Pushdown automata

Ashley Montanaro
ashley@cs.bris.ac.uk

Department of Computer Science, University of Bristol
Bristol, UK

10 March 2014

Pushdown automata

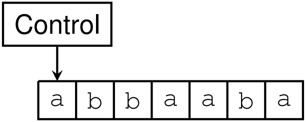
- You have seen that there are some languages which cannot be recognised by nondeterministic finite automata (NFAs).
- We now discuss a way of extending the concept of NFAs to make them more powerful, by adding access to a simple data storage device.

Pushdown automata

- You have seen that there are some languages which cannot be recognised by nondeterministic finite automata (NFAs).
- We now discuss a way of extending the concept of NFAs to make them more powerful, by adding access to a simple data storage device.
- This is an apparently simple extension which nevertheless significantly expands the range of recognisable languages.
- It also illustrates a close connection between a natural class of languages and a natural model of computation.

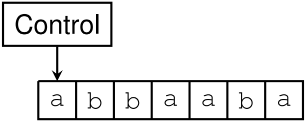
Pushdown automata

We can think of a finite automaton as follows:

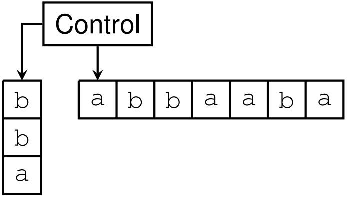


Pushdown automata

We can think of a finite automaton as follows:



A pushdown automaton (PDA) is a nondeterministic finite automaton which also has read/write access to a stack.



Pushdown automata

- The stack starts empty, grows downwards and the automaton has access to the top element.
- At each step, it can push an element onto the top of the stack and/or pop an element from the top of the stack.
- Based on what the top element is, the PDA can make different transitions.

Pushdown automata

- The stack starts empty, grows downwards and the automaton has access to the top element.
- At each step, it can push an element onto the top of the stack and/or pop an element from the top of the stack.
- Based on what the top element is, the PDA can make different transitions.
- This provides a simple kind of storage, allowing PDAs to do more than finite automata can.

Pushdown automata

- The stack starts empty, grows downwards and the automaton has access to the top element.
- At each step, it can push an element onto the top of the stack and/or pop an element from the top of the stack.
- Based on what the top element is, the PDA can make different transitions.
- This provides a simple kind of storage, allowing PDAs to do more than finite automata can.

We can have a special symbol `$`, which lets the PDA determine whether the stack is empty.

- The PDA starts out by pushing `$` onto the stack; at a later stage it can test whether `$` is at the top of the stack.

Example

- Imagine we want to recognise the language $\mathcal{L}_P$ of properly nested parentheses.
- That is, strings like:

$$(),\quad (()(())),\quad ((())()(())),\quad \dots$$

but not like:

$$)(,\quad (()()))),\quad ((())()(()),\quad \dots$$

Example

- Imagine we want to recognise the language $\mathcal{L}_P$ of properly nested parentheses.
- That is, strings like:

$$(),\quad (()(())),\quad ((())()(())),\quad \ldots$$

but not like:

$$)(,\quad (()())))),\quad ((())()(()),\quad \ldots$$

Characterisation of $\mathcal{L}_P$

$s \in \mathcal{L}_P$ if:

- at any point scanning along s, we have seen no more)'s than ('s;
- at the end of s, we have seen exactly as many)'s as ('s.

Example

- Imagine we want to recognise the language $\mathcal{L}_P$ of properly nested parentheses.
- That is, strings like:

$$\texttt{()},\quad \texttt{(()(()))},\quad \texttt{((())()(()))},\quad \ldots$$

but not like:

$$\texttt{)(},\quad \texttt{(()()))},\quad \texttt{((())()(())},\quad \ldots$$

Characterisation of $\mathcal{L}_P$

$s \in \mathcal{L}_P$ if:

- at any point scanning along s, we have seen no more $\texttt{)}$'s than $\texttt{(}$'s;
- at the end of s, we have seen exactly as many $\texttt{)}$'s as $\texttt{(}$'s.

This characterisation suggests a PDA for $\mathcal{L}_P$...

Example

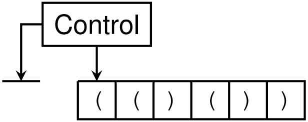
Determining whether $s \in \mathcal{L}_P$

1. Push `$` onto the stack.
2. Read each symbol of s in turn.
3. If it's a '`(`', push '`(`' onto the stack. If it's a '`)`', try to pop a '`(`' off the stack.
4. If the top element of the stack is `$` when we get to the end of s, accept.

Example

Determining whether $s \in \mathcal{L}_P$

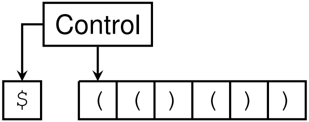
1. Push `$` onto the stack.
2. Read each symbol of *s* in turn.
3. If it's a '`(`', push '`(`' onto the stack. If it's a '`)`', try to pop a '`(`' off the stack.
4. If the top element of the stack is `$` when we get to the end of *s*, accept.



Example

Determining whether $s \in \mathcal{L}_P$

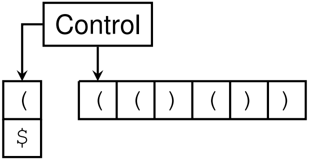
1. Push `$` onto the stack.
2. Read each symbol of *s* in turn.
3. If it's a '`(`', push '`(`' onto the stack. If it's a '`)`', try to pop a '`(`' off the stack.
4. If the top element of the stack is `$` when we get to the end of *s*, accept.



Example

Determining whether $s \in \mathcal{L}_P$

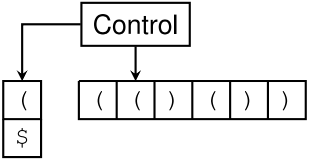
1. Push `$` onto the stack.
2. Read each symbol of *s* in turn.
3. If it's a '`(`', push '`(`' onto the stack. If it's a '`)`', try to pop a '`(`' off the stack.
4. If the top element of the stack is `$` when we get to the end of *s*, accept.



Example

Determining whether $s \in \mathcal{L}_P$

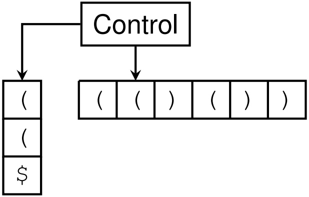
1. Push `$` onto the stack.
2. Read each symbol of *s* in turn.
3. If it's a '`(`', push '`(`' onto the stack. If it's a '`)`', try to pop a '`(`' off the stack.
4. If the top element of the stack is `$` when we get to the end of *s*, accept.



Example

Determining whether $s \in \mathcal{L}_P$

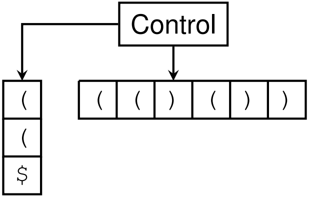
1. Push `$` onto the stack.
2. Read each symbol of *s* in turn.
3. If it's a '`(`', push '`(`' onto the stack. If it's a '`)`', try to pop a '`(`' off the stack.
4. If the top element of the stack is `$` when we get to the end of *s*, accept.



Example

Determining whether $s \in \mathcal{L}_P$

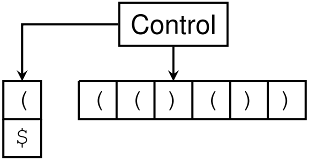
1. Push `$` onto the stack.
2. Read each symbol of *s* in turn.
3. If it's a '`(`', push '`(`' onto the stack. If it's a '`)`', try to pop a '`(`' off the stack.
4. If the top element of the stack is `$` when we get to the end of *s*, accept.



Example

Determining whether $s \in \mathcal{L}_P$

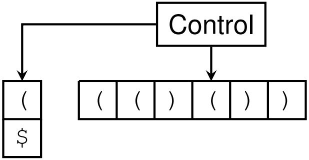
1. Push `$` onto the stack.
2. Read each symbol of *s* in turn.
3. If it's a '`(`', push '`(`' onto the stack. If it's a '`)`', try to pop a '`(`' off the stack.
4. If the top element of the stack is `$` when we get to the end of *s*, accept.



Example

Determining whether $s \in \mathcal{L}_P$

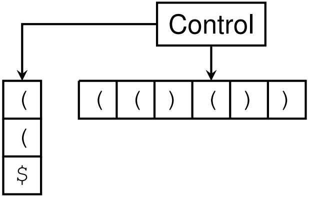
1. Push `$` onto the stack.
2. Read each symbol of *s* in turn.
3. If it's a '`(`', push '`(`' onto the stack. If it's a '`)`', try to pop a '`(`' off the stack.
4. If the top element of the stack is `$` when we get to the end of *s*, accept.



Example

Determining whether $s \in \mathcal{L}_P$

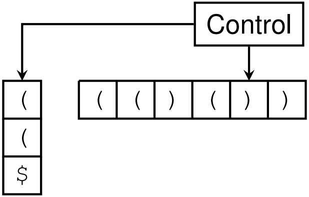
1. Push `$` onto the stack.
2. Read each symbol of *s* in turn.
3. If it's a '`(`', push '`(`' onto the stack. If it's a '`)`', try to pop a '`(`' off the stack.
4. If the top element of the stack is `$` when we get to the end of *s*, accept.



Example

Determining whether $s \in \mathcal{L}_P$

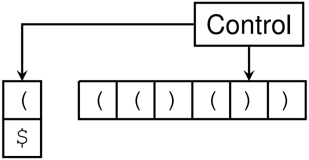
1. Push `$` onto the stack.
2. Read each symbol of s in turn.
3. If it's a '`(`', push '`(`' onto the stack. If it's a '`)`', try to pop a '`(`' off the stack.
4. If the top element of the stack is `$` when we get to the end of s, accept.



Example

Determining whether $s \in \mathcal{L}_P$

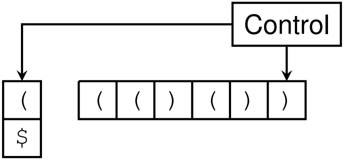
1. Push `$` onto the stack.
2. Read each symbol of *s* in turn.
3. If it's a '`(`', push '`(`' onto the stack. If it's a '`)`', try to pop a '`(`' off the stack.
4. If the top element of the stack is `$` when we get to the end of *s*, accept.



Example

Determining whether $s \in \mathcal{L}_P$

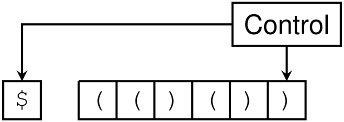
1. Push `$` onto the stack.
2. Read each symbol of *s* in turn.
3. If it's a '`(`', push '`(`' onto the stack. If it's a '`)`', try to pop a '`(`' off the stack.
4. If the top element of the stack is `$` when we get to the end of *s*, accept.



Example

Determining whether $s \in \mathcal{L}_P$

1. Push `$` onto the stack.
2. Read each symbol of *s* in turn.
3. If it's a '`(`', push '`(`' onto the stack. If it's a '`)`', try to pop a '`(`' off the stack.
4. If the top element of the stack is `$` when we get to the end of *s*, accept.



Describing pushdown automata

Just like DFAs / NFAs, PDAs can be described by their state diagrams.

- Each transition label is now of the form

$$\alpha, \beta \to \gamma$$

Describing pushdown automata

Just like DFAs / NFAs, PDAs can be described by their state diagrams.

- Each transition label is now of the form

$$\alpha, \beta \to \gamma$$

- This means:

IF input symbol is α AND β is on the top of the stack
- Make the transition
- Pop β off the stack
- Push γ onto the stack.

Describing pushdown automata

Just like DFAs / NFAs, PDAs can be described by their state diagrams.

- Each transition label is now of the form

$$\alpha,\ \beta \to \gamma$$

- This means:

IF input symbol is α AND β is on the top of the stack
- Make the transition
- Pop β off the stack
- Push γ onto the stack.

- Some special cases:
  - $\alpha,\ \beta \to \varepsilon$: Don't push anything onto the stack

Describing pushdown automata

Just like DFAs / NFAs, PDAs can be described by their state diagrams.

- Each transition label is now of the form

$$\alpha,\ \beta \to \gamma$$

- This means:

IF input symbol is α AND β is on the top of the stack
- Make the transition
- Pop β off the stack
- Push γ onto the stack.

- Some special cases:
  - $\alpha,\ \beta \to \varepsilon$: Don't push anything onto the stack
  - $\alpha,\ \varepsilon \to \gamma$: Don't pop anything from the stack

Describing pushdown automata

Just like DFAs / NFAs, PDAs can be described by their state diagrams.

- Each transition label is now of the form

$$\alpha,\ \beta \to \gamma$$

- This means:

IF input symbol is α AND β is on the top of the stack
  - Make the transition
  - Pop β off the stack
  - Push γ onto the stack.

- Some special cases:
  - $\alpha,\ \beta \to \varepsilon$: Don't push anything onto the stack
  - $\alpha,\ \varepsilon \to \gamma$: Don't pop anything from the stack
  - $\varepsilon,\ \beta \to \gamma$: Don't read any input

Describing pushdown automata

Just like DFAs / NFAs, PDAs can be described by their state diagrams.

- Each transition label is now of the form

$$\alpha,\ \beta \to \gamma$$

- This means:

  IF input symbol is α AND β is on the top of the stack
  - Make the transition
  - Pop β off the stack
  - Push γ onto the stack.

- Some special cases:
  - $\alpha,\ \beta \to \varepsilon$: Don't push anything onto the stack
  - $\alpha,\ \varepsilon \to \gamma$: Don't pop anything from the stack
  - $\varepsilon,\ \beta \to \gamma$: Don't read any input

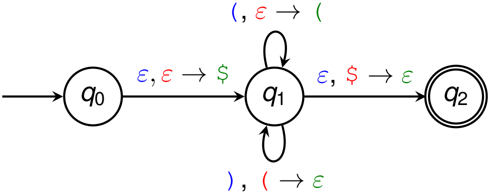
- Just like NFAs, PDAs are nondeterministic: the PDA accepts if any sequence of transitions terminates in an accepting state.

Ashley Montanaro
ashley@cs.bris.ac.uk

The PDA for $\mathcal{L}_P$

Determining whether $s \in \mathcal{L}_P$

1. Push `$` onto the stack.
2. Read each symbol of *s* in turn.
3. If it's a '`(`', push '`(`' onto the stack. If it's a '`)`', try to pop a '`(`' off the stack.
4. If the top element of the stack is `$` when we get to the end of *s*, accept.



PDAs: formal definition

Definition

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

1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the stack alphabet,
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $F \subseteq Q$ is the set of accept states.

Recall that $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$.

Ashley Montanaro
`ashley@cs.bris.ac.uk`

University of BRISTOL

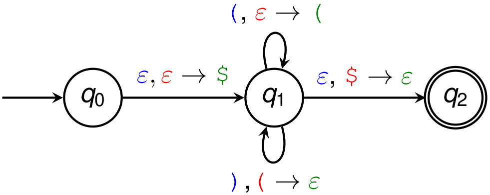
PDAs: formal definition

A PDA P defined as above accepts input w if w can be written as $w = w_1 \dots w_m$ for some m, where $w_i \in \Sigma_\varepsilon$, and there exist sequences $r_0, \dots, r_m \in Q$ and strings $s_0, \dots, s_m \in \Gamma^*$ satisfying:

1. $r_0 = q_0$ and $s_0 = \varepsilon$ (P starts in the start state with an empty stack)
2. For each i, $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, where $s_i = at$ and $s_{i+1} = bt$ for some $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$ (P moves properly according to its transition function)
3. $r_m \in F$ (an accept state occurs at the end of the input)

Example

Let N be the following PDA:



The formal description of N is:

$$N = (\ \{q_0, q_1, q_2\},\ \{\,(\,,\,)\,\},\ \{\,(\,,\,)\,,\$\},\ \delta,\ q_0,\ \{q_2\}\ )$$

where δ is the transition function defined by the table

Input:	(				)				ε			
Stack:	(	)	\$	ε	(	)	\$	ε	(	)	\$	ε
q_0												$\{(q_1, \$)\}$
q_1				$\{(q_1, ()\}$	$\{(q_1, \varepsilon)\}$						$\{(q_2, \varepsilon)\}$	
q_2												

Ashley Montanaro
ashley@cs.bris.ac.uk

Second example

How would we design a PDA to recognise the language

$$\mathcal{L} = \{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\}?$$

This is the language of strings containing a number of `a`'s followed by an equal number of `b`'s. So, for example:

$$\mathtt{aabb} \in \mathcal{L}, \quad \varepsilon \in \mathcal{L}, \quad \text{but} \quad \mathtt{abab} \notin \mathcal{L}.$$

Second example

How would we design a PDA to recognise the language

$$\mathcal{L} = \{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\}?$$

This is the language of strings containing a number of `a`'s followed by an equal number of `b`'s. So, for example:

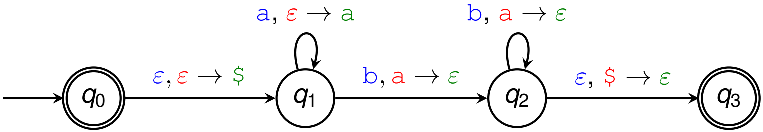
$$\mathtt{aabb} \in \mathcal{L}, \quad \varepsilon \in \mathcal{L}, \quad \text{but} \quad \mathtt{abab} \notin \mathcal{L}.$$

Idea for determining whether $s \in \mathcal{L}$

1. Start by reading `a`'s. For each `a` read, push it onto the stack.
2. When the first `b` is seen, switch to popping `a`'s off the stack. Pop one `a` off the stack for each `b` read.
3. If the stack is empty, accept.

Second example

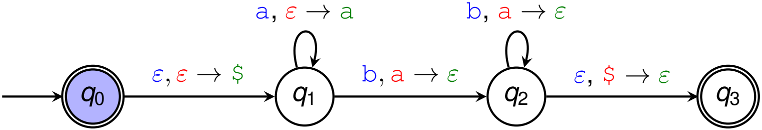
The following PDA implements the above idea.

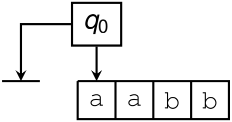


Note that it is nondeterministic.

Second example

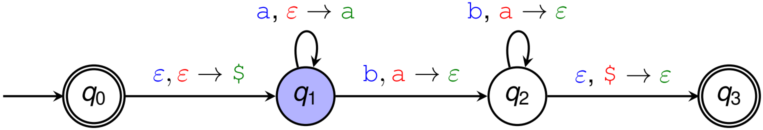
We track one path of this PDA's execution, demonstrating that it accepts the string $\texttt{aabb} \in \mathcal{L}$.

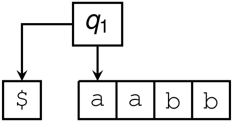




Second example

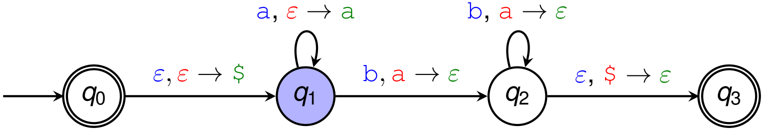
We track one path of this PDA's execution, demonstrating that it accepts the string $\texttt{aabb} \in \mathcal{L}$.

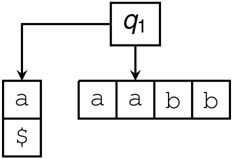




Second example

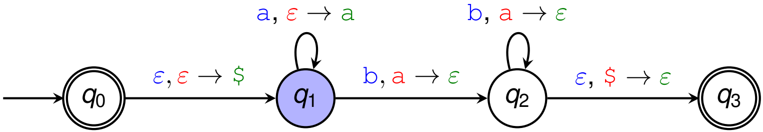
We track one path of this PDA's execution, demonstrating that it accepts the string aabb $\in \mathcal{L}$.

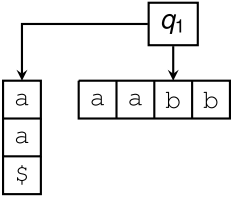




Second example

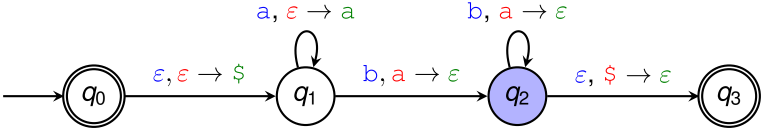
We track one path of this PDA's execution, demonstrating that it accepts the string $\mathtt{aabb} \in \mathcal{L}$.

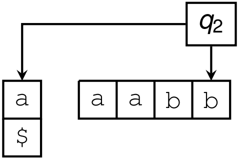




Second example

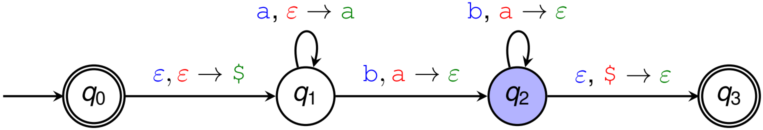
We track one path of this PDA's execution, demonstrating that it accepts the string $\texttt{aabb} \in \mathcal{L}$.

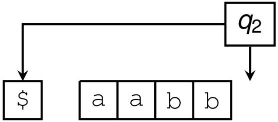




Second example

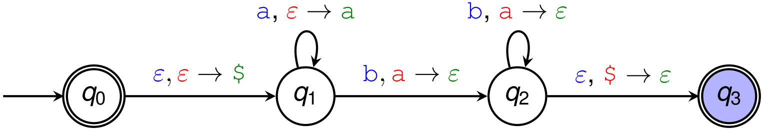
We track one path of this PDA's execution, demonstrating that it accepts the string $\mathtt{aabb} \in \mathcal{L}$.

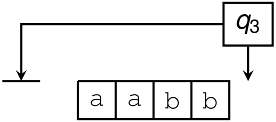




Second example

We track one path of this PDA's execution, demonstrating that it accepts the string aabb $\in \mathcal{L}$.





Second example

- By playing around with this PDA you should convince yourself that it does indeed recognise the language

$$\mathcal{L} = \{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\}$$

…although we won't formally prove this here.

- Recall that you showed, using the pumping lemma, that there is no finite automaton that recognises this language.

- Therefore, PDAs are more powerful than finite automata!

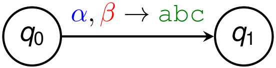
A slight generalisation of PDAs

One simple way in which we can generalise PDAs is by allowing them to push multiple symbols onto the stack.

A slight generalisation of PDAs

One simple way in which we can generalise PDAs is by allowing them to push multiple symbols onto the stack.

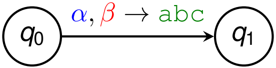
Imagine we would like to push the string `abc` onto the stack, which we could write as the transition



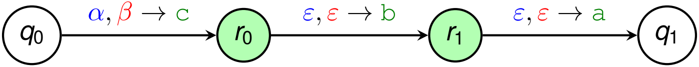
A slight generalisation of PDAs

One simple way in which we can generalise PDAs is by allowing them to push multiple symbols onto the stack.

Imagine we would like to push the string `abc` onto the stack, which we could write as the transition



We can split this into a sequence of transitions as follows:



Deterministic PDAs

- PDAs as we described them are intrinsically nondeterministic, but the concept of deterministic PDAs also makes sense.
- A deterministic PDA (DPDA) is a PDA which has at most one possible choice of transition to make at each step.

Deterministic PDAs

- PDAs as we described them are intrinsically nondeterministic, but the concept of deterministic PDAs also makes sense.
- A deterministic PDA (DPDA) is a PDA which has at most one possible choice of transition to make at each step.
- The transition function is of the form

  $$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow (Q \times \Gamma_\varepsilon) \cup \emptyset$$

  and for each $q \in Q$, $a \in \Sigma$ and $x \in \Gamma$, exactly one of

  $$\delta(q, a, x), \quad \delta(q, a, \varepsilon), \quad \delta(q, \varepsilon, x), \quad \delta(q, \varepsilon, \varepsilon)$$

  is not $\emptyset$.

Ashley Montanaro
`ashley@cs.bris.ac.uk`

Deterministic PDAs

- PDAs as we described them are intrinsically nondeterministic, but the concept of deterministic PDAs also makes sense.
- A deterministic PDA (DPDA) is a PDA which has at most one possible choice of transition to make at each step.
- The transition function is of the form

  $$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow (Q \times \Gamma_\varepsilon) \cup \emptyset$$

  and for each $q \in Q$, $a \in \Sigma$ and $x \in \Gamma$, exactly one of

  $$\delta(q, a, x), \quad \delta(q, a, \varepsilon), \quad \delta(q, \varepsilon, x), \quad \delta(q, \varepsilon, \varepsilon)$$

  is not $\emptyset$.
- Unlike the situation with DFAs and NFAs, it turns out that the class of languages recognised by DPDAs is a strict subset of that recognised by PDAs.

Summary and further reading

- A pushdown automaton (PDA) is a nondeterministic finite automaton equipped with a stack.
- Using a stack allows PDAs to recognise non-regular languages.
- PDAs can be described by state diagrams or by a more formal text description.
- They can be generalised by allowing the PDA to write multiple symbols to the stack.
- Further reading: Sipser §2.2 (for DPDAs: Sipser 3rd edition §2.4).

Ashley Montanaro
ashley@cs.bris.ac.uk