COMP30026 Models of Computation

Pushdown Automata

Anna Kalenkova

Lecture Week 9

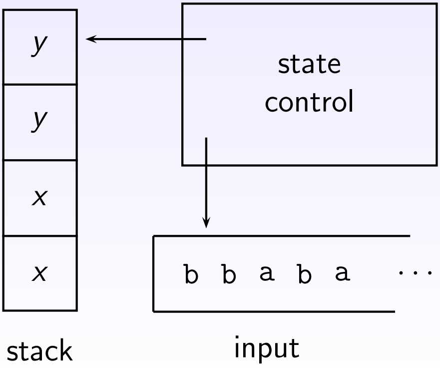
Semester 2, 2020

Pushdown Automata

The automata we saw so far were limited by their lack of memory.

A pushdown automaton (PDA) is a finite-state automaton, equipped with a stack.

The language $\{a^i b^i \mid i \geq 0\}$ is not recognised by any DFA, since it requires the ability of a recogniser to remember how many consecutive as have been consumed from the input.

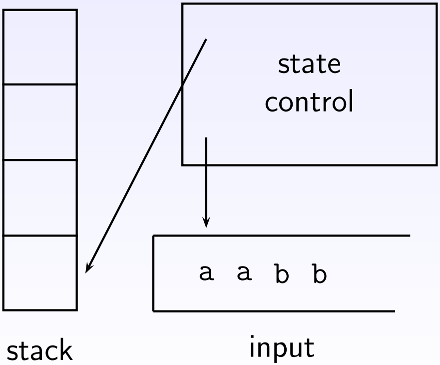


Pushdown Automata

The automata we saw so far were limited by their lack of memory.

A pushdown automaton (PDA) is a finite-state automaton, equipped with a stack.

The language $\{a^i b^i \mid i \geq 0\}$ is not recognised by any DFA, since it requires the ability of a recogniser to remember how many consecutive as have been consumed from the input.

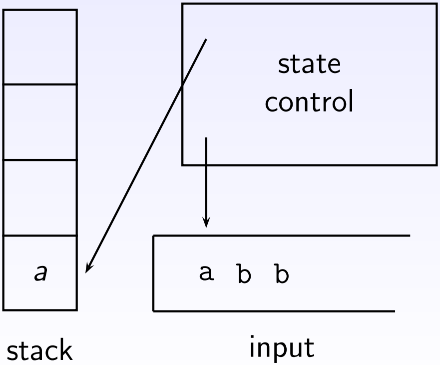


Pushdown Automata

The automata we saw so far were limited by their lack of memory.

A pushdown automaton (PDA) is a finite-state automaton, equipped with a stack.

The language $\{\mathtt{a}^i\mathtt{b}^i \mid i \geq 0\}$ is not recognised by any DFA, since it requires the ability of a recogniser to remember how many consecutive as have been consumed from the input.

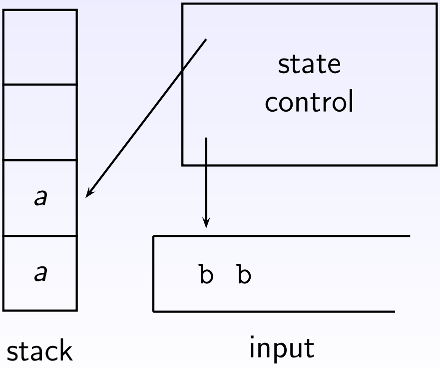


Pushdown Automata

The automata we saw so far were limited by their lack of memory.

A pushdown automaton (PDA) is a finite-state automaton, equipped with a stack.

The language $\{a^i b^i \mid i \geq 0\}$ is not recognised by any DFA, since it requires the ability of a recogniser to remember how many consecutive as have been consumed from the input.

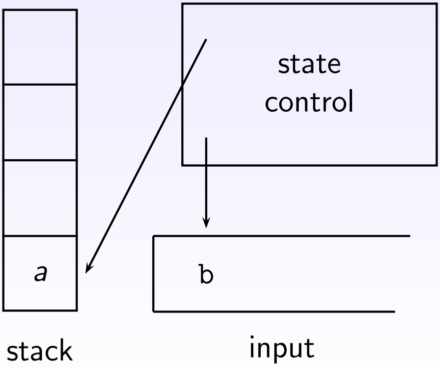


Pushdown Automata

The automata we saw so far were limited by their lack of memory.

A pushdown automaton (PDA) is a finite-state automaton, equipped with a stack.

The language $\{a^i b^i \mid i \geq 0\}$ is not recognised by any DFA, since it requires the ability of a recogniser to remember how many consecutive as have been consumed from the input.

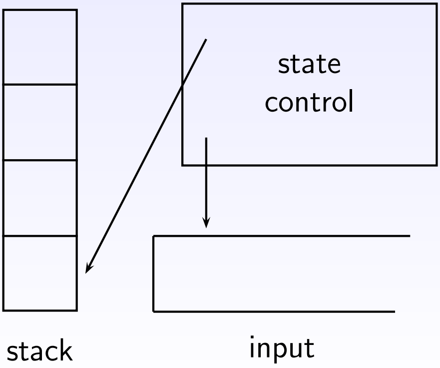


Pushdown Automata

The automata we saw so far were limited by their lack of memory.

A pushdown automaton (PDA) is a finite-state automaton, equipped with a stack.

The language $\{\mathtt{a}^i\mathtt{b}^i \mid i \geq 0\}$ is not recognised by any DFA, since it requires the ability of a recogniser to remember how many consecutive as have been consumed from the input.



Fine but Important Points

Based on (1) input symbol, (2) top stack symbol, and (3) the current state, PDA will decide which state to go to next, as well as, what operation apply to the stack.

In one transition step, PDA reads a symbol from input and pops the top stack symbol, or pushes to the stack, or both (replaces the top stack symbol).

We shall consider the non-deterministic version of a PDA.

It may also ignore the input.

Pushdown Automata Formally

A pushdown automaton is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where

- Q is a finite set of states,
- Σ is the finite input alphabet,
- Γ is the finite stack alphabet,
- $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to \mathcal{P}(Q \times \Gamma_\epsilon)$ is the transition function,
- $q_0 \in Q$ is the start state, and
- $F \subseteq Q$ are the accept states.

Example Transitions

$\delta(q_5, \mathtt{a}, \mathtt{b}) = \{(q_7, \epsilon)\}$ means:

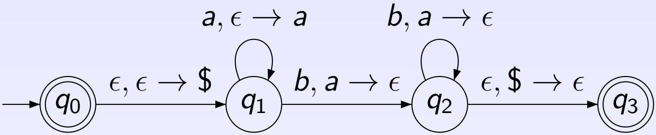
If in state q_5, when reading input symbol a, provided the top of the stack holds 'b', consume the a, pop the b, and go to state q_7.

$\delta(q_5, \epsilon, \mathtt{b}) = \{(q_6, \mathtt{a}), (q_7, \mathtt{b})\}$ means:

If in state q_5, and if the top of the stack holds 'b', either replace that b by a and go to state q_6, or leave the stack as is and go to state q_7. In either case do not consume an input symbol.

PDA Example 1

This PDA recognises $\{a^n b^n \mid n \geq 0\}$:



- $Q = \{q_0, q_1, q_2, q_3\}$;
- $\Sigma = \{a, b\}$;
- $\Gamma = \{a, \$\}$;
- $\delta(q_0, \epsilon, \epsilon) = \{(q_1, \$)\}$,$\delta(q_1, a, \epsilon) = \{(q_1, a)\}$,
  $\delta(q_1, b, a) = \{(q_2, \epsilon)\}$,$\delta(q_2, b, a) = \{(q_2, \epsilon)\}$,
  $\delta(q_2, \epsilon, \$) = \{(q_3, \epsilon)\}$, for other inputs δ returns $\emptyset$;
- $q_0 = q_0$;
- $F = \{q_0, q_3\}$.

Acceptance Precisely

The PDA $(Q, \Sigma, \Gamma, \delta, q_0, F)$ accepts input w iff $w = v_1 v_2 \cdots v_n$ with each $v_i \in \Sigma_\epsilon$, and there are states $r_0, r_1, \ldots, r_n \in Q$ and strings $s_0, s_1, \ldots, s_n \in \Gamma^*$ such that

1. $r_0 = q_0$ and $s_0 = \epsilon$.
2. $(r_{i+1}, b) \in \delta(r_i, v_{i+1}, a)$, $s_i = at$, $s_{i+1} = bt$ with $a, b \in \Gamma_\epsilon$ and $t \in \Gamma_\epsilon^*$.
3. $r_n \in F$.

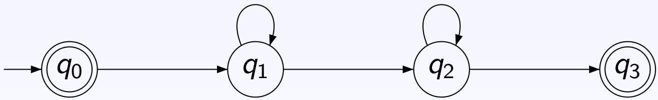
Note 1: There is no requirement that $s_n = \epsilon$, so the stack may be non-empty when the machine stops (even when it accepts).

Note 2: Trying to pop an empty stack leads to rejection of input, rather than "runtime error".

PDA Example 2

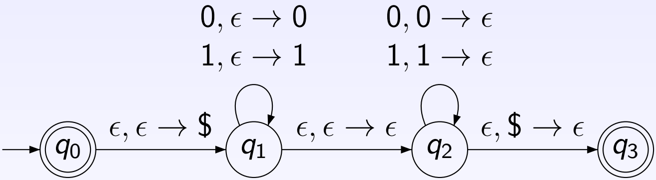
Let $w^{\mathcal{R}}$ denote the string w reversed.

Let us design a PDA to recognise $\{ww^{\mathcal{R}} \mid w \in \{0, 1\}^*\}$, the set of even-length binary palindromes:



PDA Example 2

This PDA recognises $\{ww^{\mathcal{R}} \mid w \in \{0,1\}^*\}$:



CFLs Have PDAs as Recognisers

Given a context-free language L (in the form of a grammar), we can find a PDA which recognises L.

And, every PDA recognises a context-free language.

We won't prove the second claim, but the first claim can easily be seen to hold.

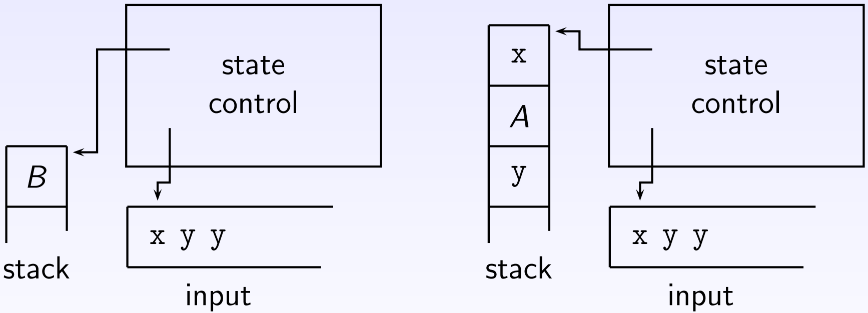
Namely, given a CFG G, we show how to construct a PDA P such that $L(P) = L(G)$.

The idea is to let the PDA use its stack to store a list of "pending" recogniser tasks.

The construction does not give the cleverest PDA, but it always works.

From Context-Free Grammars to PDAs

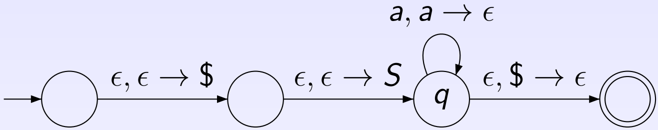
Say $B \rightarrow xAy$ is a rule in G, and the PDA finds the symbol B on top of its stack, it may pop B and push y, A, and x, in that order.



If it finds the terminal x on top of the stack, and x is the next input symbol, it may consume the input and pop x.

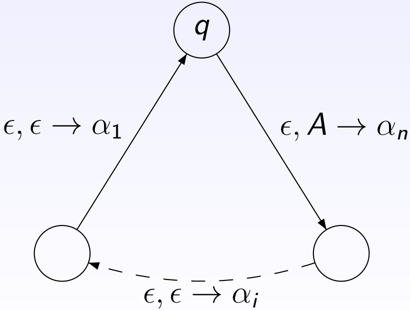
From Context-Free Grammars to PDAs

Construct the PDA like this:



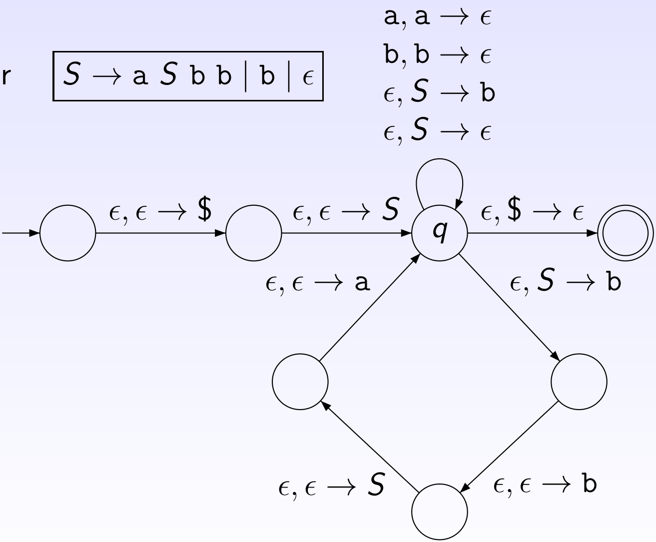
with a self-loop from q for each terminal a
(S is the grammar's start symbol).

For each rule $A \to \alpha_1 \ldots \alpha_n$,
add this loop from q to q:



Example Recogniser

For the grammar $\boxed{S \to \texttt{a}\ S\ \texttt{b}\ \texttt{b} \mid \texttt{b} \mid \epsilon}$



Pumping Lemma for CFLs

There are languages that are not context-free, and again there is a pumping lemma that can be used to show (some) languages non-context-free:

If A is context-free then there is a number p such that for any string $s \in A$ with $|s| \geq p$, s can be written as $s = uvxyz$, satisfying

1. $uv^ixy^iz \in A$ for all $i \geq 0$
2. $|vy| > 0$
3. $|vxy| \leq p$

We won't prove this lemma, but we give two examples of its use.

Pumping Example 1

$A = \{ww \mid w \in \{0,1\}^*\}$ is not context-free.

Assume it is, let p be the pumping length, take $0^p1^p0^p1^p$.

By the pumping lemma, $0^p1^p0^p1^p = uvxyz$, with uv^ixy^iz in A for all $i \geq 0$, and $|vxy| \leq p$.

There are three ways that vxy can be part of

$$00\ldots0011\ldots1100\ldots0011\ldots11$$

If it straddles the midpoint, it has form 1^n0^m, so pumping down, we are left with $0^p1^i0^j1^p$, with $i < p$, or $j < p$, or both.

If it is in the first half, uv^2xy^2z will have pushed a 1 into the first position of the second half.

Similarly if vxy is in the second half.

Pumping Example 2

$B = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \in \mathbb{N}\}$ is not context-free.

Assume it is, let p be the pumping length, and take $\mathtt{a}^p\mathtt{b}^p\mathtt{c}^p \in B$.

By the pumping lemma, $\mathtt{a}^p\mathtt{b}^p\mathtt{c}^p = uvxyz$, with uv^ixy^iz in B for all i.

Either v or y is non-empty (or both are).

If one of them contains two different symbols from $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ then uv^2xy^2z has symbols in the wrong order, and so cannot be in B.

So both v and y must contain only one kind of symbol. But then uv^2xy^2z can't have the same number of as, bs, and cs.

In all cases we have a contradiction.

Closure Properties for CFLs

The class of context-free languages is closed under

- union,
- concatenation,
- Kleene star,
- reversal.

Closure Properties for CFLs

The class of context-free languages is not closed under intersection!

Hence it is not closed under complement either (why?)

Consider these two CFLs:

$$\begin{array}{rcl} C & = & \{\mathtt{a}^m\mathtt{b}^n\mathtt{c}^n \mid m, n \in \mathbb{N}\} \\ D & = & \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^m \mid m, n \in \mathbb{N}\} \end{array}$$

Exercise: Prove that they are context-free!

But $C \cap D$ is the language $B = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \in \mathbb{N}\}$ which we just showed is not context-free.

However, we do have: If A is context-free and R is regular then $A \cap R$ is context-free.

Deterministic PDAs

Is a deterministic PDA (a DPDA) as powerful as a PDA?

No. A DPDA can recognise the context-free

$$\{wcw^{\mathcal{R}} \mid c \in \Sigma, w \in (\Sigma \setminus \{c\})^*\}$$

but not the context-free $\{ww^{\mathcal{R}} \mid w \in \Sigma^*\}$.

Intuitively a deterministic machine cannot know when the middle of the input has been reached. Suppose it gets input

000011000000000110000

A deterministic machine won't know when to start popping the stack.