Course 9

LR(k) Parsing (cont.)

LR(k) parsing:
LR(0), SLR, LR(1), LALR

- Define item
- Construct set of states
- Construct table

Executed 1 time

- Parse sequence based on moves between configurations

2. Construct set of states

- What a state contains – Algorithm *closure_LR(0)*
- How to move from a state to another – Function *goto_LR(0)*
- Construct set of states – Algorithm *ColCan_LR(0)*

Canonical collection

Algorithm *ColCan_LR(0)*

INPUT: G'- gramatica îmbogăţită
OUTPUT: C - colecţia canonică de stări

$\mathcal{C} := \emptyset;$
$s_0 := closure(\{[S' \to .S]\})$ // state corresponding to prod. of S' = initial state
$\mathcal{C} := \mathcal{C} \cup \{s_0\};$ //initialize collection with s_0
repeat
  for $\forall s \in \mathcal{C}$ **do**
    for $\forall X \in N \cup \Sigma$ **do**
      if $goto(s, X) \neq \emptyset$ and $goto(s, X) \notin \mathcal{C}$ **then**
        $\mathcal{C} = \mathcal{C} \cup goto(s, X)$ //add new state
      end if
    end for
  end for
until $\mathcal{C}$ nu se mai modifică

Algorithm *Closure*

I = LR(0) item of the form [A->$\boldsymbol{\alpha}.\boldsymbol{\beta}$]

INPUT: I-element de analiză; G'- gramatica îmbogăţită
OUTPUT: C = closure(I);
$C := \{I\};$ //initialize Closure with the LR(0) item
repeat
  for $\forall [A \to \alpha . B\beta] \in C$ **do** //search productions with dot in front of nonterminal
    for $\forall B \to \gamma \in P$ **do** //search productions of that nonterminal
      if $[B \to .\gamma] \notin C$ **then**
        $C = C \cup [B \to .\gamma]$ //adds item formed from production with dot in //front of right hand side of the production
      end if
    end for
  end for
until C nu se mai modifică

Function *goto*

goto : $P(\mathcal{E}_0) \times (N \cup \Sigma) \rightarrow P(\mathcal{E}_0)$ //creates new states

where $\mathcal{E}_0$ = set of LR(0) items

goto(s, X) = closure($\{[A \rightarrow \alpha X.\beta] \mid [A \rightarrow \alpha.X\beta] \in s\}$)

goto(s,X): in state **s**, search LR(0) item that has dot in front of symbol **X**. Move the dot after symbol **X** and call closure for this new item.

SLR Parser

Prediction = next symbols on input sequence

- SLR = Simple LR

- Remark:

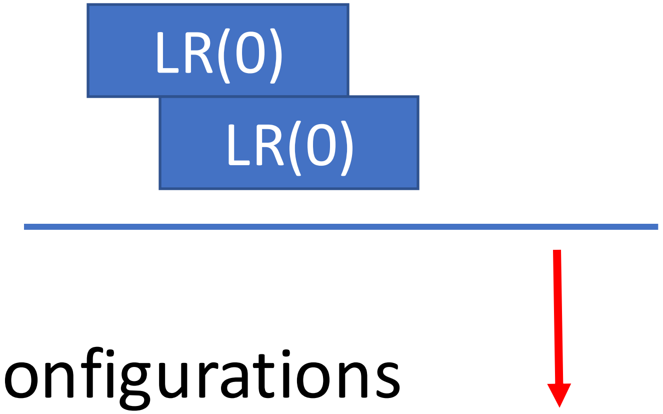
  LR(0) – lots of conflicts – solved if considering prediction

=>

1. LR(0) canonical collection of states– prediction of length 0
2. Table and parsing sequence – prediction of length 1

SLR Parsing:

- define item
- Construct set of states
- Construct table
- Parse sequence based on moves between configurations



Construct SLR table

Remarks:

1. Prediction = next symbol from input sequence => FOLLOW

   - see LL(1)

2. Structure – LR(k):
   - Lines - states
   - action + goto

action – a column for each prediction $\in \Sigma$

goto – a column for each symbol $X \in N \cup \Sigma$

Optimize table structure: merge *action* and *goto* columns for Σ

Remark (LR(0) table):

- *if* s is accept state *then* goto(s, X) = $\emptyset$, $\forall X \in N \cup \Sigma$.
- *If* in state ***s*** action is reduce *then* goto(s, X) = $\emptyset$, $\forall X \in N \cup \Sigma$.

SLR table

And goto

	Action		GOTO	
	a_1 …	a_n	B_1 …	B_m
s_0				
s_1				
…				
s_k				

$a_1,\ldots,a_n \in \Sigma$
$B_1,\ldots,B_m \in N$
$s_0,\ldots,s_k$ - states

Rules for SLR table

1. If $[A \rightarrow \alpha.\beta] \in s_i$ and $goto(s_i, a) = s_j$ then **action(s_i,a)**=**shift s_j**

   // dot is not at the end

2. if $[A \rightarrow \beta.] \in s_i$ and $A \neq S'$ then **action(s_i,u)**=**reduce l**, where l – number of production $A \rightarrow \beta$, $\forall u \in FOLLOW(A)$

   //dot is at the end, but not for S'

3. if $[S' \rightarrow S.] \in s_i$ then **action(s_i,\$)**=**acc**

   // dot is at the end, prod. of S'

4. if $goto(s_i, X) = s_j$ then **goto(s_i, X) = s_j**, $\forall X \in N$

5. otherwise **error**

Remarks

1. Similarity with LR(0)

2. A grammar is SLR if the SLR table does not contain conflicts (more than one value in a cell)

Parsing sequences

- INPUT:
  - Grammar G' = (NU{S'}, $\mathbf{\Sigma}$, P U {S'->S},S')
  - SLR table
  - Input sequence w =$a_1 \ldots a_n$
- OUTPUT:

  if (w ∈L(G)) *then* **string of productions**

  else **error & location of error**

SLR = LR(0) configurations

$$(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\pi})$$

where:

- α = working stack
- β = input stack
- π = output (result)

Initial configuration:
($\$s_0$,w\$,ε)

Final configuration:
($\$s_{acc}$, \$, π)

Moves

head(β) = prediction

1. **Shift**

if action(s_m,a_i)= shift s_j **then**

$(\$s_0x_1 \ldots x_m s_m, a_i \ldots a_n\$, \pi) \vdash (\$s_0x_1 \ldots x_m s_m a_i s_j, a_{i+1} \ldots a_n\$, \pi)$

2. **Reduce**

if action(s_m,a_i) = reduce t AND (t) $A \to x_{m-p+1} \ldots x_m$ AND goto(s_{m-p},A) = s_j

then

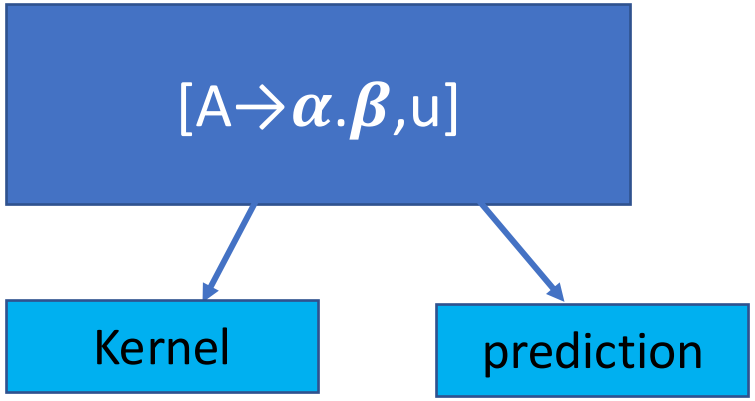
$(\$s_0 \ldots x_m s_m, a_i \ldots a_n\$, \pi) \vdash (\$s_0 \ldots x_{m-p} s_{m-p} A s_j, a_i \ldots a_n\$, t\,\pi)$

3. **Accept**

if action(s_m,\$) = accept **then** $(\$s_m, \$, \pi)$=acc

4. **Error** - otherwise

LR(1) Parser



1. Define item
2. Construct set of states
3. Construct table
4. Parse sequence based on moves between configurations

Construct LR(1) set of states

- Alg *ColCan_LR1*
- Function *goto_LR1*
- Alg *Closure_LR1*

Algorithm *ColCan_LR1*

INPUT: G' – enhanced grammar

OUTPUT: **C1** – cannonical collection of states

C1 =∅

S0 = *Closure_LR1*({[S'→.S,$]})

C1 := **C1** U {s_0}

Repeat

    for ∀s ∈ **C1 do**

        for ∀ X ∈ N U*Σ* **do**

            T = *goto_LR1*(s,X)

            if T≠ ∅ **and** T ∉ **C1 then**

                    C1 = **C1** U T

            endif

        endfor

    endfor

Until C1 *unchanged*

Function *goto_LR1*

Goto_LR1 : $P(\mathcal{E}_1) \times (N \cup \Sigma) \rightarrow P(\mathcal{E}_1)$

where $\mathcal{E}_1$ = set ofLR(1) items

Goto_LR1(s, X) = *Closure_LR1*$(\{[A \rightarrow \alpha X.\beta,u] \mid [A \rightarrow \alpha.X\beta,u] \in s\})$

Algorithm *Closure_LR1*

- [A → α.Bβ,u] valid for live prefix γα =>

$$S \overset{*}{\Rightarrow}_{dr} \gamma A w \Rightarrow_{dr} \gamma \alpha B \beta w$$

$$u = FIRST_k(w)$$

- [B → .δ, *smth*]∈P => $S \overset{*}{\Rightarrow} \gamma A w \Rightarrow_{dr} \gamma \alpha B \beta w \Rightarrow_{dr} \gamma \alpha \delta \beta w.$

=>

Algorithm *Closure_LR1*

INPUT: I-element de analiză; G'- gramatica îmbogăţită;
$FIRST(X), \forall X \in N \cup \Sigma$;
OUTPUT: C_1 = closure(I);
$C_1 := \{I\}$;
repeat
  for $\forall [A \to \alpha.B\beta, a] \in C_1$ **do**
    for $\forall B \to \gamma \in P$ **do**
      for $\forall b \in FIRST(\beta a)$ **do**
        if $[B \to .\gamma, b] \notin C_1$ **then**
          $C_1 = C_1 \cup [B \to .\gamma, b]$
        end if
      end for
    end for
  end for
until C_1 nu se mai modifică

Construct LR(1) table

- Structure – SLR
- Rules:

1. if $[A \rightarrow \alpha.\beta,u] \in s_i$ and $goto(s_i,a) = s_j$ then **action(s_i,a)=shift s_j**
2. if $[A \rightarrow \beta.,u] \in s_i$ and $A \neq S'$ then **action(s_i,u)=reduce l**, where l – number of production $A \rightarrow \beta$
3. if $[S' \rightarrow S.,\$] \in s_i$ then **action(s_i,\$)=acc**
4. if $goto(s_i, X) = s_j$ then **goto(s_i, X) = s_j** , $\forall X \in N$
5. otherwise = **error**

Remarks

1. A grammar is LR(1) if the LR(1) table does not contain conflicts

2. Number of states – significantly increase

4. Define configurations and moves

- INPUT:
  - Grammar G' = (NU{S'}, **Σ**, P U {S'->S},S')
  - LR(1) table
  - Input sequence w =$a_1 \ldots a_n$
- OUTPUT:

  if (w ∈L(G)) *then* **string of productions**

  else **error & location of error**

LR(1) configurations

$$(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\pi})$$

where:

- α = working stack
- β = input stack
- π = output (result)

Initial configuration:
($\$s_0$,w$,ε)

Final configuration:
($\$s_{acc}$, $, π)

Moves

head(β) = prediction

1. **Shift**

if action(s_m,a_i)= shift s_j **then**

$(\$s_0x_1 \ldots x_m s_m, a_i \ldots a_n\$, \pi) \vdash (\$s_0x_1 \ldots x_m s_m a_i s_j, a_{i+1} \ldots a_n\$, \pi)$

2. **Reduce**

if action(s_m,a_i) = reduce t AND (t) $A \rightarrow x_{m-p+1} \ldots x_m$ AND goto(s_{m-p},A) = s_j

then

$(\$s_0 \ldots x_m s_m, a_i \ldots a_n\$, \pi) \vdash (\$s_0 \ldots x_{m-p} s_{m-p} A s_j, a_i \ldots a_n\$, t\ \pi)$

3. **Accept**

if action(s_m,\$) = accept **then** $(\$s_m, \$, \pi)$=acc

4. **Error** - otherwise

LALR Parser

- LALR = Look Ahead LR(1)

- why?

LALR principle

$[A \rightarrow \alpha\beta., u] \in s_i$ apply reduce (k) then goto(s_i,A) =s_m
$[A \rightarrow \alpha\beta., v] \in s_j$ apply reduce (k) then goto(s_j,A) =s_n

$[A \rightarrow \alpha.\beta, u] \in s_i$

=> $[A \rightarrow \alpha.\beta, u|v] \in s_{i,j}$

$[A \rightarrow \alpha.\beta, v] \in s_j$

- Merge states with the same kernel, conserving all predictions, if **no conflict** is created

LALR Parsing

- Same as LR(1)
- Number of LALR states = number of SLR / LR(0) states

- How? - LR(1) states

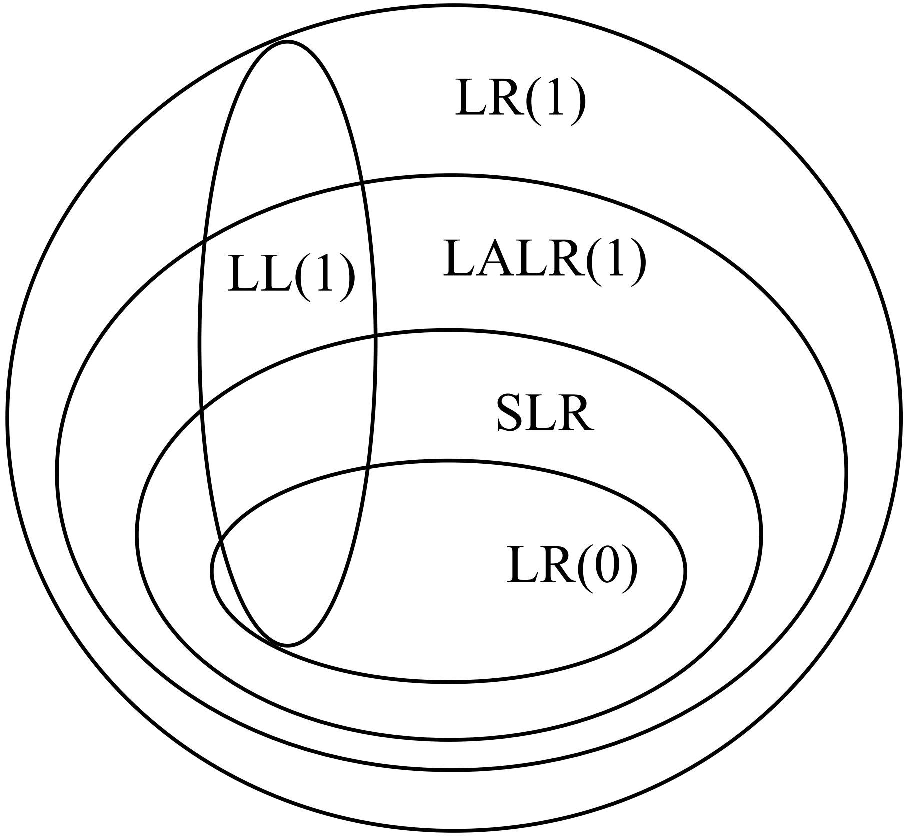
LR(k) Parsers

- LR(0):
  - Items ignore prediction
  - Reduce can be applied only in singular states (contain one item)
  - Lot of conflicts
- SLR:
  - Use same items as LR(0)
  - When reduce consider prediction
  - Eliminate several LR(0) conflicts (not all)
- LR(1):
  - Performant algorithm for set of states
  - Generate few conflicts
  - Generate lot of states
- LALR:
  - Merge LR(1) states ccorresponding to same kernel
  - Most used algorithm (most performant)

Parsing - recap

	Descendent	Ascendent
Recursive	Descendent recursive parser	Ascendent recursive parser
Linear	LL(1)	LR(0), SLR, LR(1), LALR

Parsing - recap



Push-Down Automata
(PDA)

Definition

- A push-down automaton (APD) is a 7-tuple $M = (Q,\boldsymbol{\Sigma},\boldsymbol{\Gamma},\boldsymbol{\delta},q_0,Z_0,F)$ where:
  - Q – finite set of states
  - $\boldsymbol{\Sigma}$ - alphabet (finite set of input symbols)
  - $\boldsymbol{\Gamma}$ – stack alphabet (finite set of stack symbols)
  - $\boldsymbol{\delta} : Q \times (\boldsymbol{\Sigma} \cup \{\boldsymbol{\varepsilon}\}) \times \boldsymbol{\Gamma} \rightarrow \mathcal{P}(Q \times \boldsymbol{\Gamma}^*)$ –transition function
  - $q_0 \in Q$ – initial state
  - $Z_0 \in \boldsymbol{\Gamma}$ – initial stack symbol
  - $F \subseteq Q$ – set of final states

Push-down automaton

Transition is determined by:

- Current state
- Current input symbol
- Head of stack

Reading head -> input band:

- Read symbol
- No action

Stack:

- Zero symbols => pop
- One symbol => push
- Several symbols => repeated push

Configurations and transition / moves

- Configuration:

$$(\mathbf{q}, \mathbf{x}, \boldsymbol{\alpha}) \in Q \times \boldsymbol{\Sigma}^* \times \boldsymbol{\Gamma}^*$$

where:

- PDA is in state **q**
- Input band contains **x**
- Head of stack is $\boldsymbol{\alpha}$

- Initial configuration (q_0, w, Z_0)

Configurations and moves(cont.)

- Moves between configurations:

p,q $\in$Q, a$\in\boldsymbol{\Sigma}$, Z $\in\boldsymbol{\Gamma}$, w $\in\boldsymbol{\Sigma}^*$,$\boldsymbol{\alpha}$,$\boldsymbol{\gamma}$ $\in\boldsymbol{\Gamma}^*$

(q,aw,Z$\boldsymbol{\alpha}$) $\vdash$ (p,w,$\boldsymbol{\gamma}$Z$\boldsymbol{\alpha}$) iff $\boldsymbol{\delta}$(q,a,Z) $\ni$ (p,$\boldsymbol{\gamma}$Z)

(q,aw,Z$\boldsymbol{\alpha}$) $\vdash$ (p,w, $\boldsymbol{\alpha}$) iff $\boldsymbol{\delta}$(q,a,Z) $\ni$ (p, $\boldsymbol{\varepsilon}$)

(q,aw,Z$\boldsymbol{\alpha}$) $\vdash$ (p,aw,$\boldsymbol{\gamma}$Z$\boldsymbol{\alpha}$) iff $\boldsymbol{\delta}$(q,$\boldsymbol{\varepsilon}$,Z) $\ni$ (p,$\boldsymbol{\gamma}$Z)

(${\boldsymbol{\varepsilon}}$-move)

- $\overset{k}{\vdash}$, $\overset{+}{\vdash}$, $\overset{*}{\vdash}$

Language accepted by PDA

- Empty stack principle:

$L_{\boldsymbol{\varepsilon}}(M) = \{w|w\in\boldsymbol{\Sigma}^*, (q_0,w,Z_0) \vdash^* (q,\boldsymbol{\varepsilon},\boldsymbol{\varepsilon}), q\in Q\}$

- Final state principle:

$L_f(M) = \{w|w\in\boldsymbol{\Sigma}^*, (q_0,w,Z_0) \vdash^* (q_f,\boldsymbol{\varepsilon},\boldsymbol{\gamma}), q_f\in F\}$

Representations

- Enumerate
- Table
- Graphic

Construct PDA

- $L = \{0^n 1^n \mid n \geq 1\}$
- States, stack, moves?

1. States:
   - Initial state:q_0 – beginning and process symbols ‘0’
   - When first symbol ‘1’ is found – move to new state => q_1
   - Final: final state q_2
2. Stack:
   - Z_0 – initial symbol
   - X – to count symbols:
     - When reading a symbol ’0’ – push X in stack
     - When reading a symbol ‘1’ – pop X from stack

Exemple 1 (enumerate)

$M = (\{q_0,q_1,q_2\}, \{0,1\}, \{Z_0,X\},\boldsymbol{\delta},q_0,Z_0,\{q_2\})$

$\boldsymbol{\delta}(q_0,0,Z_0) = (q_0,XZ_0)$

$\boldsymbol{\delta}(q_0,0,X) = (q_0,XX)$

$\boldsymbol{\delta}(q_0,1,X) = (q_1,\boldsymbol{\varepsilon})$

$\boldsymbol{\delta}(q_1,1,X) = (q_1,\boldsymbol{\varepsilon})$

$\cancel{\boldsymbol{\delta}(q_1,\boldsymbol{\varepsilon},Z_0) = (q_2,Z_0)}$

$\boxed{\boldsymbol{\delta}(q_1,\boldsymbol{\varepsilon},Z_0) = (q_1, \boldsymbol{\varepsilon})}$

Empty stack

$\vdash (q_1, \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon})$

$(q_0,0011,Z_0) \vdash (q_0,011,XZ_0) \vdash (q_0,11,XXZ_0) \vdash (q_1,1,XZ_0) \vdash (q_1, \boldsymbol{\varepsilon}, Z_0) \vdash (q_2, \boldsymbol{\varepsilon}, Z_0)$

Final state

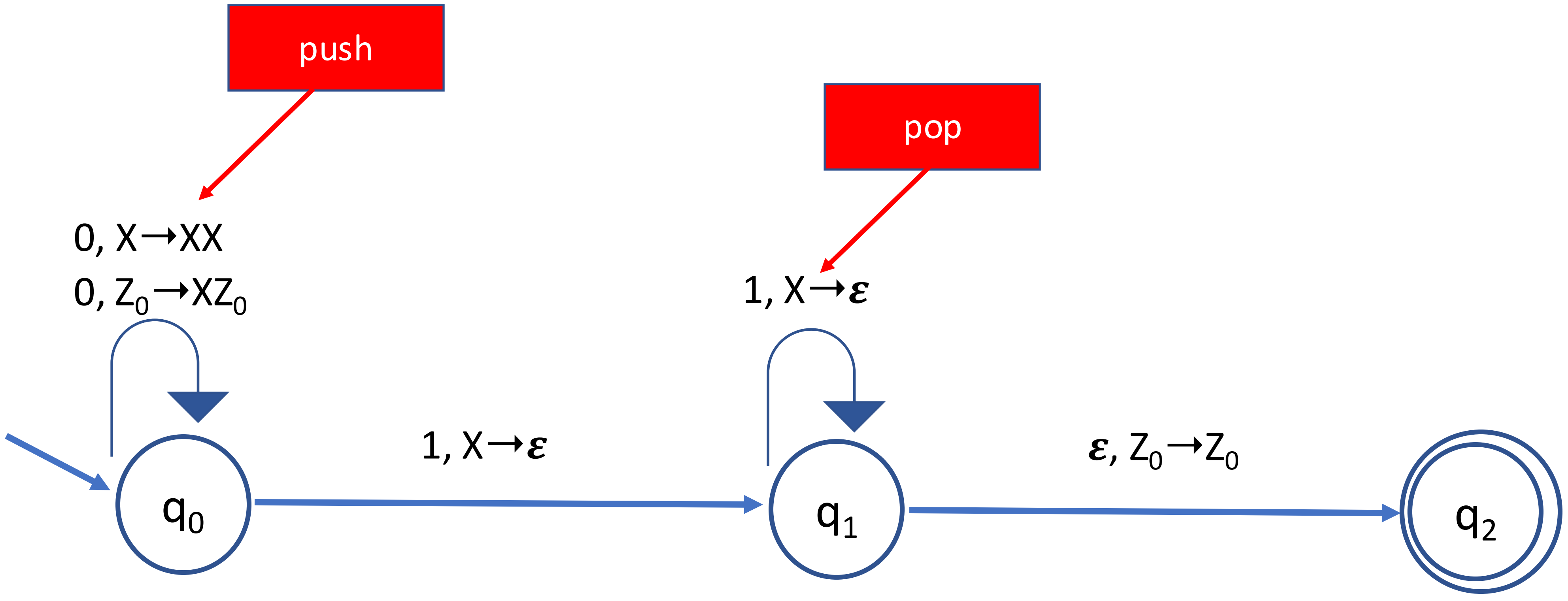
Exemple 1 (table)

		0	1	ε
q_0	Z_0	q_0,XZ_0		
	X	q_0,XX	q_1,ε	
q_1	Z_0			q_2,Z_0 (q1, ε)
	X		q_1,ε	
q_2	Z_0			
	X			

(q0,0011,Z0) |- (q0,011,XZ0) |- (q0,11,XXZ0) |- (q1,1,XZ0)
|- (q1, ε,Z0) |- (q2, ε,Z0) q2 final seq. is acc based on final state

(q0,0011,Z0) |- (q0,011,XZ0) |- (q0,11,XXZ0) |- (q1,1,XZ0)
|- (q1, ε,Z0) |-(q1, ε, ε) seq is acc based on empty stack

Exemple 1 (graphic)



Properties

Theorem 1: For any PDA M, there exists a PDA M' such that

$$L_{\varepsilon}(M) = L_f(M')$$

Theorem 2: For any PDA M, there exists a context free grammar such that

$$L_{\varepsilon}(M) = L(G)$$

Theorem 3: For any context free grammar there exists a PDA M such that

$$L(G) = L_{\varepsilon}(M)$$

HW

- Parser:
  - Descendent recursive
  - LL(1)
  - LR(0), SLR, LR(1)

Corresponding PDA