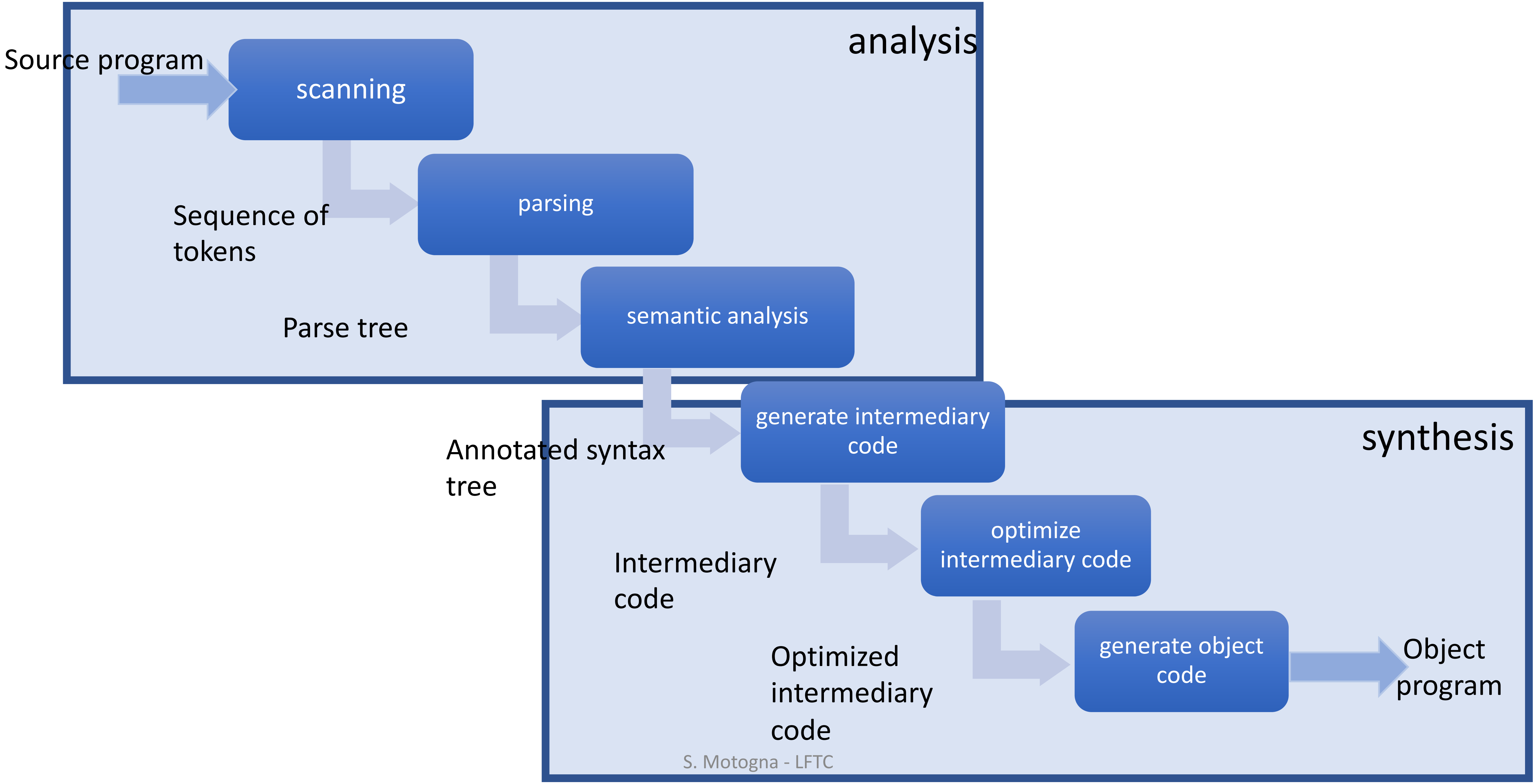
Course 11

Structure of compiler



Generate object code

= translate intermediary code statements into statements of object code (machine language)

- Depend on “machine”: architecture and OS

2 aspects:

- Register allocation – way in which variable are stored and manipulated;

- Instruction selection – way and order in which the intermediary code statements are mapped to machine instructions

2 computational models

- Computer with accumulator (stack machine)

- Computer with registers

Computer with accumulator (stack machine)

- Accumulator – to execute operation
- Stack to store subexpressions and results
- 2 types of statements:
  - move and copy values in and from head of stack to memory
  - Operations on stack head, functioning as follows: operands are popped from stack, execute operation in accumulator and then put the result in stack

Example: 4 * (5+1)

Code	acc	stack
acc ← 4	4	<>
push acc	4	<4>
acc ← 5	5	<4>
push acc	5	<5,4>
acc ← 1	1	<5,4>
acc ← acc + head	6	<5,4>
pop	6	<4>
acc ← acc * head	24	<4>
pop	24	<>

Computer with registers

- Registers
- Memory

- Instructions:
  - LOAD v,R – load value **v** in register **R**
  - STORE R,v – put value **v** from register **R** in memory
  - Operations - ex: ADD R1,R2 – add to the value from register **R1**, value from register **R2** and store the result in **R1** (initial value is lost!)

Remarks:

1. A register can be available or occupied =>

   VAR(R) = set of variables whose values are stored in register R

2. For every variable, the place (register, stack or memory) in which the current value of the value exists=>

   MEM(x)= set of locations in which the value of variable x exists (will be stored in Symbol Table)

Example: $F := A * B - (C + B) * (A * B)$

Intermediary code	Object code	VAR	MEM
		VAR(R0) = {} VAR(R1) = {}	
(1) T1 = A * B			
(2) T2 = C + B			
(3) T3 = T2 * T1			
(4) F:= T1 – T3			

Example: F := A ∗ B − (C + B) ∗ (A * B)

Intermediary code	Object code	VAR	MEM
		VAR(R0) = {} VAR(R1) = {}	
(1) T1 = A * B	LOAD A, R0 MUL R0, B	VAR(R0) = {T1}	MEM(T1) = {R0}
(2) T2 = C + B			
(3) T3 = T2 * T1			
(4) F:= T1 – T3			

Example: F := A ∗ B – (C + B) ∗ (A * B)

Intermediary code	Object code	VAR	MEM
		VAR(R0) = {} VAR(R1) = {}	
(1) T1 = A * B	LOAD A, R0 MUL R0, B	VAR(R0) = {T1}	MEM(T1) = {R0}
(2) T2 = C + B	LOAD C, R1 ADD R1, B	VAR(R1) = {T2}	MEM(T2) = {R1}
(3) T3 = T2 * T1			
(4) F:= T1 – T3			

Example: F := A ∗ B – (C + B) ∗ (A * B)

Intermediary code	Object code	VAR	MEM
		VAR(R0) = {} VAR(R1) = {}	
(1) T1 = A * B	LOAD A, R0 MUL R0, B	VAR(R0) = {T1}	MEM(T1) = {R0}
(2) T2 = C + B	LOAD C, R1 ADD R1, B	VAR(R1) = {T2}	MEM(T2) = {R1}
(3) T3 = T2 * T1	MUL R1,R0	VAR(R1) = {T3}	MEM(T2) = {} MEM(T3) = {R1}
(4) F:= T1 – T3			

Example: F := A ∗ B – (C + B) ∗ (A * B)

Intermediary code	Object code	VAR	MEM
		VAR(R0) = {} VAR(R1) = {}	
(1) T1 = A * B	LOAD A, R0 MUL R0, B	VAR(R0) = {T1}	MEM(T1) = {R0}
(2) T2 = C + B	LOAD C, R1 ADD R1, B	VAR(R1) = {T2}	MEM(T2) = {R1}
(3) T3 = T2 * T1	MUL R1,R0	VAR(R1) = {T3}	MEM(T2) = {} MEM(T3) = {R1}
(4) F:= T1 – T3	SUB R0,R1 STORE RO, F	VAR(R0) = {F} VAR(R1) = {}	MEM(T1) = {} MEM(F) = {R0, F}

Push-Down Automata (PDA)

Intuitive Model

Definition

- A push-down automaton (PDA) 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:

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

- Final state principle:

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

Representations

- Enumerate
- Table
- Graphic

Construct PDA

- $L = \{0^n1^n \mid n \geq 1\}$
- 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

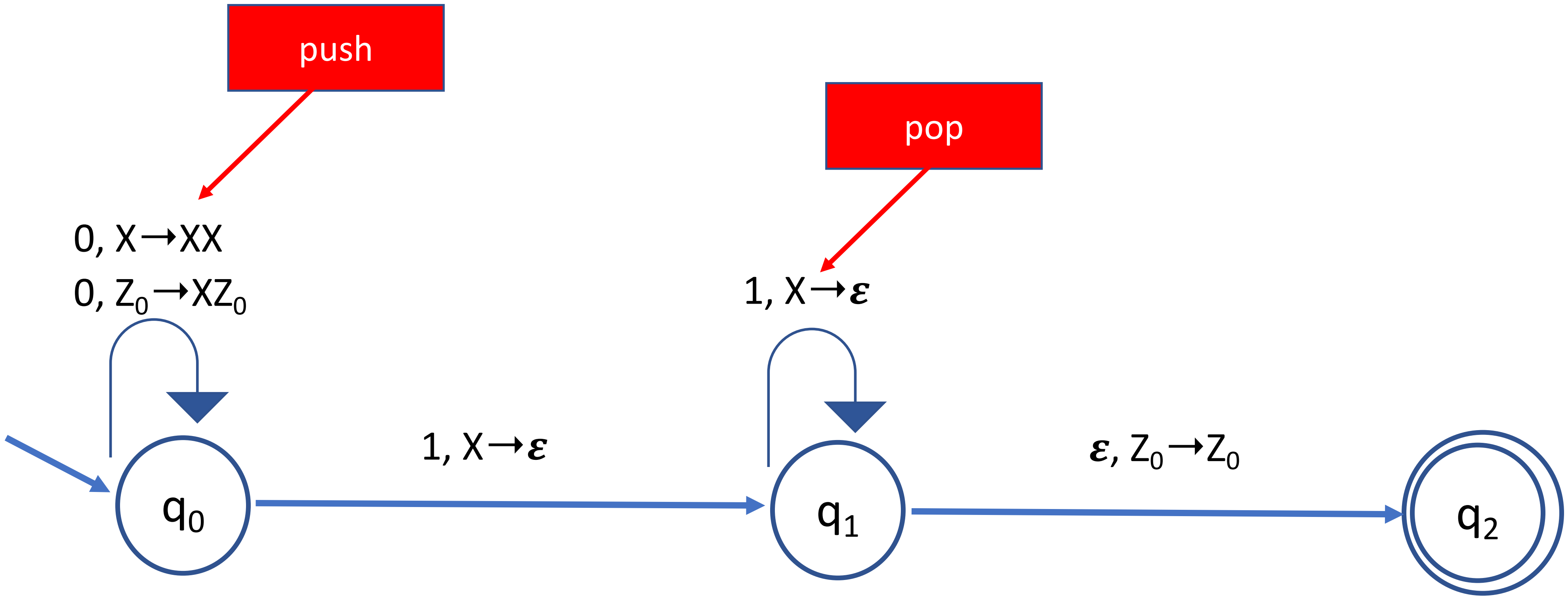
Exemple 1 (table)

		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 (q_1, 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

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