

Course 11

Push-Down Automata (PDA)

Intuitive Model

Definition

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

- Read symbol
- No action

Stack:

- Zero symbols \Rightarrow pop
- One symbol \Rightarrow push
- Several symbols \Rightarrow repeated push

Configurations and transition / moves

- Configuration:

$$(q, x, \alpha) \in Q \times \Sigma^* \times \Gamma^*$$

where:

- PDA is in state q
- Input band contains x
- Head of stack is α
- Initial configuration (q_0, w, Z_0)

Configurations and moves(cont.)

- Moves between configurations:

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

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

$(q, aw, Z\alpha) \vdash (p, w, \alpha)$ iff $\delta(q, a, Z) \ni (p, \epsilon)$

$(q, aw, Z\alpha) \vdash (p, aw, \gamma Z\alpha)$ iff $\delta(q, \epsilon, Z) \ni (p, \gamma Z)$
(ϵ -move)

- $\vdash^k, \vdash^\dagger, \vdash^*$

Language accepted by PDA

- Empty stack principle:

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

- Final state principle:

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

Representations

- Enumerate
- Table
- Graphic

Construct PDA

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

1. States:

- Initial state: q_0 – beginning and process symbols '0'
- When first symbol '1' is found – move to new state $\Rightarrow q_1$
- 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)

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

$$\delta(q_0, 0, Z_0) = (q_0, XZ_0)$$

$$\delta(q_0, 0, X) = (q_0, XX)$$

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

$$\delta(q_1, 1, X) = (q_1, \varepsilon)$$

~~$$\delta(q_1, \varepsilon, Z_0) = (q_2, Z_0)$$~~

$$\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$$

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

Empty stack

$$\vdash (q_1, \varepsilon, \varepsilon)$$

Final state

Example 1 (table)

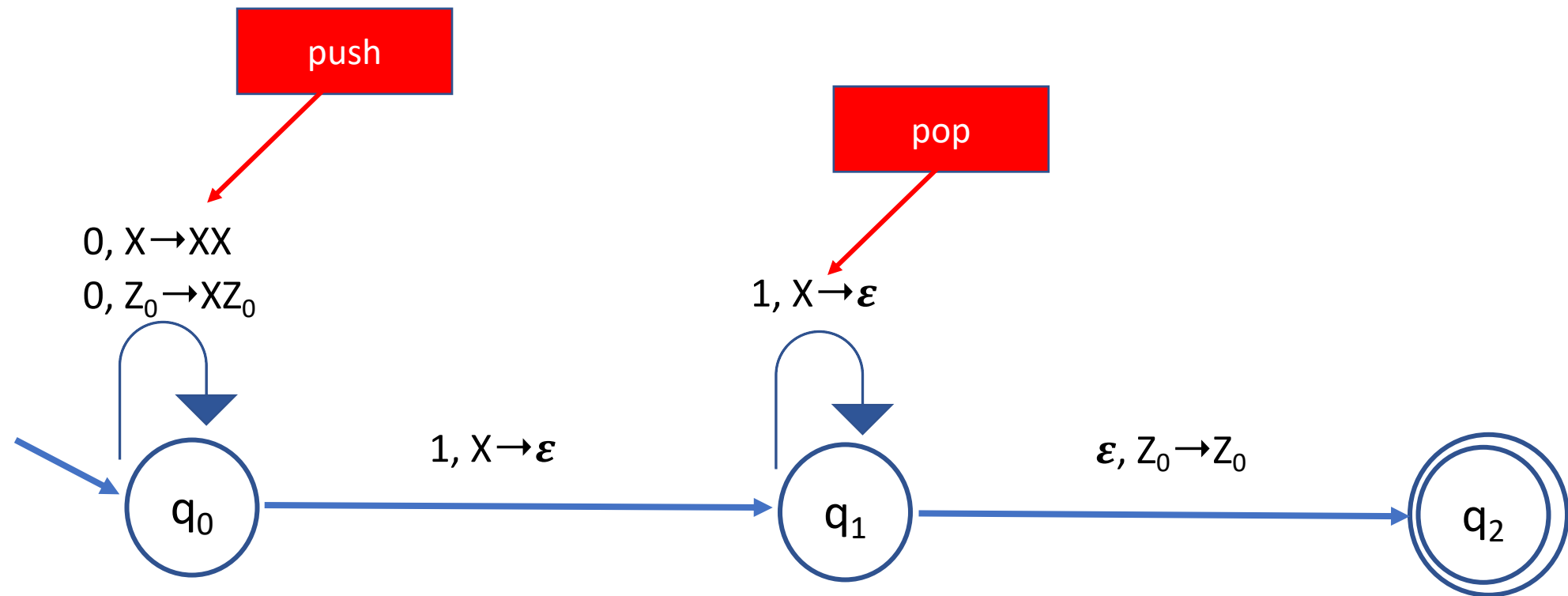
		0	1	ϵ
q_0	Z_0	q_0, XZ_0		
	X	q_0, XX	q_1, ϵ	
q_1	Z_0			q_2, Z_0
	X		q_1, ϵ	
q_2	Z_0			
	X			

(q_1, ϵ)

$(q_0, 0011, Z_0) \mid - (q_0, 011, XZ_0) \mid - (q_0, 11, XXZ_0) \mid - (q_1, 1, XZ_0)$
 $\mid - (q_1, \epsilon, Z_0) \mid - (q_2, \epsilon, Z_0)$ q_2 final seq. is acc based on final state

$(q_0, 0011, Z_0) \mid - (q_0, 011, XZ_0) \mid - (q_0, 11, XXZ_0) \mid - (q_1, 1, XZ_0)$
 $\mid - (q_1, \epsilon, Z_0) \mid - (q_1, \epsilon, \epsilon)$ seq is acc based on empty stack

Example 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

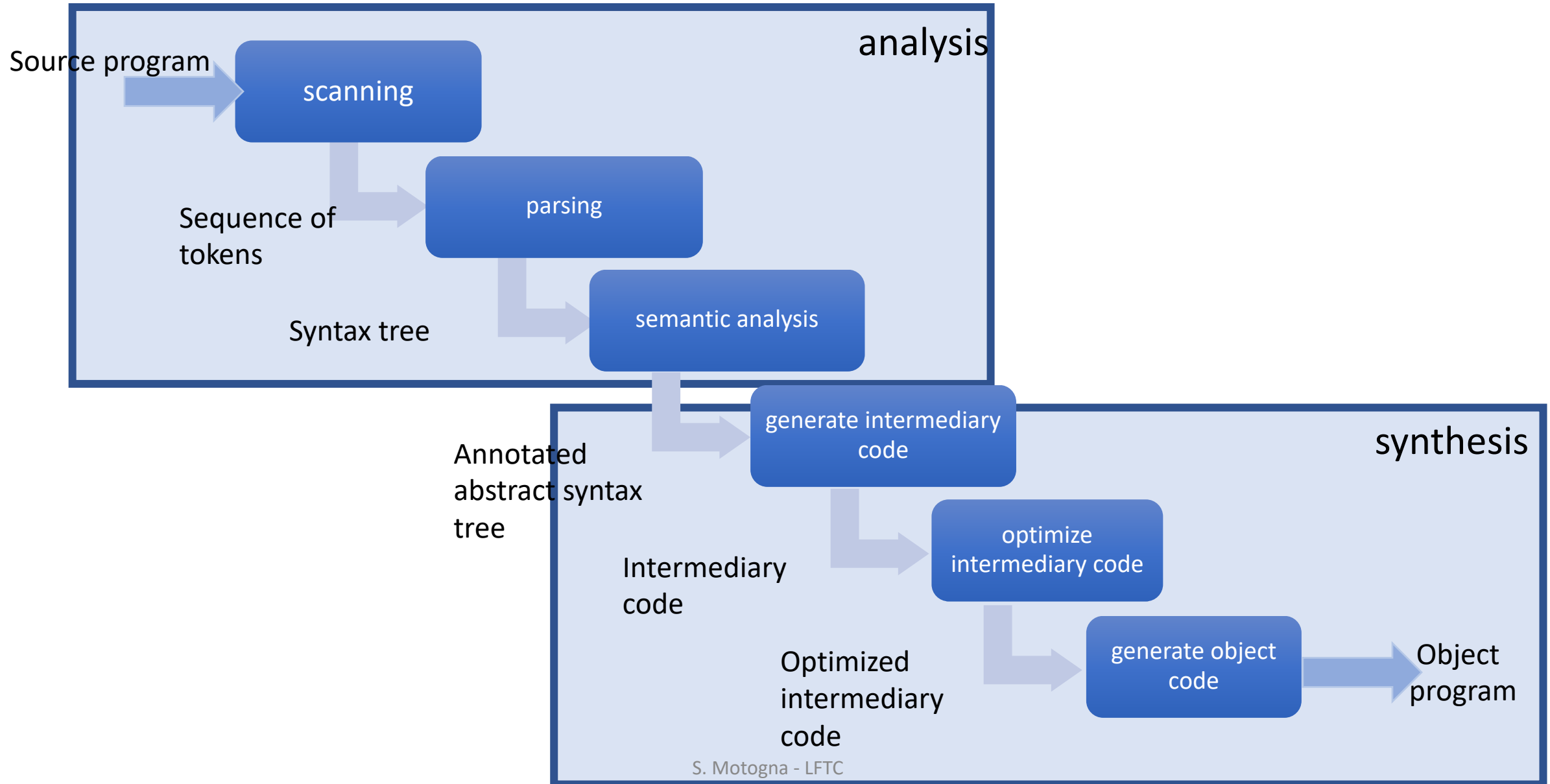
$$L(G) = L_{\varepsilon}(M)$$

HW

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

Corresponding PDA

Structure of compiler



Semantic analysis

- Parsing – result: syntax tree (ST)
- Simplification: abstract syntax tree (AST)
- Annotated abstract syntax tree (AAST)
 - Attach semantic info in tree nodes

Example

Semantic analysis

- Attach meanings to syntactical constructions of a program
- What:
 - Identifiers -> values / how to be evaluated
 - Statements -> how to be executed
 - Declaration -> determine space to be allocated and location to be stored
- Examples:
 - Type checkings
 - Verify properties
- How:
 - **Attribute grammars**
 - Manual methods

Attribute grammar

- Syntactical constructions (nonterminals) – attributes

$$\forall X \in N \cup \Sigma: A(X)$$

- Productions – rules to compute/ evaluate attributes

$$\forall p \in P: R(p)$$

Definition

AG = (G,A,R) is called ***attribute grammar*** where:

- $G = (N, \Sigma, P, S)$ is a context free grammar
- $A = \{A(X) \mid X \in N \cup \Sigma\}$ – is a finite set of attributes
- $R = \{R(p) \mid p \in P\}$ – is a finite set of rules to compute/evaluate attributes

Example 1

- $G = (\{N, B\}, \{0, 1\}, P, N)$

P:

$$\begin{array}{l} N \xrightarrow{1} N \xrightarrow{2} B \\ \underline{N \rightarrow B} \\ B \rightarrow 0 \\ B \rightarrow 1 \end{array}$$

$$\begin{array}{l} N_1.v = 2 * N_2.v + B.v \\ \underline{N.v = B.v} \\ B.v = 0 \\ \underline{B.v = 1} \end{array}$$

Attribute – value of number = v

- **Synthesized attribute: $A(lhp)$ depends on rhp**
- **Inherited attribute: $A(rhp)$ depends on lhp**

Evaluate attributes

- Traverse the tree: can be an infinite cycle
- Special classes of AG:
 - L-attribute grammars: for any node the depending attributes are on the “*left*”;
 - can be evaluated in one left-to-right traversal of syntax tree
 - Incorporated in top-down parser (LL(1))
 - S-attribute grammars: synthesized attributes
 - Incorporated in bottom-up parser (LR)

Steps

- What? - decide what you want to compute (type, value, etc.)
- Decide attributes:
 - How many
 - Which attribute is defined for which symbol
- Attach evaluation rules:
 - For each production – which rule/rules

Example 2 (L-attribute grammar)

Decl \rightarrow DeclTip ListId

ListId \rightarrow Id

ListId \rightarrow ListId, Id

ListId.type = DeclTip.type

Id.type = ListId.type

ListId₂.type = ListId₁.type

Id.type = ListId₁.type

Attribute – type

int i,j

Example 3 (S-attribute grammar)

ListDecl \rightarrow ListDecl; Decl

ListDecl \rightarrow Decl

Decl \rightarrow Type ListId

Type \rightarrow int

Type \rightarrow long

ListId \rightarrow Id

ListId \rightarrow ListId, Id

$\text{ListDecl}_1.\text{dim} = \text{ListDecl}_2.\text{dim} + \text{Decl}.\text{dim}$

$\text{ListDecl}.\text{dim} = \text{Decl}.\text{dim}$

$\text{Decl}.\text{dim} = \text{Type}.\text{dim} * \text{ListId}.\text{no}$

$\text{Type}.\text{dim} = 4$

$\text{Type}.\text{dim} = 8$

$\text{ListId}.\text{no} = 1$

$\text{ListId}_1.\text{no} = \text{ListId}_2.\text{no} + 1$

Attributes – dim + no – **for which symbols**

int i,j; long k

Proposed problems (HW):

- 1) Define an attribute grammar for arithmetic expressions
- 2) Define an attribute grammar for logical expressions
- 3) Define an attribute grammar for if statement

Manual methods

- Symbolic execution
 - Using control flow graph, simulate on stack how the program will behave
 - [Grune – Modern Compiler Design]
- Data flow equations
 - Data flow – associate equations based on data consumed in each node (statement) of the control flow graph: In, Out, Generated, Killed
 - [Grune – Modern Compiler Design], [[Kildall](#)], [[course](#)]