Theoretical Computer Science

Lab Session 6

March 11, 2021

Agenda

- Deterministic Pushdown Automaton (DPDA): Notion, formal definition, configuration, transition, and acceptance.
- Exercises on DPDAs.

Pushdown Automata (Introduction)

A Pushdown Automaton (PDA) is a way to implement a Context Free Grammar in a similar way we design Finite Automaton for Regular Grammar

$\rightarrow$ It is more powerful than FSA

$\rightarrow$ FSA has a very limited memory but PDA has more memory

$\rightarrow$ $PDA =$ Finite State Automaton $+$ a Stack

Pushdown Automata (Introduction)

A Pushdown Automaton (PDA) is a way to implement a Context Free Grammar in a similar way we design Finite Automaton for Regular Grammar

$\rightarrow$ It is more powerful than FSA

$\rightarrow$ FSA has a very limited memory but PDA has more memory

$\rightarrow$ $PDA =$ Finite State Automaton $+$ a Stack

A stack is a way we arrange elements one on top of another

A stack does two basic operations:

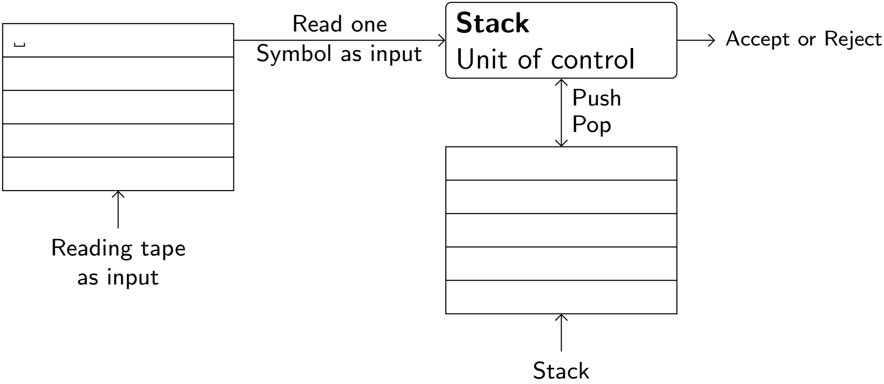
PUSH: A new element is added at the Top of the stack

POP: The Top element of the stack is read and removed

PDA-components

A Pushdown Automaton has 3 components:

1. An input tape
2. A Finite Control Unit
3. A Stack of infinite size



Acceptance criteria: Either reach final sate or stack is empty

PDA – Notion

- PDAs are similar to FSA with an auxiliary memory: a stack.
- Ex:

PDA – Notion

- PDAs are similar to FSA with an auxiliary memory: a stack.
- Ex: Build a complete FSA that recognises the following language:

$$AnBn = \{a^n b^n \mid n \geq 0\}$$

PDA – Notion

- PDAs are similar to FSA with an auxiliary memory: a stack.
- Ex: Build a complete FSA that recognises the following language:

$$AnBn = \{a^n b^n \mid n \geq 0\}$$

- It is not possible (you already know how to prove it!)

PDA – Notion

$$AnBn = \{a^n b^n \mid n \geq 0\}$$

- When a PDA reads an input symbol, it will be able to save it (or save other symbols) in its memory.
- For deciding if an input string is in the language *AnBn*, the PDA needs to remember the numbers of *a*'s.
- Whenever the PDA reads the input symbol *b*, two things should happen:
  1. it should change states: from now on the only legal input symbols are *b*'s.
  2. it should delete one *a* from its memory for every *b* it reads.

PDA – Notion (Moves)

A single move of a PDA will depend on:

- the current state,
- the next input (it could be no symbol: ϵ symbol), and
- the symbol currently on top of the stack.

PDA will be assumed to begin operation with an initial start symbol Z_0 on its stack

- Z_0 is not essential, but useful to simplify definitions
- Z_0 is on top means that the stack is effectively empty.

PDA – Formal Definition

A Pushdown Automaton

A PDA is a tuple $\langle Q, I, \Gamma, \delta, q_0, Z_0, F\rangle$ where

- Q is a finite set of states.
- I and Γ are finite sets, the input and stack alphabets.
- δ, the transition function, is a partial function from $Q \times (I \cup \{\epsilon\}) \times \Gamma$ to the set of finite subsets of $Q \times \Gamma^*$.
- $q_0 \in Q$, the initial state.
- $Z_0 \in \Gamma$, the initial stack symbol.
- $F \subseteq Q$, the set of accepting states.

PDA – Formal Definition II

δ takes as argument a triple $\delta(q, a, X)$ where

- (i) q is a state in Q
- (ii) a is either an Input Symbol in I or $a = \epsilon$
- (iii) X is a Stack Symbol, that is a member of Γ

The output of δ is finite set of pairs (p, y) where:
p is a new state
y is a string of stack symbols that replaces X at the top of the stack

- If $y = X$ then the stack is unchanged as we pop and then push the same symbol
- Otherwise X is replaced by the string y

A Deterministic PDA – Formal Definition (the one seen in the lecture)

A Deterministic Pushdown Automaton (DPDA)

A PDA $M = \langle Q, I, \Gamma, \delta, q_0, Z_0, F \rangle$ is deterministic if it satisfies both of the following conditions.

1. For every $q \in Q$, every $x \in I \cup \{\epsilon\}$, and every $\gamma \in \Gamma$, the set $\delta(q, x, \gamma)$ has at most one element.
2. For every $q \in Q$, every $x \in I$, and every $\gamma \in \Gamma$, the two sets $\delta(q, x, \gamma)$ and $\delta(q, \epsilon, \gamma)$ cannot both be non-empty.

Configuration

A configuration is a generalization of the notion of state. It shows:

- the current state,
- the portion of the input string that has not yet been read, and
- the stack.

It is a snapshot of the PDA.

Configuration – Formal Definition

Configuration

A Configuration of the PDA $\langle Q, I, \Gamma, \delta, q_0, Z_0, F \rangle$ is a triple

$$(q, x, \gamma)$$

where

- $q \in Q$, is the current state of the control device,
- $x \in I^*$, is the unread portion of the input string, and
- $\gamma \in \Gamma^*$, is the string of symbols in the stack.

Transition

Transitions between configurations ($\vdash$) depend on the transition function. It is the way to commute from a PDA snapshot to another.
There are 2 cases:

1. The transition function is defined for an input symbol.
2. The transition function is defined for an ϵ move.

Transition

If $(q', \alpha) \in \delta(q, i, A)$ then $(q, x, \gamma) \vdash (q', x', \gamma')$
If $(q', \alpha) \in \delta(q, \epsilon, A)$ then $(q, x, \gamma) \vdash (q', x'', \gamma')$
where (old snapshot)

- q is the current state
- $x = iy$
- $\gamma = A\beta$ (for some $\beta \in \Gamma^*$)

then (new snapshot)

- q' is the new state
- $x' = y$
- $x'' = x$
- $\gamma' = \alpha\beta$

Acceptance – Informally

A string x is accepted by a PDA if there is a path coherent with x on the PDA that goes from the initial state to the final state.
The input string has to be read completely

Acceptance – Formal Definition

Reflexive transitive closure of $\vdash$

Let M be the PDA $\langle Q, I, \Gamma, \delta, q_0, Z_0, F \rangle$, and $c_i = (q, x, \beta)$, $c_j = (q', x', \beta')$ be configurations of M:

$$c_i \vdash^* c_j$$

is the sequence of zero or more moves taking M from c_i to c_j

Acceptance by a PDA

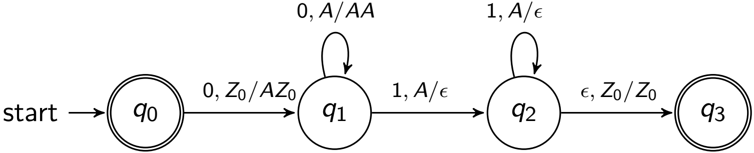
Let M be the PDA $\langle Q, I, \Gamma, \delta, q_0, Z_0, F \rangle$, and $x \in I^*$. The string x is accepted by M if

$$(q_0, x, Z_0) \vdash^* (q, \epsilon, \gamma)$$

for some $\gamma \in \Gamma^*$ and some $q \in F$.

Simple solution

problem : Construct a PDA that accepts $L = \{0^n 1^n \mid n \geqslant 0\}$



Exercises!

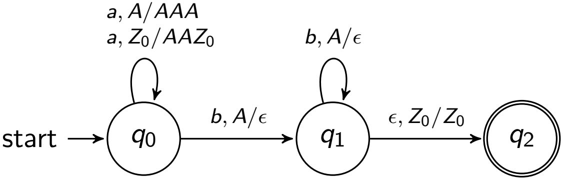
Exercises – Part 1

Build DPDAs that recognise the following languages:

1. $L = \{a^n b^{2n} \mid n \geq 1\}$
2. $SimplePal = \{xcx^R \mid x \in \{a, b\}^* \wedge |x| > 0\}$ (where x^R is the reversed string x), the alphabet is $I = \{a, b, c\}$
3. The language of nested and balanced brackets and parentheses. E.g. a string in the language: (([])())(), a string that does not belong to the language: ([(]))()() – the alphabet is $I = \{(,), [,]\}$

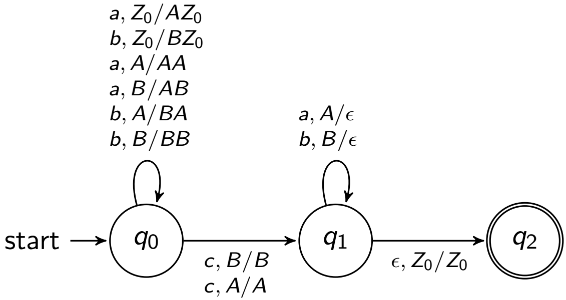
Exercises – Part 1 (1)

DPDA accepting $L = \{a^n b^{2n} \mid n \geq 1\}$



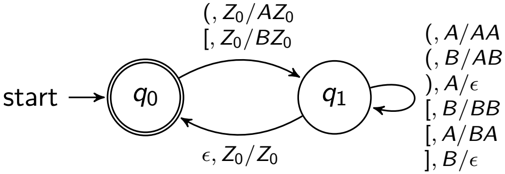
Exercises – Part 1 (2)

PDA accepting $SimplePal = \{xcx^R \mid x \in \{a, b\}^* \wedge |x| > 0\}$



Exercises – Part 1 (3)

PDA accepting the language of nested and balanced brackets and parentheses.



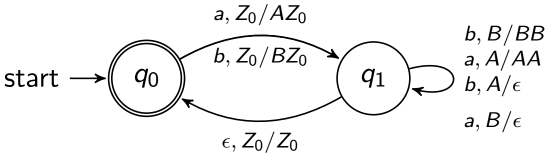
Exercises – Part 2

Build DPDAs that recognise the following languages:

1. $L_1 = \{w \in \{a, b\}^* \mid \phi(w, a) = \phi(w, b)\}$ where $\phi(s, c)$ is the number of occurrences of the character c in the string s.
2. $L_2 = \{a^n b^m a^m b^n \mid n > 0 \wedge m > 0\}$
3. $L_3 = L_2^*$
4. $L_4 = \{a^{n_1} b^{n_1} a^{n_2} b^{n_2} a^{n_3} b^{n_3} \dots a^{n_k} b^{n_k} \mid k \geq 1 \wedge n_i \geq 1 \wedge 1 \leq i \leq k\}$

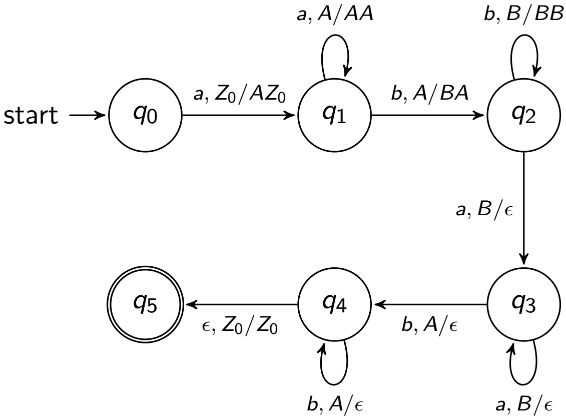
Exercises – Part 2 (1)

DPDA accepting $L_1 = \{w \in \{a, b\}^* \mid \phi(w, a) = \phi(w, b)\}$



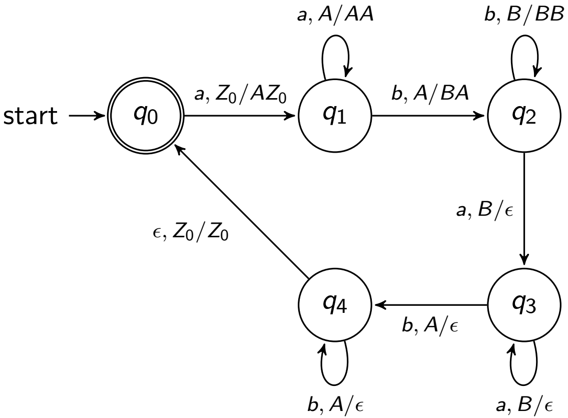
Exercises – Part 2 (2)

DPDA accepting $L_2 = \{a^n b^m a^m b^n \mid n > 0 \land m > 0\}$



Exercises – Part 2 (3)

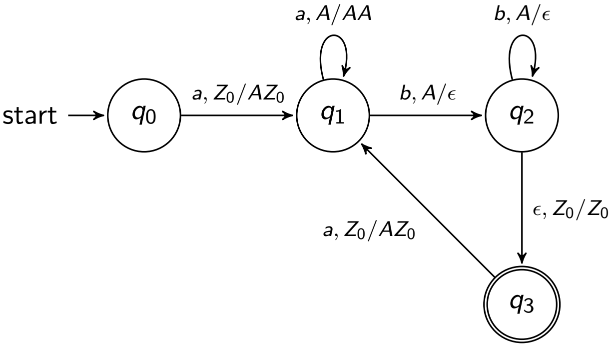
DPDA accepting $L_3 = L_2^*$



Exercises – Part 2 (4)

DPDA accepting

$$L_4 = \bigcup_{k \geq 1} \{a^{n_1} b^{n_1} a^{n_2} b^{n_2} a^{n_3} b^{n_3} \ldots a^{n_k} b^{n_k} \mid n_1, \ldots, n_k \geq 1\}$$



Exercises – Homework (1)

Consider the language of well-formed parentheses of the arithmetic expressions (binary operations). Examples of strings belonging to the language are:

- (a + b)
- ((a) + (b * c))
- ((a + b))

Define a DPDA that recognises this language. For simplicity, consider the following alphabet $I = \{a, (,), +\}$ – a single symbol ('a') that represents terms 'a', 'b', 'c', ... And a single operator ('+').