

LECTURE 8

PUSHDOWN AUTOMATA

8.1 Pushdown Automata	1
8.2 Deterministic Pushdown Automata	2
8.3 Nondeterministic Pushdown Automata	4
8.4 Context-free languages	5

8.1 Pushdown Automata

- A finite state automaton (FSA) has limited memory. This prevents it from recognising languages that are not regular.
- A pushdown automaton (PDA) consist of a finite state automaton along with a *stack*, which provides unlimited memory.
- If we allow the FSA to access two stacks, we obtain a more powerful machine, equivalent in computation power to a Turing machine.

Example 8.1. Consider the language of *well-formed brackets*, defined to be the set of strings for which

- There is an equal number of brackets in the expression.
- Starting at the first symbol and finishing anywhere in the string, there are no more closing brackets than opening brackets.

To determine whether an expression is well formed, all opening brackets are placed on the stack, and are removed as the corresponding closing brackets are encountered.

- For simplicity, we replace the symbols ‘(’ and ‘)’ by the symbols *a* and *b* respectively. The language of well-formed brackets then becomes the language $L(G)$ generated by the context-free grammar $G = (V, T, S, P)$ having $V = \{S, a, b\}$, $T = \{a, b\}$ and $P = \{S \rightarrow SS, S \rightarrow aSb, S \rightarrow \lambda\}$.

As the string *aaabbbbaababb* is processed, the stack changes as follows:

step	input	stack	step	input	stack
1	a	aZ	7	a	aZ
2	a	aaZ	8	a	aaZ
3	a	aaaZ	9	b	aZ
4	b	aaZ	10	a	aaZ
5	b	aZ	11	b	aZ
6	b	Z	12	b	Z

Table 8.1: Stack trace for the string *aaabbbbaababb*

- The symbol *Z* designates the empty stack.
- When a valid string has been processed, *Z* is on top of the stack.

At each step, the FSA part of the PDA takes a pair of symbols:

- The input symbol
- The top-of-stack symbol

For the language $L(G)$, the set of input pairs is:

$$\begin{matrix} (\lambda, Z), & (\lambda, a), & (\lambda, b), \\ (a, Z), & (a, a), & (a, b) \\ (b, Z), & (b, a), & (b, b) \end{matrix}$$

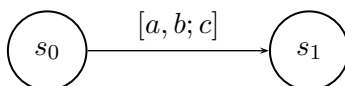
- The input symbol is on the left; the top-of-stack symbol is on the right.

Given an input pair, the PDA can *change state* and/or perform a *stack operation*

input pair	stack operation	new top-of-stack	new state
(λ, Z)	none	Z	PASS
(λ, a)	none	a	intermediate
(λ, b)	n/a	n/a	n/a
(a, b)	n/a	n/a	n/a
(b, b)	n/a	n/a	n/a
(a, Z)	PUSH(a)	a	intermediate
(a, a)	PUSH(a)	a	intermediate
(b, a)	POP	a or Z	intermediate
(b, Z)	none	Z	FAIL

Table 8.2: Transition table of PDA to accept well-formed brackets

To represent a PDA using a state diagram, the transitions are labelled as follows



where a is the input symbol
 b is the top-of-stack symbol
 c is stack operation

Note:

- If b is ‘_’, the rule applies regardless of the top-of-stack symbol.
- If c is ‘_’, no stack action is performed.

In Figure 8.1 we construct a state diagram for a PDA that recognises strings of well-formed brackets.

8.2 Deterministic Pushdown Automata

Definition 8.1. A *deterministic pushdown automaton* (DPDA) $M = (S, I, J, f, s_0, F)$ consists of



Figure 8.1: State diagram of PDA to accept strings of well-formed brackets

- (1) a finite set of *states* S ,
- (2) an *input* alphabet I ,
- (3) a *stack* alphabet J ,
- (4) a *start state* $s_0 \in S$,
- (5) a set of *final states* $F \subseteq S$ and
- (6) a *transition function*

$$f : S \times (I \cup \{\lambda\}) \times J \rightarrow S \times J^*$$

Definition 8.2. A DPDA is said to *recognise* or *accept* an input string if, starting in its initial state, it is able to process the entire string and finish with an empty stack in one of its final states.

Note: Acceptance by empty stack and acceptance by final state are equivalent:

- A final state can perform a POP loop to get to an empty stack.
- A machine can detect an empty stack, and enter a final state by detecting a unique symbol pushed by the initial state.

Example 8.2. Construct a PDA that recognises strings over the alphabet $V = \{0, 1, 2\}$ of the form $w2w^r$, where w is a string consisting of 0s and 1s and w^r is equal to w in reverse order.

Solution:

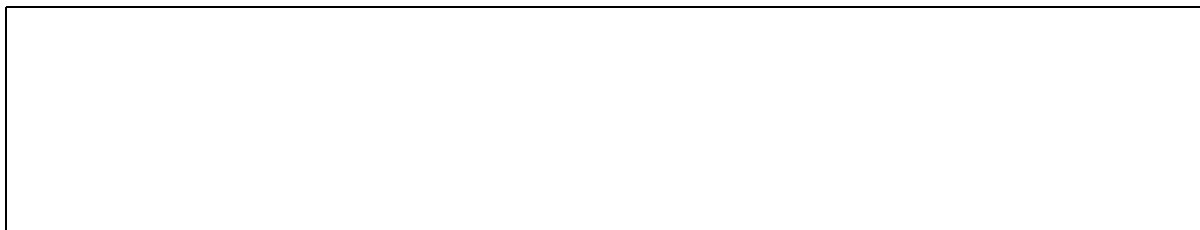


Figure 8.2: PDA to recognise strings of the form $w2w^r$ (Example 8.2)

8.3 Nondeterministic Pushdown Automata

Definition 8.3. A *nondeterministic pushdown automaton* (NPDA) $M = (S, I, J, f, s_0, F)$ consists of

- (1) a finite *set* of states S ,
- (2) an *input* alphabet I ,
- (3) a *stack* alphabet J ,
- (4) a *start state* $s_0 \in S$,
- (5) a set of *final states* $F \subseteq S$ and
- (6) a *transition function* f such that

$$f : S \times (I \cup \{\lambda\}) \times J \rightarrow \mathcal{P}(S \times J^*)$$

where $\mathcal{P}(S \times J^*)$ denotes the set of all subsets of $S \times J^*$.

Definition 8.4. A NPDA is said to *recognise* or *accept* an input string if, starting in its initial state, there is some sequence of transitions that allows it to read the entire string and finish in one of its final states with an empty stack.

Example 8.3. Construct a PDA that recognises strings over the alphabet $V = \{0, 1\}$ of the form ww^r , where w^r is equal to w in reverse order.

Solution:

- If a DPDA reads a string x of the form ww^r , it cannot know where to split x into w and w^r because it cannot see the entire string at once (and therefore does not know its length).
- A NPDA will check *all* possible ways of splitting x into w and w^r . If there is a ‘correct’ way to do this, then it will find it.

The NPDA shown in Figure 8.3 recognises all strings of the form ww^r :



Figure 8.3: PDA to recognise strings of the form ww^r (Example 8.3)

8.4 Context-free languages

- Kleene's theorem states that the class of languages accepted by FSA is exactly the class of regular languages.
- The following theorem states that the class of languages accepted by NPDA is exactly the class of *context-free* languages.

Theorem 8.1. A language is context-free if and only if it is recognised by a non-deterministic pushdown automaton.