The Pumping Lemma

For every regular language L, there is a number $\ell \geq 1$ satisfying the **pumping lemma property**:

All $w \in L$ with $|w| \geq \ell$ can be expressed as a concatenation of three strings, $w = u_1 v u_2$, where u_1, v and u_2 satisfy:

- $|v| \geq 1$ (i.e. $v \neq \varepsilon$)
- $|u_1 v| \leq \ell$
- for all $n \geq 0$, $u_1 v^n u_2 \in L$
  (i.e. $u_1 u_2 \in L$, $u_1 v u_2 \in L$ [but we knew that anyway], $u_1 v v u_2 \in L$, $u_1 v v v u_2 \in L$, etc.)

Note similarity to construction in Kleene (B)

Suppose $L = L(M)$ for a DFA $M = (Q, \Sigma, \delta, s, F)$. Taking ℓ to be the number of elements in Q, if $n \geq \ell$, then in

$$s = \underbrace{q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots \xrightarrow{a_\ell} q_\ell}_{\ell+1 \text{ states}} \cdots \xrightarrow{a_n} q_n \in F$$

$q_0, \ldots, q_\ell$ can't all be distinct states. So $q_i = q_j$ for some $0 \leq i < j \leq \ell$. So the above transition sequence looks like

$$s = q_0 \xrightarrow{u_1}{}^* q_i = q_j \xrightarrow{u_2}{}^* q_n \in F$$

where

$$u_1 \triangleq a_1 \ldots a_i \qquad v \triangleq a_{i+1} \ldots a_j \qquad u_2 \triangleq a_{j+1} \ldots a_n$$

How to use the Pumping Lemma to prove that a language L is *not* regular

For each $\ell \geq 1$, find some $w \in L$ of length $\geq \ell$ so that

$$\left.\begin{array}{l}\text{no matter how } w \text{ is split into three, } w = u_1 v u_2, \\ \text{with } |u_1 v| \leq \ell \text{ and } |v| \geq 1 \text{, there is some } n \geq 0 \\ \text{for which } u_1 v^n u_2 \text{ is \textit{not} in } L\end{array}\right\} (\dagger)$$

Examples

None of the following three languages are regular:

(i) $L_1 \triangleq \{a^n b^n \mid n \geq 0\}$

$$\boxed{L_1 = \{a^n b^n \mid n \geq 0\}}$$

For each $\ell \geq 1$, take $w = a^\ell b^\ell \in L_1$

If $w = u_1 v u_2$ with $|u_1 v| \leq \ell$ & $|v| \geq 1$, then for some r and s:

- $u_1 = a^r$

$$\boxed{L_1 = \{a^n b^n \mid n \geq 0\}}$$

For each $\ell \geq 1$, take $w = a^\ell b^\ell \in L_1$

If $w = u_1 v u_2$ with $|u_1 v| \leq \ell$ & $|v| \geq 1$, then for some r and s:

- $u_1 = a^r$
- $v = a^s$, with $r + s \leq \ell$ and $s \geq 1$

$\boxed{L_1 = \{a^n b^n \mid n \geq 0\}}$

For each $\ell \geq 1$, take $w = a^\ell b^\ell \in L_1$

If $w = u_1 v u_2$ with $|u_1 v| \leq \ell$ & $|v| \geq 1$, then for some r and s:

- $u_1 = a^r$
- $v = a^s$, with $r + s \leq \ell$ and $s \geq 1$
- $u_2 = a^{l-r-s} b^\ell$

$\boxed{L_1 = \{a^n b^n \mid n \geq 0\}}$

For each $\ell \geq 1$, take $w = a^\ell b^\ell \in L_1$

If $w = u_1 v u_2$ with $|u_1 v| \leq \ell$ & $|v| \geq 1$, then for some r and s:

- $u_1 = a^r$
- $v = a^s$, with $r + s \leq \ell$ and $s \geq 1$
- $u_2 = a^{l-r-s} b^\ell$

so $u_1 v^0 u_2 =$

$$\boxed{L_1 = \{a^n b^n \mid n \geq 0\}}$$

For each $\ell \geq 1$, take $w = a^\ell b^\ell \in L_1$

If $w = u_1 v u_2$ with $|u_1 v| \leq \ell$ & $|v| \geq 1$, then for some r and s:

- $u_1 = a^r$
- $v = a^s$, with $r + s \leq \ell$ and $s \geq 1$
- $u_2 = a^{l-r-s} b^\ell$

so $u_1 v^0 u_2 = \quad a^r \, \epsilon \, a^{\ell-r-s} b^\ell =$

$\boxed{L_1 = \{a^n b^n \mid n \geq 0\}}$

For each $\ell \geq 1$, take $w = a^\ell b^\ell \in L_1$

If $w = u_1 v u_2$ with $|u_1 v| \leq \ell$ & $|v| \geq 1$, then for some r and s:

- $u_1 = a^r$
- $v = a^s$, with $r + s \leq \ell$ and $s \geq 1$
- $u_2 = a^{l-r-s} b^\ell$

so $u_1 v^0 u_2 = \quad a^r \, \epsilon \, a^{\ell-r-s} b^\ell = \quad a^{\ell-s} b^\ell$

$$\boxed{L_1 = \{a^n b^n \mid n \geq 0\}}$$

For each $\ell \geq 1$, take $w = a^\ell b^\ell \in L_1$

If $w = u_1 v u_2$ with $|u_1 v| \leq \ell$ & $|v| \geq 1$, then for some r and s:

- $u_1 = a^r$
- $v = a^s$, with $r + s \leq \ell$ and $s \geq 1$
- $u_2 = a^{l-r-s} b^\ell$

so $u_1 v^0 u_2 = \quad a^r \epsilon \; a^{\ell-r-s} b^\ell = \quad a^{\ell-s} b^\ell$

But $a^{\ell-s} b^\ell \notin L_1$

$$\boxed{L_1 = \{a^n b^n \mid n \geq 0\}}$$

For each $\ell \geq 1$, take $w = a^\ell b^\ell \in L_1$

If $w = u_1 v u_2$ with $|u_1 v| \leq \ell$ & $|v| \geq 1$, then for some r and s:

- $u_1 = a^r$
- $v = a^s$, with $r + s \leq \ell$ and $s \geq 1$
- $u_2 = a^{l-r-s} b^\ell$

so $u_1 v^0 u_2 = \quad a^r \, \epsilon \, a^{\ell-r-s} b^\ell = \quad a^{\ell-s} b^\ell$

But $a^{\ell-s} b^\ell \notin L_1$, so, By the Pumping Lemma, L_1 is not a regular language

Examples

None of the following three languages are regular:

(i) $L_1 \triangleq \{a^n b^n \mid n \geq 0\}$

[For each $\ell \geq 1$, $a^\ell b^\ell \in L_1$ is of length $\geq \ell$ and has property (†).]

Examples

None of the following three languages are regular:

(i) $L_1 \triangleq \{a^n b^n \mid n \geq 0\}$
[For each $\ell \geq 1$, $a^\ell b^\ell \in L_1$ is of length $\geq \ell$ and has property (†).]

(ii) $L_2 \triangleq \{w \in \{a, b\}^* \mid w \text{ a palindrome}\}$

Examples

None of the following three languages are regular:

(i) $L_1 \triangleq \{a^n b^n \mid n \geq 0\}$
[For each $\ell \geq 1$, $a^\ell b^\ell \in L_1$ is of length $\geq \ell$ and has property (†).]

(ii) $L_2 \triangleq \{w \in \{a, b\}^* \mid w$ a palindrome$\}$
[For each $\ell \geq 1$, $a^\ell b a^\ell \in L_1$ is of length $\geq \ell$ and has property (†).]

Examples

None of the following three languages are regular:

(i) $L_1 \triangleq \{a^n b^n \mid n \geq 0\}$
[For each $\ell \geq 1$, $a^\ell b^\ell \in L_1$ is of length $\geq \ell$ and has property (†).]

(ii) $L_2 \triangleq \{w \in \{a, b\}^* \mid w \text{ a palindrome}\}$
[For each $\ell \geq 1$, $a^\ell b a^\ell \in L_1$ is of length $\geq \ell$ and has property (†).]

(iii) $L_3 \triangleq \{a^p \mid p \text{ prime}\}$

$\boxed{L_3 = \{a^p \mid p \text{ prime}\}}$

For each $\ell \geq 1$ let $w = a^p \in L_3$, p prime & $p > 2\ell$

If $w = u_1 v u_2$ with the usual ...

$$\boxed{L_3 = \{a^p \mid p \text{ prime}\}}$$

For each $\ell \geq 1$ let $w = a^p \in L_3$, p prime & $p > 2\ell$

If $w = u_1 v u_2$ with the usual ...

then $u_1 = a^r \quad v = a^s \quad u_2 = a^{p-r-s}$

with $s \geq 1$ & $r + s \leq \ell$

$\boxed{L_3 = \{a^p \mid p \text{ prime}\}}$

For each $\ell \geq 1$ let $w = a^p \in L_3$, p prime & $p > 2\ell$

If $w = u_1 v u_2$ with the usual ...

then $u_1 = a^r \quad v = a^s \quad u_2 = a^{p-r-s}$

with $s \geq 1$ & $r + s \leq \ell$

so $u_1 v^{p-s} u_2 =$

$\boxed{L_3 = \{a^p \mid p \text{ prime}\}}$

For each $\ell \geq 1$ let $w = a^p \in L_3$, p prime & $p > 2\ell$

If $w = u_1 v u_2$ with the usual …

then $u_1 = a^r \quad v = a^s \quad u_2 = a^{p-r-s}$

with $s \geq 1$ & $r + s \leq \ell$

so $u_1 v^{p-s} u_2 = \quad a^r \, a^{s(p-s)} \quad a^{p-r-s} =$

$\boxed{L_3 = \{a^p \mid p \text{ prime}\}}$

For each $\ell \geq 1$ let $w = a^p \in L_3$, p prime & $p > 2\ell$

If $w = u_1 v u_2$ with the usual ...

then $u_1 = a^r \quad v = a^s \quad u_2 = a^{p-r-s}$

with $s \geq 1$ & $r + s \leq \ell$

so $u_1 v^{p-s} u_2 = \quad a^r \, a^{s(p-s)} \quad a^{p-r-s} = \quad a^{(p-s)(s+1)}$

$\boxed{L_3 = \{a^p \mid p \text{ prime}\}}$

For each $\ell \geq 1$ let $w = a^p \in L_3$, p prime & $p > 2\ell$

If $w = u_1 v u_2$ with the usual ...

then $u_1 = a^r \quad v = a^s \quad u_2 = a^{p-r-s}$

with $s \geq 1$ & $r + s \leq \ell$

so $u_1 v^{p-s} u_2 = \quad a^r \, a^{s(p-s)} \quad a^{p-r-s} = \quad a^{(p-s)(s+1)}$

But $s \geq 1 \Rightarrow s + 1 \geq 2$
and $(p - s) > (2\ell - \ell) \geq 1 \Rightarrow (p - s) \geq 2$

$$\boxed{L_3 = \{a^p \mid p \text{ prime}\}}$$

For each $\ell \geq 1$ let $w = a^p \in L_3$, p prime & $p > 2\ell$

If $w = u_1 v u_2$ with the usual ...

then $u_1 = a^r \quad v = a^s \quad u_2 = a^{p-r-s}$

with $s \geq 1$ & $r + s \leq \ell$

so $u_1 v^{p-s} u_2 = \quad a^r \; a^{s(p-s)} \quad a^{p-r-s} = \quad a^{(p-s)(s+1)}$

But $s \geq 1 \Rightarrow s + 1 \geq 2$
and $(p - s) > (2\ell - \ell) \geq 1 \Rightarrow (p - s) \geq 2$

so $a^{(p-s)(s+1)} \notin L_3$

Examples

None of the following three languages are regular:

(i) $L_1 \triangleq \{a^n b^n \mid n \geq 0\}$

[For each $\ell \geq 1$, $a^\ell b^\ell \in L_1$ is of length $\geq \ell$ and has property (†).]

(ii) $L_2 \triangleq \{w \in \{a, b\}^* \mid w$ a palindrome$\}$

[For each $\ell \geq 1$, $a^\ell b a^\ell \in L_1$ is of length $\geq \ell$ and has property (†).]

(iii) $L_3 \triangleq \{a^p \mid p$ prime$\}$

[For each $\ell \geq 1$, we can find a prime p with $p > 2\ell$ and then $a^p \in L_3$ has length $\geq \ell$ and has property (†).]

Pumping Lemma property is necessary for a language to be regular

It is not sufficient

Example of a non-regular language with the pumping lemma property

$$L \triangleq \{c^m a^n b^n \mid m \geq 1 \;\&\; n \geq 0\} \cup \{a^m b^n \mid m, n \geq 0\}$$

satisfies the pumping lemma property with $\ell = 1$.

[For any $w \in L$ of length ≥ 1, can take $u_1 = \varepsilon$, $v =$ first letter of w, $u_2 =$ rest of w.]

But L is not regular – see Exercise 5.1.

L is not regular: (sketch)

.

L is not regular: (sketch)

If L is regular there is a DFA M with $L = L(M)$.
Let's build a new machine, M' from it.

.

L is not regular: (sketch)

If L is regular there is a DFA M with $L = L(M)$. Let's build a new machine, M' from it.

Take a c transition from the start state of M. Make the state you reach the start state of M'.

L is not regular: (sketch)

If L is regular there is a DFA M with $L = L(M)$. Let's build a new machine, M' from it.

Take a c transition from the start state of M. Make the state you reach the start state of M'.

Delete all transitions involving c (and remove c from the alphabet). But don't remove any states and keep the same accept states.

L is not regular: (sketch)

If L is regular there is a DFA M with $L = L(M)$. Let's build a new machine, M' from it.

Take a c transition from the start state of M. Make the state you reach the start state of M'.

Delete all transitions involving c (and remove c from the alphabet). But don't remove any states and keep the same accept states.

What language does M' recognise?

The way ahead, in THEORY

- What does is mean for a function to be computable?
  [IB Computation Theory]

The way ahead, in THEORY

- What does is mean for a function to be computable?
  [IB Computation Theory]
- Are some computational tasks intrinsiclaly unfeasible?
  [IB ComplexityTheory]

The way ahead, in THEORY

- What does is mean for a function to be computable?
  [IB Computation Theory]
- Are some computational tasks intrinsiclaly unfeasible?
  [IB ComplexityTheory]
- How do we specify and reason about program behaviour?
  [IB Logic and Proof,
  IB Semantics of PLs]

The way ahead, in FORMAL LANGUAGE…

- Are there other useful language classes?

The way ahead, in FORMAL LANGUAGE

- Are there other useful language classes?
- Are there other useful automata classes that have a correspondence to them?

The way ahead, in FORMAL LANGUAGE

- Are there other useful language classes?
- Are there other useful automata classes that have a correspondence to them?
- What if we ask the same questions about them that we asked about regular languages?

Chomsky Hierarchy of Languages

Regular Languages
$\subset$ Context Free Languages
$\subset$ Context Sensitive Languages
$\subset$ Recursively Enumerable Languages

Grammars

Grammars are a shorthand way of expressing the inductive definition of subset inclusion for strings in a Language.

Grammars

Grammars are a shorthand way of expressing the inductive definition of subset inclusion for strings in a Language.

Often by convention we use capitals for non-terminal symbols (which are disjoint from symbols in the alphabet used by the language).

Grammars

Grammars are a shorthand way of expressing the inductive definition of subset inclusion for strings in a Language.

Often by convention we use capitals for non-terminal symbols (which are disjoint from symbols in the alphabet used by the language).

We also have productions (or production rules) of the form e.g. $A \rightarrow a$ which says that the non-terminal symbol A can be replaced by the (terminal) symbol a. More complex productions are allowed.

Grammars

Grammars are a shorthand way of expressing the inductive definition of subset inclusion for strings in a Language.

Often by convention we use capitals for non-terminal symbols (which are disjoint from symbols in the alphabet used by the language).

We also have productions (or production rules) of the form e.g. $A \rightarrow a$ which says that the non-terminal symbol A can be replaced by the (terminal) symbol a. More complex productions are allowed.

There is also a distinguished non-terminal called the goal symbol (we'll use G)

Everybody's favourite grammar

$$G \to E \quad \Delta_0$$
$$E \to E + T \quad \Delta_1$$
$$E \to T \quad \Delta_2$$
$$T \to T * P \quad \Delta_3$$
$$T \to P \quad \Delta_4$$
$$P \to (E) \quad \Delta_5$$
$$P \to x \quad \Delta_6$$

Everybody's favourite grammar

$G \rightarrow E \quad \boldsymbol{\Delta}_0$

$E \rightarrow E + T \quad \boldsymbol{\Delta}_1$

$E \rightarrow T \quad \boldsymbol{\Delta}_2$

$T \rightarrow T * P \quad \boldsymbol{\Delta}_3$

$T \rightarrow P \quad \boldsymbol{\Delta}_4$

$P \rightarrow (E) \quad \boldsymbol{\Delta}_5$

$P \rightarrow x \quad \boldsymbol{\Delta}_6$

so, e.g. $G \xrightarrow{\Delta_0} E$

Everybody's favourite grammar

$G \to E \quad \Delta_0$

$E \to E + T \quad \Delta_1$

$E \to T \quad \Delta_2$

$T \to T * P \quad \Delta_3$

$T \to P \quad \Delta_4$

$P \to (E) \quad \Delta_5$

$P \to x \quad \Delta_6$

so, e.g. $G \xrightarrow{\Delta_0} E \xrightarrow{\Delta_1} E + T$

Everybody's favourite grammar

$G \to E \quad \Delta_0$

$E \to E + T \quad \Delta_1$

$E \to T \quad \Delta_2$

$T \to T * P \quad \Delta_3$

$T \to P \quad \Delta_4$

$P \to (E) \quad \Delta_5$

$P \to x \quad \Delta_6$

so, e.g. $G \xrightarrow{\Delta_0} E \xrightarrow{\Delta_1} E + T \xrightarrow{\Delta_4} E + P$

Everybody's favourite grammar

$G \to E \quad \mathbf{\Delta_0}$

$E \to E + T \quad \mathbf{\Delta_1}$

$E \to T \quad \mathbf{\Delta_2}$

$T \to T * P \quad \mathbf{\Delta_3}$

$T \to P \quad \mathbf{\Delta_4}$

$P \to (E) \quad \mathbf{\Delta_5}$

$P \to x \quad \mathbf{\Delta_6}$

so, e.g. $G \xrightarrow{\mathbf{\Delta_0}} E \xrightarrow{\mathbf{\Delta_1}} E + T \xrightarrow{\mathbf{\Delta_4}} E + P \xrightarrow{\mathbf{\Delta_6}} E + x$

Everybody's favourite grammar

$G \to E \quad \mathbf{\Delta_0}$

$E \to E + T \quad \mathbf{\Delta_1}$

$E \to T \quad \mathbf{\Delta_2}$

$T \to T * P \quad \mathbf{\Delta_3}$

$T \to P \quad \mathbf{\Delta_4}$

$P \to (E) \quad \mathbf{\Delta_5}$

$P \to x \quad \mathbf{\Delta_6}$

so, e.g. $G \xrightarrow{\Delta_0} E \xrightarrow{\Delta_1} E + T \xrightarrow{\Delta_4} E + P \xrightarrow{\Delta_6} E + x \xrightarrow{\Delta_2}$
$T + x$

Everybody's favourite grammar

$G \to E \quad \Delta_0$

$E \to E + T \quad \Delta_1$

$E \to T \quad \Delta_2$

$T \to T * P \quad \Delta_3$

$T \to P \quad \Delta_4$

$P \to (E) \quad \Delta_5$

$P \to x \quad \Delta_6$

so, e.g. $G \xrightarrow{\Delta_0} E \xrightarrow{\Delta_1} E + T \xrightarrow{\Delta_4} E + P \xrightarrow{\Delta_6} E + x \xrightarrow{\Delta_2}$
$T + x \xrightarrow{\Delta_4} P + x$

Everybody's favourite grammar

$G \to E \quad \boldsymbol{\Delta_0}$

$E \to E + T \quad \boldsymbol{\Delta_1}$

$E \to T \quad \boldsymbol{\Delta_2}$

$T \to T * P \quad \boldsymbol{\Delta_3}$

$T \to P \quad \boldsymbol{\Delta_4}$

$P \to (E) \quad \boldsymbol{\Delta_5}$

$P \to x \quad \boldsymbol{\Delta_6}$

so, e.g. $G \xrightarrow{\Delta_0} E \xrightarrow{\Delta_1} E + T \xrightarrow{\Delta_4} E + P \xrightarrow{\Delta_6} E + x \xrightarrow{\Delta_2}$
$T + x \xrightarrow{\Delta_4} P + x \xrightarrow{\Delta_5} (E) + x$

Everybody's favourite grammar

$G \to E \quad \mathbf{\Delta_0}$

$E \to E + T \quad \mathbf{\Delta_1}$

$E \to T \quad \mathbf{\Delta_2}$

$T \to T * P \quad \mathbf{\Delta_3}$

$T \to P \quad \mathbf{\Delta_4}$

$P \to (E) \quad \mathbf{\Delta_5}$

$P \to x \quad \mathbf{\Delta_6}$

so, e.g. $G \xrightarrow{\Delta_0} E \xrightarrow{\Delta_1} E + T \xrightarrow{\Delta_4} E + P \xrightarrow{\Delta_6} E + x \xrightarrow{\Delta_2}$
$T + x \xrightarrow{\Delta_4} P + x \xrightarrow{\Delta_5} (E) + x \to \ldots \to (x + x) + x$

Everybody's favourite grammar

$G \to E \quad \Delta_0$

$E \to E + T \quad \Delta_1$

$E \to T \quad \Delta_2$

$T \to T * P \quad \Delta_3$

$T \to P \quad \Delta_4$

$P \to (E) \quad \Delta_5$

$P \to x \quad \Delta_6$

so, e.g. $G \xrightarrow{\Delta_0} E \xrightarrow{\Delta_1} E + T \xrightarrow{\Delta_4} E + P \xrightarrow{\Delta_6} E + x \xrightarrow{\Delta_2}$
$T + x \xrightarrow{\Delta_4} P + x \xrightarrow{\Delta_5} (E) + x \to \ldots \to (x + x) + x$
is a derivation of $(x + x) + x$

Language classes by forms of production

α, β, γ any strings of terminals and non-terminals

Language classes by forms of production

α, β, γ any strings of terminals and non-terminals

Context Free Languages: [Type 2]
productions of form $N \to \beta$

Language classes by forms of production

α, β, γ any strings of terminals and non-terminals

Context Free Languages: [Type 2]
productions of form $N \rightarrow \beta$

Context Sensitive Languages: [Type 1]
productions of the form $\alpha N \beta \rightarrow \alpha \gamma \beta$

Language classes by forms of production

α, β, γ any strings of terminals and non-terminals

Context Free Languages: [Type 2]
productions of form $N \to \beta$

Context Sensitive Languages: [Type 1]
productions of the form $\alpha N \beta \to \alpha \gamma \beta$

Recursively Enumerable Languages: [Type 0]
productions of the form $\alpha \to \beta$

Language classes by forms of production

Language classes by forms of production

How about Regular Languages? [Type 3]

Language classes by forms of production

How about Regular Languages? [Type 3]

A, B any non-terminals, a any terminal symbol, S any non-terminal that doesn't appear on right side

Language classes by forms of production

How about Regular Languages? [Type 3]

A, B any non-terminals, a any terminal symbol, S any non-terminal that doesn't appear on right side

production of the form $A \to a$ or $S \to \varepsilon$ or $A \to aB$ (right regular)

Language classes by forms of production

How about Regular Languages? [Type 3]

A, B any non-terminals, a any terminal symbol, S any non-terminal that doesn't appear on right side

production of the form $A \to a$ or $S \to \varepsilon$ or $A \to aB$ (right regular)

or of the form $A \to a$ or $S \to \varepsilon$ or $A \to Ba$ (left regular)

Language classes by forms of production

How about Regular Languages? [Type 3]

A, B any non-terminals, a any terminal symbol, S any non-terminal that doesn't appear on right side

production of the form $A \to a$ or $S \to \varepsilon$ or $A \to aB$ (right regular)

or of the form $A \to a$ or $S \to \varepsilon$ or $A \to Ba$ (left regular)

But not both left and right regular in the same grammar

Machines?????

- Regular Languages: Deterministic Finite Automata

Machines?????

- Regular Languages: Deterministic Finite Automata
- Context Free Languages: Nondeterministic Push-Down Automata

Machines?????

- Regular Languages: Deterministic Finite Automata
- Context Free Languages: Nondeterministic Push-Down Automata
- Context Sensitive Languages: Linear Bounded Nondeterministic Turing Machine

Machines?????

- Regular Languages: Deterministic Finite Automata
- Context Free Languages: Nondeterministic Push-Down Automata
- Context Sensitive Languages: Linear Bounded Nondeterministic Turing Machine
- Recursively Enumerable Languages: Turing Machine

Machines?????

- Regular Languages: Deterministic Finite Automata
- Context Free Languages: Nondeterministic Push-Down Automata
- Context Sensitive Languages: Linear Bounded Nondeterministic Turing Machine
- Recursively Enumerable Languages: Turing Machine

Context Free Languages (and particularly the subset that can be recognised by deterministic push-down automata) are important since most programming languages are deterministic context free languages.

Deterministic Push-Down Automata (Sketch)

Idea: need some way to remember arbitrary number of things that we have seen, eg $a^n b^n$

Deterministic Push-Down Automata (Sketch)

Idea: need some way to remember arbitrary number of things that we have seen, eg $a^n b^n$

Slightly modified DFA along with a stack which stores pairs of states and symbols.

Deterministic Push-Down Automata (Sketch)

Idea: need some way to remember arbitrary number of things that we have seen, eg $a^n b^n$

Slightly modified DFA along with a stack which stores pairs of states and symbols.

DPDA looks at top of stack as well as input to decide what to do

Deterministic Push-Down Automata (Sketch)

Idea: need some way to remember arbitrary number of things that we have seen, eg $a^n b^n$

Slightly modified DFA along with a stack which stores pairs of states and symbols.

DPDA looks at top of stack as well as input to decide what to do

on state transitions, DPDA can pop and/or push things on the stack as well as (perhaps) reading symbol

What about our "questions"?

What about our "questions"?

Given two DPDA, M_1 and M_2, can we determine if $L(M_1) = L(M_2)$?

What about our "questions"?

Given two DPDA, M_1 and M_2, can we determine if $L(M_1) = L(M_2)$?

Yes.

What about our "questions"?

Given two DPDA, M_1 and M_2, can we determine if $L(M_1) = L(M_2)$?

Yes. Proved in 1997.

What about our "questions"?

Given two DPDA, M_1 and M_2, can we determine if $L(M_1) = L(M_2)$?

Yes. Proved in 1997. Earned 2002 Gödel Prize.

What about our "questions"?

Given two DPDA, M_1 and M_2, can we determine if $L(M_1) = L(M_2)$?

Yes. Proved in 1997. Earned 2002 Gödel Prize.

But for NPDA, the question of equivalence is

What about our "questions"?

Given two DPDA, M_1 and M_2, can we determine if $L(M_1) = L(M_2)$?

Yes. Proved in 1997. Earned 2002 Gödel Prize.

But for NPDA, the question of equivalence is undecidable