**(**	**)**	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

(Aho,Sethi,Ullman, pp. 220)

Construirea tabelelor de parsare

Toate parserele LR folosesc acelasi algoritm pe care l-am aratat in slide-urile precedente.

Diferentierea intre ele se face prin tabelele action si goto

Simple LR (SLR): merge pe cele mai putine GRAMATICI, dar e cel mai usor de implementat (AhoSethiUllman pp. 221-230).

Canonical LR: merge pe cele mai multe GRAMATICI, dar e cel mai greu de implementat. Imparte starile cand e necesar pentru a preveni reduceri care ar bloca parserul. (AhoSethiUllman pp. 230-236).

Lookahead LR (LALR): merge pe majoritatea constructiilor sintactice folosite in limbajele de programare, dar produce tabele mult mai mici decat Canonical LR.

(AhoSethiUllman pp. 236-247).

Analiza sintactica Shift-Reduce

- Actiunile analizorului : o secventa de operatii shift si reduce
- Starea analizorului : O stiva de terminali si neterminali, si o stiva de stari
- Pasul curent in analiza e dat atat de stiva cat si de banda de intrare

Actiuni Shift-Reduce

- Parsarea e o secventa de actiuni shift si reduce
- Shift: muta token-ul de look-ahead pe stiva

stiva	intrare	actiune
(	1+2+(3+4))+5	shift 1
(1	+2+(3+4))+5	

- Reduce: Inlocuieste simbolurile β din varful stivei cu simbolul neterminal X din partea stanga a productiei $X \rightarrow \beta$ (adica: pop β, push X)

stiva	intrare	actiune
(S+E	+(3+4))+5	reduce S → S+ E
(S	+(3+4))+5	

Analiza Shift-Reduce

$S \rightarrow S + E \mid E$
$E \rightarrow num \mid (S)$

derivarea	stiva	sir de intrare	actiune
(1+2+(3+4))+5		(1+2+(3+4))+5	shift
(1+2+(3+4))+5	(	1+2+(3+4))+5	shift
(1+2+(3+4))+5	(1	+2+(3+4))+5	reduce E→num
(E+2+(3+4))+5	(E	+2+(3+4))+5	reduce S→E
(S+2+(3+4))+5	(S	+2+(3+4))+5	shift
(S+2+(3+4))+5	(S+	2+(3+4))+5	shift
(S+2+(3+4))+5	(S+2	+(3+4))+5	reduce E→num
(S+E+(3+4))+5	(S+E	+(3+4))+5	reduce S→S+E
(S+(3+4))+5	(S	+(3+4))+5	shift
(S+(3+4))+5	(S+	(3+4))+5	shift
(S+(3+4))+5	(S+(	3+4))+5	shift
(S+(3+4))+5	(S+(3	+4))+5	reduce E→ num

...

Probleme (selectarea actiunii)

- De unde stim ce actiune sa aplicam: shift sau reduce? Si cu ce regula sa reducem?
  - Uneori putem reduce dar nu e bine sa o facem – am ajunge cu analiza intr-un punct mort (sau nu am respecta precedenta operatorilor)
  - Uneori putem reduce stiva in mai multe feluri, nu intr-un singur fel

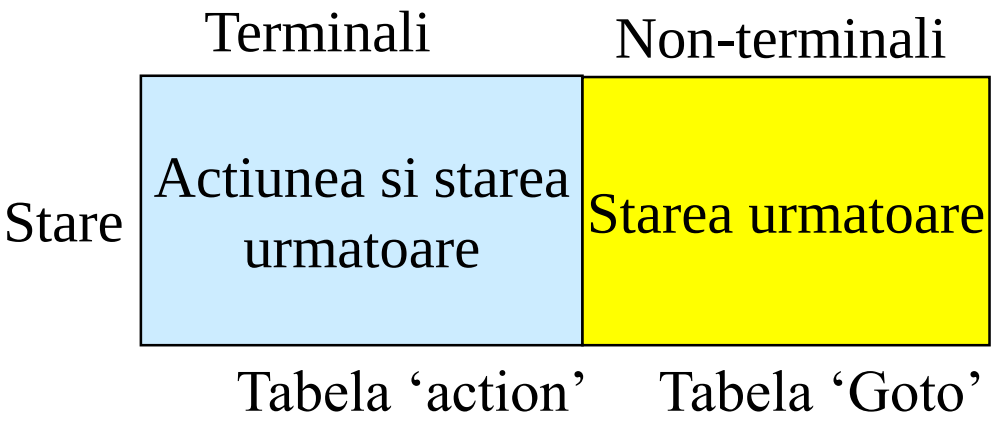
Selectarea actiunii

- Starea curenta:
  - stiva β
  - simbolul look-ahead b
  - exista productia X → γ, si stiva e de forma $\beta = \alpha\gamma$
- Ar trebui analizorul sa:
  - Shifteze b pe stiva, trasformand-o in βb ?
  - Reduca aplicand productia X → γ, presupunand ca stiva e de forma $\beta = \alpha\gamma$ - si sa o transforme astfel in αX ?
- Decizie in functie de b şi de prefixul α
  - α e diferit pentru productii diferite, deoarece partea dreapta a productiilor(γ-urile) poate avea lungimi diferite

Algoritmul de parsare LR

- Mecanismul de baza
  - Folosim un set de stari ale parser-ului
  - Folosim o stiva de stari (eventual, alternam simboluri cu stari)
    - De ex., 1 (6 S 10 + 5 (verde = stari)
  - Folosim tabela de parsare pentru:
    - A determina ce actiune se aplica (shift/reduce)
    - A determina starea urmatoare
- Actiunile analizorului pot fi determinate cu exactitate din tabelele de parsare

Tabela de parsare LR



- Algoritm: ne uitam la starea curenta S si terminalul C de la intrare
  - Daca Action[S,C] = s(S’) atunci ‘shift’ si trece in S’:
    - push(C), push(S’)
  - Daca Action[S,C] = X→ α atunci ‘reduce’:
    - pop(2*|α|), S’= top(), push(X), push(Goto[S’,X])

Gramatici LR(k)

- LR(k) = Left-to-right scanning, right-most derivation, k lookahead tokens
- Cazurile principale
  - LR(0), LR(1)
  - Variante: SLR and LALR(1)
- Analizoarele pt. gramatici LR(0):
  - Aleg shift/reduce fara sa se uite la lookahead
  - Incepem cu ele, caci ne vor ajuta sa intelegem parsarea shift-reduce

Cursul viitor:

- Analiza semantică

Întrebări?