PDAs and CFGs

Ashley Montanaro
ashley@cs.bris.ac.uk

Department of Computer Science, University of Bristol
Bristol, UK

14 March 2014

PDAs and CFGs

One of the main reasons for studying PDAs is that there is a close connection between them and context-free grammars (CFGs).

Theorem

For any language $\mathcal{L}$, there exists a PDA which recognises $\mathcal{L}$ if and only if $\mathcal{L}$ is context-free.

PDAs and CFGs

One of the main reasons for studying PDAs is that there is a close connection between them and context-free grammars (CFGs).

Theorem

For any language $\mathcal{L}$, there exists a PDA which recognises $\mathcal{L}$ if and only if $\mathcal{L}$ is context-free.

The proof of the theorem is split into two parts:

1. If $\mathcal{L}$ is context-free, then there exists a PDA which recognises it.
2. If a PDA recognises $\mathcal{L}$, then there is a CFG which generates $\mathcal{L}$.

CFLs: a recap

Recall that a context-free grammar (CFG) is a set of rules like

$$\begin{aligned} S &\rightarrow \mathtt{a}S\mathtt{a} \mid \mathtt{b}T\mathtt{b} \\ T &\rightarrow T\mathtt{a} \mid \varepsilon \end{aligned}$$

generating strings like

$$S \rightarrow \mathtt{a}S\mathtt{a} \rightarrow \mathtt{ab}T\mathtt{ba} \rightarrow \mathtt{ab}T\mathtt{aba} \rightarrow \mathtt{ababa}$$

Variables are denoted by capital letters, terminals (input characters) by lower-case letters.

$\mathcal{L}$ is a context-free language (CFL) if all its strings can be produced by the application of a sequence of rules from some CFG.

Recognising CFLs

We want to show that, given a language which is generated by a CFG, it can be recognised by a PDA.

Our proof will take full advantage of the nondeterminism of PDAs.

Recognising CFLs

We want to show that, given a language which is generated by a CFG, it can be recognised by a PDA.

Our proof will take full advantage of the nondeterminism of PDAs.

Rough idea

1. Start out by pushing the start variable onto the stack.

Recognising CFLs

We want to show that, given a language which is generated by a CFG, it can be recognised by a PDA.

Our proof will take full advantage of the nondeterminism of PDAs.

Rough idea

1. Start out by pushing the start variable onto the stack.
2. Repeatedly guess substitutions that replace variables on the stack with new variables.

Recognising CFLs

We want to show that, given a language which is generated by a CFG, it can be recognised by a PDA.

Our proof will take full advantage of the nondeterminism of PDAs.

Rough idea

1. Start out by pushing the start variable onto the stack.
2. Repeatedly guess substitutions that replace variables on the stack with new variables.
3. We eventually end up with a string of terminal characters.

Recognising CFLs

We want to show that, given a language which is generated by a CFG, it can be recognised by a PDA.

Our proof will take full advantage of the nondeterminism of PDAs.

Rough idea

1. Start out by pushing the start variable onto the stack.
2. Repeatedly guess substitutions that replace variables on the stack with new variables.
3. We eventually end up with a string of terminal characters.
4. Accept if this string matches the input string.

Recognising CFLs

We want to show that, given a language which is generated by a CFG, it can be recognised by a PDA.

Our proof will take full advantage of the nondeterminism of PDAs.

Rough idea

1. Start out by pushing the start variable onto the stack.
2. Repeatedly guess substitutions that replace variables on the stack with new variables.
3. We eventually end up with a string of terminal characters.
4. Accept if this string matches the input string.

A problem with this idea: it seems that we need to replace variables midway down the stack.

Recognising CFLs

We can deal with this issue by first popping any terminal symbols off the stack before replacing any variables.

Recognising CFLs

We can deal with this issue by first popping any terminal symbols off the stack before replacing any variables.

Recognising a CFL on a PDA

1. Push $\$$, followed by the start variable, onto the stack.

Recognising CFLs

We can deal with this issue by first popping any terminal symbols off the stack before replacing any variables.

Recognising a CFL on a PDA

1. Push `$`, followed by the start variable, onto the stack.
2. Repeat the following test forever. If the top of the stack is…

Recognising CFLs

We can deal with this issue by first popping any terminal symbols off the stack before replacing any variables.

Recognising a CFL on a PDA

1. Push $, followed by the start variable, onto the stack.
2. Repeat the following test forever. If the top of the stack is...
   - ...a variable A, pop A off the stack and substitute it with the right-hand side of one of the rules for A.

Recognising CFLs

We can deal with this issue by first popping any terminal symbols off the stack before replacing any variables.

Recognising a CFL on a PDA

1. Push `$`, followed by the start variable, onto the stack.
2. Repeat the following test forever. If the top of the stack is...
   - ...a variable A, pop A off the stack and substitute it with the right-hand side of one of the rules for A.
   - ...a terminal symbol `a`, read the next input symbol and compare it with `a`. If they do not match, reject.

Ashley Montanaro
ashley@cs.bris.ac.uk

Recognising CFLs

We can deal with this issue by first popping any terminal symbols off the stack before replacing any variables.

Recognising a CFL on a PDA

1. Push `$`, followed by the start variable, onto the stack.
2. Repeat the following test forever. If the top of the stack is. . .
   - . . . a variable A, pop A off the stack and substitute it with the right-hand side of one of the rules for A.
   - . . . a terminal symbol `a`, read the next input symbol and compare it with `a`. If they do not match, reject.
   - . . . the symbol `$`, and the input has all been read, accept.

Recognising CFLs

We can deal with this issue by first popping any terminal symbols off the stack before replacing any variables.

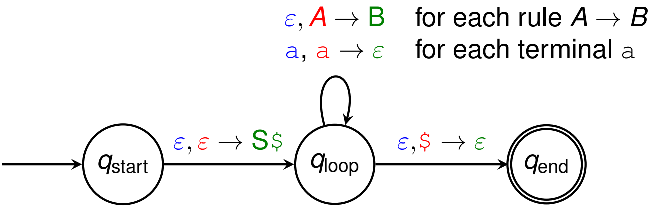
Recognising a CFL on a PDA

1. Push `$`, followed by the start variable, onto the stack.
2. Repeat the following test forever. If the top of the stack is...
   - ...a variable A, pop A off the stack and substitute it with the right-hand side of one of the rules for A.
   - ...a terminal symbol `a`, read the next input symbol and compare it with `a`. If they do not match, reject.
   - ...the symbol `$`, and the input has all been read, accept.

This algorithm can be described by a PDA with only three states!

Recognising CFLs

This is described by the diagram



- Here S is the start variable.
- The stack alphabet is given by the union of the set of terminal symbols, the set of variable symbols, and $\{\$\}$.

Example

Consider the language described by the following CFG:

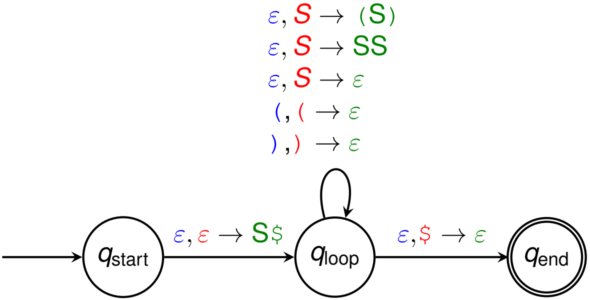
$$S \rightarrow (S) \mid SS \mid \varepsilon$$

Example

Consider the language described by the following CFG:

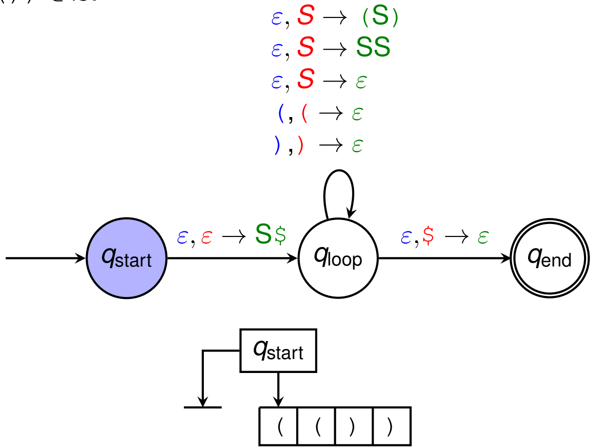
$$S \rightarrow (S) \mid SS \mid \varepsilon$$

Using this construction, we get the following (generalised) PDA:

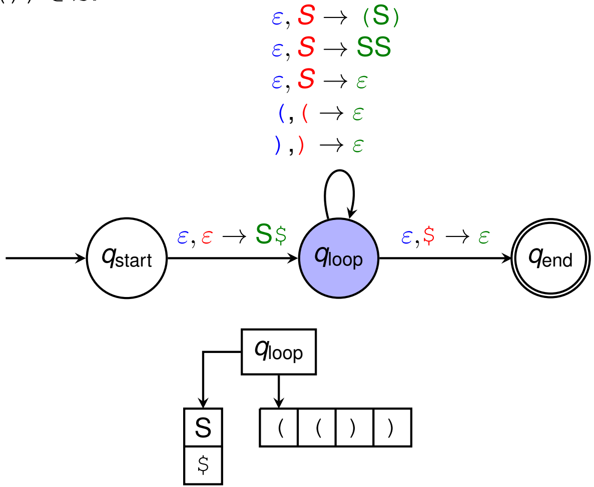


We can then modify the PDA as before to replace the transitions which push multiple symbols onto the stack.

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

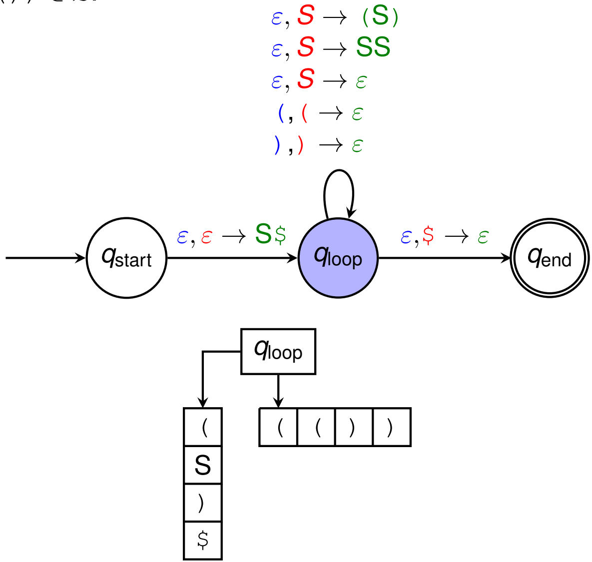


We track one path of this PDA's execution, demonstrating that it accepts the string (( )) $\in \mathcal{L}$.

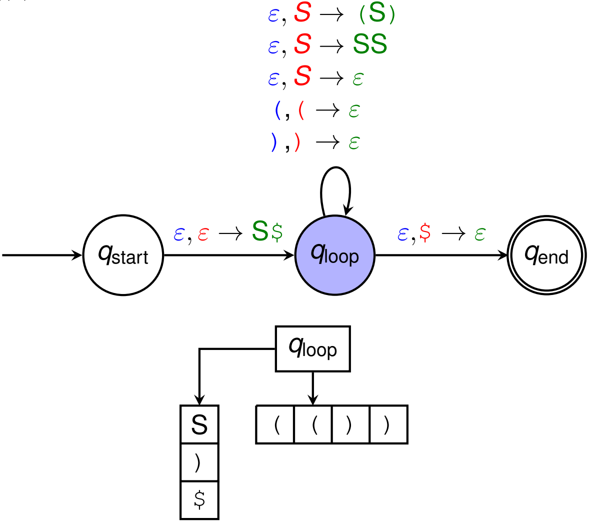


Ashley Montanaro
ashley@cs.bris.ac.uk

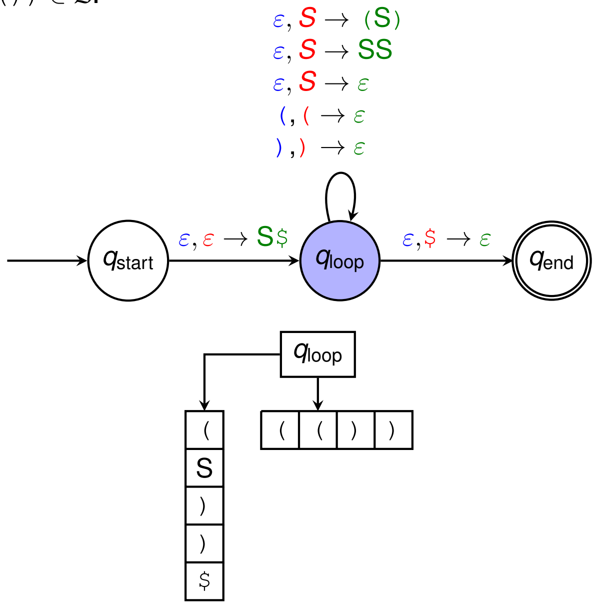
We track one path of this PDA's execution, demonstrating that it accepts the string (( )) $\in \mathcal{L}$.



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

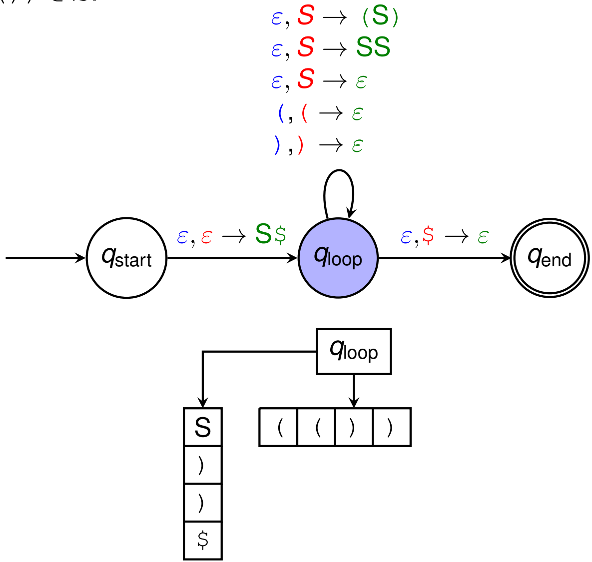


We track one path of this PDA's execution, demonstrating that it accepts the string (( )) $\in \mathcal{L}$.

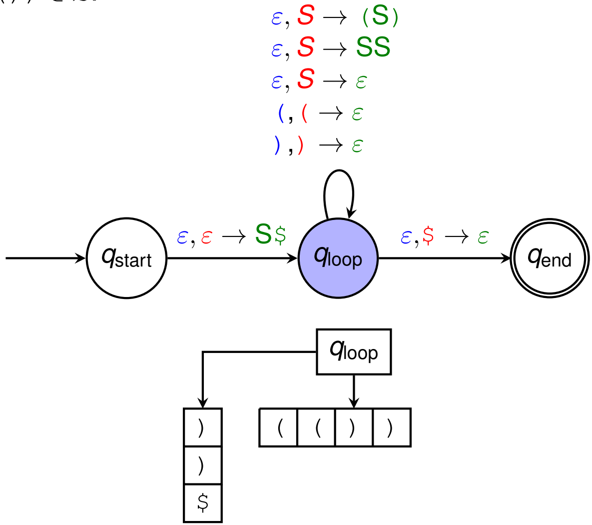


University of BRISTOL

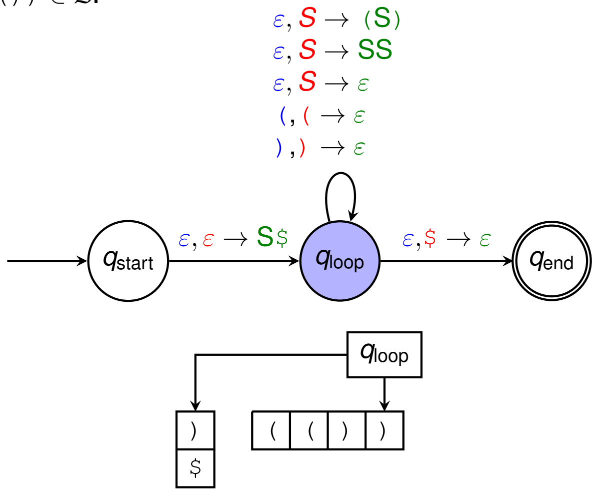
We track one path of this PDA's execution, demonstrating that it accepts the string (( )) $\in \mathcal{L}$.



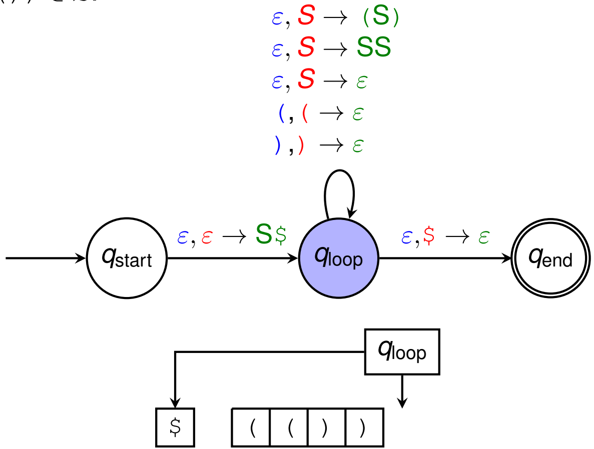
We track one path of this PDA's execution, demonstrating that it accepts the string (( )) $\in \mathcal{L}$.



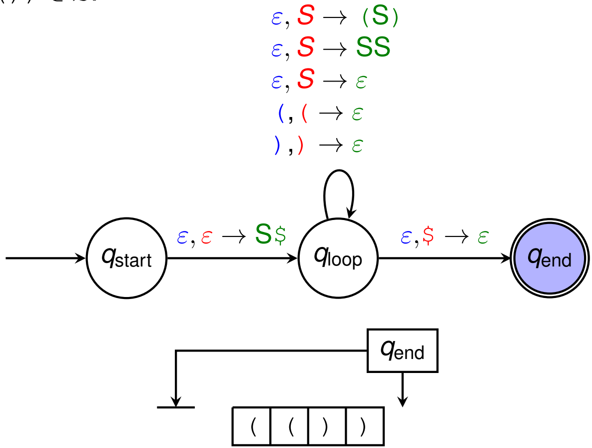
We track one path of this PDA's execution, demonstrating that it accepts the string (( )) $\in \mathcal{L}$.



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



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



The other direction: PDAs to CFGs

- We now need to show that, given a PDA which recognises some language $\mathcal{L}$, there exists a context-free grammar for $\mathcal{L}$.

The other direction: PDAs to CFGs

- We now need to show that, given a PDA which recognises some language $\mathcal{L}$, there exists a context-free grammar for $\mathcal{L}$.

- In other words: given a PDA P, we need to make a CFG G that generates all the strings that P accepts.

The other direction: PDAs to CFGs

- We now need to show that, given a PDA which recognises some language $\mathcal{L}$, there exists a context-free grammar for $\mathcal{L}$.
- In other words: given a PDA P, we need to make a CFG G that generates all the strings that P accepts.
- We first simplify P in several ways:
  1. We make it have only one accept state, q_{accept};

The other direction: PDAs to CFGs

- We now need to show that, given a PDA which recognises some language $\mathcal{L}$, there exists a context-free grammar for $\mathcal{L}$.
- In other words: given a PDA P, we need to make a CFG G that generates all the strings that P accepts.
- We first simplify P in several ways:
  1. We make it have only one accept state, q_{accept};
  2. We make it have an empty stack when it accepts;

The other direction: PDAs to CFGs

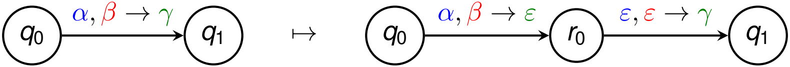
- We now need to show that, given a PDA which recognises some language $\mathcal{L}$, there exists a context-free grammar for $\mathcal{L}$.
- In other words: given a PDA P, we need to make a CFG G that generates all the strings that P accepts.
- We first simplify P in several ways:
  1. We make it have only one accept state, q_{accept};
  2. We make it have an empty stack when it accepts;
  3. Each transition either pushes a symbol onto the stack, or pops a symbol off, but not both.

The other direction: PDAs to CFGs

- We now need to show that, given a PDA which recognises some language $\mathcal{L}$, there exists a context-free grammar for $\mathcal{L}$.
- In other words: given a PDA P, we need to make a CFG G that generates all the strings that P accepts.
- We first simplify P in several ways:
  1. We make it have only one accept state, q_{accept};
  2. We make it have an empty stack when it accepts;
  3. Each transition either pushes a symbol onto the stack, or pops a symbol off, but not both.
- For (1), we add a new state with $\varepsilon, \varepsilon \to \varepsilon$ transitions from all the accepting states.
- For (2): exercise!

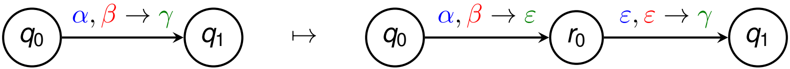
Push or pop, but not both

We can replace each transition that both pushes and pops as follows:

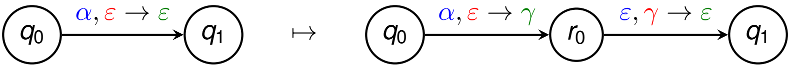


Push or pop, but not both

We can replace each transition that both pushes and pops as follows:



We can replace each transition that neither pushes nor pops as follows:



Ashley Montanaro
ashley@cs.bris.ac.uk

The basic idea

We construct a CFG G from P as follows:

- For every pair of states q and r in P, we have a variable A_{qr}.
- We will define the rules of G so that A_{qr} generates all the strings which can transform P from state q (with an empty stack) to state r (with an empty stack).
- Then, if q_{start} is the start state and q_{accept} is the end state, we have $A_{q_{\text{start}} q_{\text{accept}}}$ as the start variable in G.
- Then a string x is derivable from G if and only if P accepts x.

The rules of G

Any sequence of transitions from q to r such that the stack is empty at the start and at the end must begin with a push and end with a pop.

- Either the first symbol pushed onto the stack is popped off the stack at the end, or popped off the stack somewhere in the middle. If the former, the stack isn't empty until the end; if the latter, the stack is empty when the first symbol is popped.

The rules of G

Any sequence of transitions from q to r such that the stack is empty at the start and at the end must begin with a push and end with a pop.

- Either the first symbol pushed onto the stack is popped off the stack at the end, or popped off the stack somewhere in the middle. If the former, the stack isn't empty until the end; if the latter, the stack is empty when the first symbol is popped.
- The first case corresponds to the rule $A_{qr} \to \mathtt{a} A_{st} \mathtt{b}$, where $\mathtt{a}$ is the input symbol read in state q, s is the state following q, t is the state preceding r, and $\mathtt{b}$ is the symbol read in state t.
- The second case corresponds to the rule $A_{qr} \to A_{qu} A_{ur}$, where u is the state where the stack becomes empty.

More formally

Assume we have a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$.

The CFG G corresponding to P

- The variables are $\{A_{qr} \mid q, r \in Q\}$
- The start variable is $A_{q_0 q_{\text{accept}}}$

More formally

Assume we have a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$.

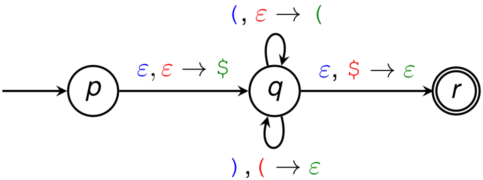
The CFG G corresponding to P

- The variables are $\{A_{qr} \mid q, r \in Q\}$
- The start variable is $A_{q_0 q_{\text{accept}}}$
- The rules are:
  - For each $p, q, r, s \in Q$, $\mathtt{t} \in \Gamma$, and $\mathtt{a}, \mathtt{b} \in \Sigma_\varepsilon$:
    if $(r, \mathtt{t}) \in \delta(p, \mathtt{a}, \varepsilon)$ and $(q, \varepsilon) \in \delta(s, \mathtt{b}, \mathtt{t})$, $A_{pq} \to \mathtt{a} A_{rs} \mathtt{b}$
  - For each $p, q, r \in Q$: $A_{pq} \to A_{pr} A_{rq}$
  - For each $p \in Q$: $A_{pp} \to \varepsilon$

More informally, the first case is where there is a transition from p to r which pushes $\mathtt{t}$ and a transition from s to q which pops $\mathtt{t}$.

Example

We convert the following PDA into a CFG:



1. We have 9 variables: A_{pp}, A_{pq}, A_{pr}, A_{qp}, A_{qq}, A_{qr}, A_{rp}, A_{rq}, A_{rr}.
2. The start variable is A_{pr}.
3. We have the following rules:

$$\begin{aligned} A_{pr} &\to \varepsilon A_{qq} \varepsilon \\ A_{qq} &\to (A_{qq}) \end{aligned}$$

4. We also have the rule $A_{ac} \to A_{ab}A_{bc}$ for every triple of states (a, b, c).
5. Finally, we have the rules $A_{pp} \to \varepsilon$, $A_{qq} \to \varepsilon$, $A_{rr} \to \varepsilon$.

Proof of correctness

We will show that, for any states q and r, A_{qr} generates x if and only if x can take P from state q (with empty stack) to state r (with empty stack).

Proof of correctness

We will show that, for any states q and r, A_{qr} generates x if and only if x can take P from state q (with empty stack) to state r (with empty stack).

Claim (first direction)

If A_{qr} generates x, x can take P from state q (with an empty stack) to state r (with an empty stack).

Proof of correctness

We will show that, for any states q and r, A_{qr} generates x if and only if x can take P from state q (with empty stack) to state r (with empty stack).

Claim (first direction)

If A_{qr} generates x, x can take P from state q (with an empty stack) to state r (with an empty stack).

Proof sketch

We use induction on the number of steps in the derivation of x from A_{qr}.

- Base case: The only derivations with one step are of the form $A_{pp} \to \varepsilon$, for which this claim clearly holds.

Proof of correctness

We will show that, for any states q and r, A_{qr} generates x if and only if x can take P from state q (with empty stack) to state r (with empty stack).

Claim (first direction)

If A_{qr} generates x, x can take P from state q (with an empty stack) to state r (with an empty stack).

Proof sketch

We use induction on the number of steps in the derivation of x from A_{qr}.

- Base case: The only derivations with one step are of the form $A_{pp} \to \varepsilon$, for which this claim clearly holds.
- Inductive step: We assume the claim holds for derivations of length k, for some $k \geq 1$, and prove for derivations of length $k + 1$.

Ashley Montanaro
ashley@cs.bris.ac.uk

Proof of correctness

We will show that, for any states q and r, A_{qr} generates x if and only if x can take P from state q (with empty stack) to state r (with empty stack).

Claim (first direction)

If A_{qr} generates x, x can take P from state q (with an empty stack) to state r (with an empty stack).

Proof sketch

We use induction on the number of steps in the derivation of x from A_{qr}.

- Base case: The only derivations with one step are of the form $A_{pp} \to \varepsilon$, for which this claim clearly holds.
- Inductive step: We assume the claim holds for derivations of length k, for some $k \geq 1$, and prove for derivations of length $k+1$.
- So assume we have a derivation $A_{qr} \overset{*}{\Rightarrow} x$ of length $k+1$.

...

Proof of correctness

Claim (first direction)

If A_{qr} generates x, x can take P from state q (with an empty stack) to state r (with an empty stack).

Proof sketch

Consider the two possible forms of the start of this derivation of x:

1. $A_{qr} \Rightarrow \mathtt{a} A_{st} \mathtt{b}$.

2. $A_{qr} \Rightarrow A_{qs} A_{sr}$.

Proof of correctness

Claim (first direction)

If A_{qr} generates x, x can take P from state q (with an empty stack) to state r (with an empty stack).

Proof sketch

Consider the two possible forms of the start of this derivation of x:

1. $A_{qr} \Rightarrow \mathtt{a}A_{st}\mathtt{b}$. Then x is of the form $x = \mathtt{a}y\mathtt{b}$. By the inductive hypothesis, P can go from s to t with an empty stack.

2. $A_{qr} \Rightarrow A_{qs}A_{sr}$.

Proof of correctness

Claim (first direction)

If A_{qr} generates x, x can take P from state q (with an empty stack) to state r (with an empty stack).

Proof sketch

Consider the two possible forms of the start of this derivation of x:

1. $A_{qr} \Rightarrow \mathtt{a}A_{st}\mathtt{b}$. Then x is of the form $x = \mathtt{a}y\mathtt{b}$. By the inductive hypothesis, P can go from s to t with an empty stack. Because $A_{qr} \rightarrow \mathtt{a}A_{st}\mathtt{b}$ is a rule in G, there must be a transition of P from q to s which pushes some symbol $\mathtt{u}$ on reading $\mathtt{a}$, and also a transition from t to r which pops $\mathtt{u}$ on reading $\mathtt{b}$.
2. $A_{qr} \Rightarrow A_{qs}A_{sr}$.

Proof of correctness

Claim (first direction)

If A_{qr} generates x, x can take P from state q (with an empty stack) to state r (with an empty stack).

Proof sketch

Consider the two possible forms of the start of this derivation of x:

1. $A_{qr} \Rightarrow \mathtt{a}A_{st}\mathtt{b}$. Then x is of the form $x = \mathtt{a}y\mathtt{b}$. By the inductive hypothesis, P can go from s to t with an empty stack. Because $A_{qr} \rightarrow \mathtt{a}A_{st}\mathtt{b}$ is a rule in G, there must be a transition of P from q to s which pushes some symbol $\mathtt{u}$ on reading $\mathtt{a}$, and also a transition from t to r which pops $\mathtt{u}$ on reading $\mathtt{b}$.
2. $A_{qr} \Rightarrow A_{qs}A_{sr}$. Then $x = yz$ for strings y and z generated by A_{qs} and A_{sr}. The claim follows from the inductive hypothesis.

□

Proof of correctness

Claim (second direction)

If x can take P from state q (with an empty stack) to state r (with an empty stack), A_{qr} generates x.

Again the proof is by induction, this time on the number of steps in the computation. The proof is a formalisation of the description of the definition of G.

Proof of correctness

Claim (second direction)

If x can take P from state q (with an empty stack) to state r (with an empty stack), A_{qr} generates x.

Again the proof is by induction, this time on the number of steps in the computation. The proof is a formalisation of the description of the definition of G.

Proof sketch

- Base case:
- Inductive step:

Proof of correctness

Claim (second direction)

If x can take P from state q (with an empty stack) to state r (with an empty stack), A_{qr} generates x.

Again the proof is by induction, this time on the number of steps in the computation. The proof is a formalisation of the description of the definition of G.

Proof sketch

- Base case: The computation has 0 steps, starting and ending at some state p. We must have $x = \varepsilon$, and we have the rule $A_{pp} \to \varepsilon$.
- Inductive step:

Proof of correctness

Claim (second direction)

If x can take P from state q (with an empty stack) to state r (with an empty stack), A_{qr} generates x.

Again the proof is by induction, this time on the number of steps in the computation. The proof is a formalisation of the description of the definition of G.

Proof sketch

- Base case: The computation has 0 steps, starting and ending at some state p. We must have $x = \varepsilon$, and we have the rule $A_{pp} \to \varepsilon$.
- Inductive step: We split into two cases: Either the stack is empty only at the beginning and end of the computation, or it becomes empty in the middle.

...

Proof of correctness

Claim (second direction)

If x can take P from state q (with an empty stack) to state r (with an empty stack), A_{qr} generates x.

Proof sketch

- Case 1: Let the state of P after the first move be s, and the state before the last move be t. By the definition of G, it must contain a rule of the form $A_{qr} \rightarrow \mathtt{a} A_{st} \mathtt{b}$.

Proof of correctness

Claim (second direction)

If x can take P from state q (with an empty stack) to state r (with an empty stack), A_{qr} generates x.

Proof sketch

- Case 1: Let the state of P after the first move be s, and the state before the last move be t. By the definition of G, it must contain a rule of the form $A_{qr} \to \mathtt{a} A_{st} \mathtt{b}$. Write $x = \mathtt{a} y \mathtt{b}$. P must push some symbol $\mathtt{u}$ onto the stack at the start, and pop $\mathtt{u}$ at the end. By ignoring this symbol, on input y, P can bring s with an empty stack to t with an empty stack.

Proof of correctness

Claim (second direction)

If x can take P from state q (with an empty stack) to state r (with an empty stack), A_{qr} generates x.

Proof sketch

- Case 1: Let the state of P after the first move be s, and the state before the last move be t. By the definition of G, it must contain a rule of the form $A_{qr} \rightarrow \mathtt{a} A_{st} \mathtt{b}$. Write $x = \mathtt{a} y \mathtt{b}$. P must push some symbol $\mathtt{u}$ onto the stack at the start, and pop $\mathtt{u}$ at the end. By ignoring this symbol, on input y, P can bring s with an empty stack to t with an empty stack. By the inductive hypothesis, $A_{st} \stackrel{*}{\Rightarrow} y$, so $A_{qr} \stackrel{*}{\Rightarrow} x$.

Proof of correctness

Claim (second direction)

If x can take P from state q (with an empty stack) to state r (with an empty stack), A_{qr} generates x.

Proof sketch

- Case 1: Let the state of P after the first move be s, and the state before the last move be t. By the definition of G, it must contain a rule of the form $A_{qr} \to \mathtt{a}A_{st}\mathtt{b}$. Write $x = \mathtt{a}y\mathtt{b}$. P must push some symbol $\mathtt{u}$ onto the stack at the start, and pop $\mathtt{u}$ at the end. By ignoring this symbol, on input y, P can bring s with an empty stack to t with an empty stack. By the inductive hypothesis, $A_{st} \stackrel{*}{\Rightarrow} y$, so $A_{qr} \stackrel{*}{\Rightarrow} x$.
- Case 2: Let s be the state where the stack becomes empty, and split $x = yz$ for the parts read before and after reaching s.

Proof of correctness

Claim (second direction)

If x can take P from state q (with an empty stack) to state r (with an empty stack), A_{qr} generates x.

Proof sketch

- Case 1: Let the state of P after the first move be s, and the state before the last move be t. By the definition of G, it must contain a rule of the form $A_{qr} \rightarrow \mathtt{a} A_{st} \mathtt{b}$. Write $x = \mathtt{a} y \mathtt{b}$. P must push some symbol $\mathtt{u}$ onto the stack at the start, and pop $\mathtt{u}$ at the end. By ignoring this symbol, on input y, P can bring s with an empty stack to t with an empty stack. By the inductive hypothesis, $A_{st} \overset{*}{\Rightarrow} y$, so $A_{qr} \overset{*}{\Rightarrow} x$.
- Case 2: Let s be the state where the stack becomes empty, and split $x = yz$ for the parts read before and after reaching s. By inductive hypothesis, $A_{qs} \overset{*}{\Rightarrow} y$, $A_{sr} \overset{*}{\Rightarrow} z$.

Proof of correctness

Claim (second direction)

If x can take P from state q (with an empty stack) to state r (with an empty stack), A_{qr} generates x.

Proof sketch

- Case 1: Let the state of P after the first move be s, and the state before the last move be t. By the definition of G, it must contain a rule of the form $A_{qr} \rightarrow \mathtt{a}A_{st}\mathtt{b}$. Write $x = \mathtt{a}y\mathtt{b}$. P must push some symbol $\mathtt{u}$ onto the stack at the start, and pop $\mathtt{u}$ at the end. By ignoring this symbol, on input y, P can bring s with an empty stack to t with an empty stack. By the inductive hypothesis, $A_{st} \overset{*}{\Rightarrow} y$, so $A_{qr} \overset{*}{\Rightarrow} x$.
- Case 2: Let s be the state where the stack becomes empty, and split $x = yz$ for the parts read before and after reaching s. By inductive hypothesis, $A_{qs} \overset{*}{\Rightarrow} y$, $A_{sr} \overset{*}{\Rightarrow} z$. As $A_{qr} \rightarrow A_{qs}A_{sr}$ is in G, $A_{qr} \overset{*}{\Rightarrow} x$. $\square$

The final result

Putting these results together, we have shown:

Theorem

For any language $\mathcal{L}$, there exists a PDA which recognises $\mathcal{L}$ if and only if $\mathcal{L}$ is context-free.

The final result

Putting these results together, we have shown:

Theorem

For any language $\mathcal{L}$, there exists a PDA which recognises $\mathcal{L}$ if and only if $\mathcal{L}$ is context-free.

Corollary

The set of regular languages is strictly contained within the set of context-free languages.

(You had already seen the “strictness” part that there exist context-free languages which are not regular.)

The final result

Putting these results together, we have shown:

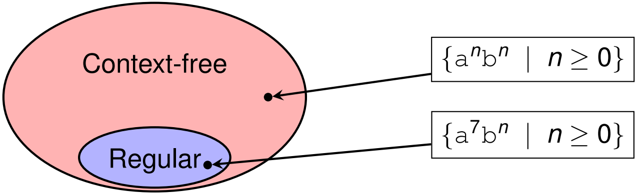
Theorem

For any language $\mathcal{L}$, there exists a PDA which recognises $\mathcal{L}$ if and only if $\mathcal{L}$ is context-free.

Corollary

The set of regular languages is strictly contained within the set of context-free languages.

(You had already seen the “strictness” part that there exist context-free languages which are not regular.)



Summary and further reading

- Given a language $\mathcal{L}$, there exists a PDA recognising $\mathcal{L}$ if and only if $\mathcal{L}$ is context-free.
- Given a CFG, we can write down a PDA recognising the corresponding language; given a PDA, we can write down a CFG generating the language that P recognises.
- The former direction is noticeably easier than the latter!
- Further reading: Sipser §2.2.