TDDD14 / TDDD85 – Lecture 9

Pushdown Automata

August Ernstsson, 2024 (based on lecture notes by Jonas Wallgren)

LINKÖPING UNIVERSITY

From last lecture

Simplification – ε productions and unit productions

- Start: P' = P = { S → aSb, S → T, T → pTq, T → ε } (4 productions)
- Extension steps:
  - **If** A → αBγ **and** B → ε are in P' **then** put A → αγ in P'
  - **If** A → B **and** B → β are in P' **then** put A → β in P'
- Application: (colored by which extension step is used)
  - **Since** S → T **and** T → ε **in** P', **put** S → ε **in** P'
  - **Since** T → pTq **and** T → ε **in** P', **put** T → pq **in** P'
  - **Since** S → aSb **and** S → ε **in** P', **put** S → ab **in** P'
  - **Since** S → T **and** T → pTq **in** P', **put** S → pTq **in** P'
  - **Since** S → pTq **and** T → ε **in** P', **put** S → pq **in** P'

(Note: First row matches both types of extension. This was what caused some confusion during the lecture.)

Let's start!

LINKÖPING UNIVERSITY

Pushdown Automata: Introduction

- Just as finite automata can express regular languages, we seek an automaton that can express context-free languages
- We start with the model for an NFA and add a memory component
  - A pushdown memory, ”stack”.
  - Only the top element can be read
  - We can push a new top element
  - Or pop the old top element and discard it

Example 1: A Pushdown Automata

Top of stack	State	Symbol read		
		0	1	C
B	q_1	PushB q_1	PushG q_1	q_2
	q_2	Pop q_2		
G	q_1	PushB q_1	PushG q_1	q_2
	q_2		Pop q_2	
R	q_1	PushB q_1	PushG q_1	q_2
	q_2	Without reading: Pop q_2		

- A push-down automaton for $L = \{ wcw^R | w \in \{0, 1\}^* \}$
- Where x^R is w reversed, i.e. *palindromes* over {0,1} with a c in the center
- Read 0, 1, or c.
  - Put B, G, or R on the stack.
  - Start in state q_1.
  - Start with R on the stack.

Example 2: PDA string acceptance

Top of stack	State	Symbol read		
		0	1	C
B	q_1	PushB q_1	PushG q_1	q_2
	q_2	Pop q_2		
G	q_1	PushB q_1	PushG q_1	q_2
	q_2		Pop q_2	
R	q_1	PushB q_1	PushG q_1	q_2
	q_2	Without reading: Pop q_2		

- Read the string 01c10

Example 2: PDA string acceptance

Top of stack	State	Symbol read		
		0	1	C
B	q_1	PushB q_1	PushG q_1	q_2
	q_2	Pop q_2		
G	q_1	PushB q_1	PushG q_1	q_2
	q_2		Pop q_2	
R	q_1	PushB q_1	PushG q_1	q_2
	q_2	Without reading: Pop q_2		

State	Remaining string	Stack	Comment
q_1	01c10	R	Start situation
q_1	1c10	BR	According to column 0, row Rq_1 in the table
q_1	c10	GBR	According to column 1, row Rq_1 in the table
q_2	10	GBR	…
q_2	0	BR	…
q_2		R	…
q_2			String is read, stack is empty: **accept**!

Example 2: PDA string acceptance

Top of stack	State	Symbol read		
		0	1	c
B	q_1	PushB q_1	PushG q_1	q_2
	q_2	Pop q_2		
G	q_1	PushB q_1	PushG q_1	q_2
	q_2		Pop q_2	
R	q_1	PushB q_1	PushG q_1	q_2
	q_2	Without reading: Pop q_2		

- Meaning of the states:
  - All symbols before c are read in q_1; all symbols after c are read in q_2.
  - If you read a 0 before the c you push a B on the stack.
  - You can read a 0 after the c only if there is a B on the stack top.
  - Correspondingly for 1 and G

Definition 1: Pushdown Automata

- A PDA is a septuple $\langle Q, \Sigma, \Gamma, \delta, s, \bot, F \rangle$
  - Q = set of states
  - Σ = (input) alphabet
  - Γ = stack alphabet
  - s = start state $\in Q$
  - F = final states $\subseteq Q$
  - $\bot$ = start stack symbol
  - δ = transition relation $\subseteq (Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma) \times (Q \times \Gamma*))$

The transition relation δ

- δ = transition relation $\subseteq (Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma) \times (Q \times \Gamma*))$

- $\langle\langle p, a, A\rangle, \langle q, B_1, B_2 \ldots B_n\rangle\rangle \in \delta$ means:
  - In state p with A on top of the stack:
    - Read a,
    - go to state q,
    - change the stack top A to $B_1B_2 \ldots B_n$ (the left end being the new stack top).

The transition relation δ

- δ = transition relation $\subseteq (Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma) \times (Q \times \Gamma*))$

- $\langle\langle p, \varepsilon, A\rangle, \langle q, B_1, B_2 \ldots B_n\rangle\rangle \in \delta$ means:
  - In state p with A on top of the stack, without reading anything:
    - Go to state q,
    - change the stack top A to $B_1B_2 \ldots B_n$ (the left end being the new stack top).

Example 3

- The PDA in Example 1 can formally be specified as
  - $\langle \{q1, q2\}, \{0, 1, c\}, \{B, G, R\}, \delta, q1, R, \varnothing \rangle$,
    - where δ is defined according to the table.

Top of stack	State	Symbol read		
		0	1	C
B	q_1	PushB q_1	PushG q_1	q_2
	q_2	Pop q_2		
G	q_1	PushB q_1	PushG q_1	q_2
	q_2		Pop q_2	
R	q_1	PushB q_1	PushG q_1	q_2
	q_2	Without reading: Pop q_2		

Configurations

- What we handled in Example 2 are called *configurations*.

- **Definition 2**. A configuration is a triple ⟨q, x, γ⟩, where:
  - q = state ∈ Q
  - x = string ∈ Σ*
  - γ = stack ∈ Γ*

From Example 2:
⟨q1, 01c01, R⟩ → ⟨q1, 1c01, BR⟩

- **Definition 3**. The next-configuration relation →
  - If ⟨⟨p, a, A⟩, ⟨q, γ⟩⟩ ∈ δ then ⟨p, ay, Aβ⟩ → ⟨q, y, γβ⟩⟩
  - If ⟨⟨p, ε, A⟩, ⟨q, γ⟩⟩ ∈ δ then ⟨p, y, Aβ⟩ → ⟨q, y, γβ⟩⟩

Acceptance

- Two different modes of acceptance are used in the literature.
- A PDA can accept a string if
  - the *stack is empty*, or
  - if it reaches a *final state*.

- **Definition 4**
  - A PDA accepts the string x if $\langle s, x, \bot \rangle \rightarrow^* \langle q, \varepsilon, \gamma \rangle$ when $q \in F$.
- **Definition 5**
  - A PDA accepts the string x if $\langle s, x, \bot \rangle \rightarrow^* \langle q, \varepsilon, \varepsilon \rangle$ when $q \in Q$

Acceptance, cont.

- Two different modes of acceptance are used in the literature.
- A PDA can accept a string if
  - the *stack is empty*, or
  - if it reaches a *final state*.

- **Definition 6.** The language of a PDA.
  - M = some PDA
  - L(M) = the set of all strings that are accepted by M.

Equivalence between ways of acceptance

- We will prove that if there is a PDA accepting the string x with empty stack then there is a PDA accepting it in final state and v.v., i.e.
  - the two ways of accepting a string are equally powerful,
  - they define the same class of languages.
- We will perform the two proofs (the two directions of the implications making up the equivalence) somewhat in parallel.
- Our starting point is the PDA $M = \langle Q, \Sigma, \Gamma, \delta, s, \bot, F \rangle$.
  - It accepts strings in either way.

Equivalence between ways of acceptance

- Two new sets are defined.

- If M accepts with **empty stack**:
  - G = Q
  - $\Delta = \{\bot\bot\}$

- If M accepts in **final state**:
  - G = F
  - $\Delta = \Gamma \cup \{\bot\bot\}$

Equivalence between ways of acceptance

- We now perform an operation a litle bit like the one done when constructing the regular expression from a DFA — we add some extra handling in the start and in the end.
- We define a new PDA $M' = \langle Q \cup \{u, t\}, \Sigma, \Gamma \cup \{\perp\!\!\!\perp\}, \delta', u, \perp\!\!\!\perp, \{t\}\rangle$, where
  - u = new start state
  - t = new final state
  - $\perp\!\!\!\perp$ = new stack bottom symbol
  - δ'=new transition relation, defined as:
    - $\delta' = \delta \cup \{ \langle\langle u, \varepsilon, \perp\!\!\!\perp\rangle, \langle s, \perp\, \perp\!\!\!\perp\rangle\rangle,$
      - $\langle\langle q, \varepsilon, A\rangle, \langle t, A\rangle\rangle$ for $q \in G, A \in \Delta,$
      - $\langle\langle t, \varepsilon, A\rangle, \langle t, \varepsilon\rangle\rangle$ for $A \in \Gamma \cup \{\perp\perp\}$ }

Equivalence between ways of acceptance

- $\delta' = \delta \cup \{ \langle\langle u, \varepsilon, \perp\!\!\!\perp\rangle, \langle s, \perp\, \perp\!\!\!\perp\rangle\rangle,$
  - $\langle\langle q, \varepsilon, A\rangle, \langle t, A\rangle\rangle$ for $q \in G, A \in \Delta,$
  - $\langle\langle t, \varepsilon, A\rangle, \langle t, \varepsilon\rangle\rangle$ for $A \in \Gamma \cup \{\perp\perp\}$ }
- The first new element of δ' says that in M', you first just go from its start state to the start state of M and you put the stack bottom of M on the stack.
  - To prepare for the simulation of M.
- The next new part of δ' says that when you are in an accepting situation in M you go to the accepting state of M'.
- The third part says that if there is anything left on the stack you could remove it.
- So, M' accepts with empty stack and in final state.

Equivalence between ways of acceptance

- **Lemma 1**. If M accepts x with empty stack then M' accepts it.
- Proof. M accepts with empty stack: $\langle s, x, \bot\rangle$ $n\rightarrow M$ $\langle q, \varepsilon, \varepsilon\rangle$.
  - Then $\langle u, x, \perp\!\!\!\perp\rangle$ $1\rightarrow M'$ $\langle s, x, \bot\perp\!\!\!\perp\rangle$ $n\rightarrow M'$ $\langle q, \varepsilon, \perp\!\!\!\perp\rangle$ $1\rightarrow M'$ $\langle t, \varepsilon, \perp\!\!\!\perp\rangle$ $1\rightarrow M'$ $\langle t, \varepsilon, \varepsilon\rangle$.
    1. The first step as the new elements of δ' states,
    2. the second step since δ' contains δ,
    3. the last two steps as the new elements of δ' states.
  - So, $\langle u, x, \perp\!\!\!\perp\rangle$ $*\rightarrow M'$ $\langle t, \varepsilon, \varepsilon\rangle$, i.e. M' accepts x if M accepts it with empty stack.

Equivalence between ways of acceptance

- **Lemma 2**. If M accepts x in final state then M' accepts it.
- Proof. M accepts in final state: $\langle s, x, \bot \rangle$ $n \rightarrow_M \langle q, \varepsilon, \gamma \rangle$, where $q \in F$.
  - Then $\langle u, x, \perp\!\!\!\perp \rangle$ $1 \rightarrow_{M'} \langle s, x, \bot \perp\!\!\!\perp \rangle$ $n \rightarrow_{M'} \langle q, \varepsilon, \gamma \perp\!\!\!\perp \rangle$ $1 \rightarrow_{M'} \langle t, \varepsilon, \gamma \perp\!\!\!\perp \rangle$ $* \rightarrow_{M'} \langle t, \varepsilon, \varepsilon \rangle$.
    1. The first step as the new elements of δ' states,
    2. the second step since δ' contains δ,
    3. the last two steps as the new elements of δ' states.

Equivalence between ways of acceptance

- **Lemma 3**. If M accepts x then M' accepts it.
  - Proof. Follows from Lemma 1 and Lemma 2

Equivalence between ways of acceptance

- **Lemma 4.** If M’ accepts x then M accepts it

- Proof. Consider this sequence of steps:
- $\langle u, x, \perp\!\!\!\perp \rangle \; 1 \rightarrow_{M'} \langle s, x, \perp \perp\!\!\!\perp \rangle \; n \rightarrow_{M'} \langle q, y, \gamma \perp\!\!\!\perp \rangle \; 1 \rightarrow_{M'} \langle t, y, \gamma \perp\!\!\!\perp \rangle \; * \rightarrow_{M'} \langle t, \varepsilon, \varepsilon \rangle$
  1. The first step is the same initial one again.
  2. For M’ to accept x it must in the process have reached a state $q \in G$ since that is the only way δ' can get the automaton into the state t. That is shown in step 2, with a y that maybe not is ε.
  3. The third step is just a move from q to t.
  4. Since we know that M’ accepts x there must be some way to perform the last step. But in state t M cannot read anything, so y must be ε. So if the second step is read from the M perspective we get $\langle s, x, \perp \rangle \; n \rightarrow_{M} \langle q, \varepsilon, \gamma \rangle$, i.e. M accepts x.

Equivalence between ways of acceptance

- **Theorem 1**.
  - Accepting with empty stack and accepting in final state are equivalent.
  - *Proof*. It follows from Lemma 3 and Lemma 4

To think about

- Can you think of a context-free language where we (at least intuitively) need the power of non-determinism in a PDA to recognize the languge?

- How could you simulate a PDA on an input string in your favourite programming language?
  - How would such a simulation differ from simulating an NFA? How much memory does your implementation need?

Coming up

- This week
  - **Monday**: Pushdown automata (PDA)
  - **Friday**: Equivalence between CFG and PDA

- Then three weeks with one lecture per week
  - Properties of CFGs and parsing methods for CFGs

Thanks for today!

LINKÖPING UNIVERSITY