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, \( w = u_1vu_2 \), where \( u_1, v \) and \( u_2 \) satisfy:

- \( |v| \geq 1 \) (i.e. \( v \neq \varepsilon \))
- \( |u_1v| \leq \ell \)
- for all \( n \geq 0 \), \( u_1v^n u_2 \in L \)
  
(i.e. \( u_1u_2 \in L \), \( u_1vu_2 \in L \) [but we knew that anyway], \( u_1vvu_2 \in L \), \( u_1vuu_2 \in L \), etc.)

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

$$s = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots \xrightarrow{a_\ell} q_\ell \cdots \xrightarrow{a_n} q_n \in F$$

$q_0, \ldots, q_\ell$ 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

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

where

$$u_1 \triangleq a_1 \cdots a_i \quad v \triangleq a_{i+1} \cdots a_j \quad u_2 \triangleq a_{j+1} \cdots a_n$$
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 no matter how $w$ is split into three, $w = u_1v_u_2$, with $|u_1v| \leq \ell$ and $|v| \geq 1$, there is some $n \geq 0$ for which $u_1v^nu_2$ is not in $L$
Examples

None of the following three languages are regular:

(i) \( L_1 \triangleq \{ a^n b^n \mid n \geq 0 \} \)
For each \( \ell \geq 1 \), take \( w = a^\ell b^\ell \in L_1 \)

If \( w = u_1vuu_2 \) with \( |u_1v| \leq \ell \neq |v| \geq 1 \), then for some \( r \) and \( s \):

- \( u_1 = a^r \)
\[ L_1 = \{ a^n b^n \mid n \geq 0 \} \]

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

If \( w = u_1 vu_2 \) with \( |u_1v| \leq \ell \neq |v| \geq 1 \), then for some \( r \) and \( s \):

- \( u_1 = a^r \)
- \( v = a^s \), with \( r + s \leq \ell \) and \( s \geq 1 \)
For each $\ell \geq 1$, take $w = a^\ell b^\ell \in L_1$

If $w = u_1v u_2$ with $|u_1 v| \leq \ell \neq |v| \geq 1$, then for some $r$ and $s$:

- $u_1 = a^r$
- $v = a^s$, with $r + s \leq \ell$ and $s \geq 1$
- $u_2 = a^{l-r-s} b^\ell$
$L_1 = \{a^n b^n \mid n \geq 0\}$

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

If $w = u_1 v u_2$ with $|u_1 v| \leq \ell \neq |v| \geq 1$, 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 $u_1 v^0 u_2 =$
\[ L_1 = \{ a^n b^n \mid n \geq 0 \} \]

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

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

\begin{itemize}
  \item \( u_1 = a^r \)
  \item \( v = a^s \), with \( r + s \leq \ell \) and \( s \geq 1 \)
  \item \( u_2 = a^{\ell - r - s} b^\ell \)
\end{itemize}

so \( u_1 v^0 u_2 = a^r \in a^{\ell - r - s} b^\ell = \)
For each \( \ell \geq 1 \), take \( w = a^\ell b^\ell \in L_1 \)

If \( w = u_1vu_2 \) with \( |u_1v| \leq \ell \neq |v| \geq 1 \), then for some \( r \) and \( s \):

\[
\begin{align*}
\triangleright & \quad u_1 = a^r \\
\triangleright & \quad v = a^s, \quad \text{with } r + s \leq \ell \text{ and } s \geq 1 \\
\triangleright & \quad u_2 = a^{\ell - r - s} b^\ell 
\end{align*}
\]

So \( u_1v^0u_2 = a^r \in a^{\ell - r - s} b^\ell = a^{\ell - s} b^\ell \)
For each \( \ell \geq 1 \), take \( w = a^\ell b^\ell \in L_1 \)

If \( w = u_1v\bar{u}_2 \) with \( |u_1v| \leq \ell \neq |v| \geq 1 \), then for some \( r \) and \( s \):

- \( u_1 = a^r \)
- \( v = a^s \), with \( r + s \leq \ell \) and \( s \geq 1 \)
- \( \bar{u}_2 = a^{\ell-r-s} b^\ell \)

so \( u_1v^0\bar{u}_2 = a^r \in a^{\ell-r-s} b^\ell = a^{\ell-s} b^\ell \)

But \( a^{\ell-s} b^\ell \not\in L_1 \)
For each $\ell \geq 1$, take $w = a^\ell b^\ell \in L_1$

If $w = u_1vu_2$ with $|u_1v| \leq \ell \neq |v| \geq 1$, then for some $r$ and $s$:

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

So $u_1v^0u_2 = a^r \in a^{\ell-r-s}b^\ell = a^{\ell-s}b^\ell$

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

(iii) \( L_3 \triangleq \{ a^p \mid p \text{ prime} \} \)
For each \( \ell \geq 1 \) let \( w = a^p \in L_3, \ p \ prime \iff p > 2\ell \)

If \( w = u_1vu_2 \) with the usual ...
\[ L_3 = \{ a^p \mid p \text{ prime} \} \]

For each \( \ell \geq 1 \) let \( w = a^p \in L_3, p \text{ prime} \iff p > 2\ell \)

If \( w = u_1 vu_2 \) with the usual ...

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

with \( s \geq 1 \iff r + s \leq \ell \)
For each \( \ell \geq 1 \) let \( w = a^p \in L_3, \text{ } p \text{ prime } \iff p > 2\ell \)

If \( w = u_1vu_2 \) with the usual …

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

with \( s \geq 1 \iff r + s \leq \ell \)

so \( u_1v a^{p-s}u_2 = \)
\[ L_3 = \{ a^p \mid p \text{ prime} \} \]

For each \( \ell \geq 1 \) let \( w = a^p \in L_3, \; p \text{ prime} \iff p > 2\ell \)

If \( w = u_1v u_2 \) with the usual ...

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

with \( s \geq 1 \iff r + s \leq \ell \)

so \( u_1v^{p-s}u_2 = a^r a^{s(p-s)} a^{p-r-s} = \)
For each $\ell \geq 1$ let $w = a^p \in L_3$, $p$ prime $\iff p > 2\ell$

If $w = u_1vu_2$ with the usual ...

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

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

so $u_1v^{p-s}u_2 = a^r a^{s(p-s)} a^{p-r-s} = a^{(p-s)(s+1)}$
For each \( \ell \geq 1 \) let \( w = a^p \in L_3, \ p \ \text{prime} \neq p > 2\ell \)

If \( w = u_1v u_2 \) with the usual ...

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

with \( s \geq 1 \neq r+s \leq \ell \)

so \( u_1v^{p-s}u_2 = a^r a^s(p-s) a^{p-r-s} = a^{(p-s)(s+1)} \)

But \( s \geq 1 \Rightarrow s + 1 \geq 2 \)

and \( (p - s) > (2\ell - \ell) \geq 1 \Rightarrow (p - s) \geq 2 \)
$L_3 = \{ a^p \mid p \text{ prime} \}$

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

If $w = u_1vu_2$ with the usual ...

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

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

so $u_1v^{p-s}u_2 = a^r \; a^s(p-s) \; a^{p-r-s} = a^{(p-s)(s+1)}$

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) \[ L_2 \triangleq \{ w \in \{a, b\}^* \mid w \text{ a palindrome} \} \]

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

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

[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 \( \geq 1 \), 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)
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 it mean for a function to be computable?

[ lb Computation Theory ]
The way ahead, in THEORY

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

- Are some computational tasks intrinsically unfeasible?
  [IB Complexity Theory]
The way ahead, in THEORY

- What does it mean for a function to be computable?  
  [ Ib Computation Theory ]

- Are some computational tasks intrinsically unfeasible?  
  [ Ib Complexity Theory ]

- 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
  ⊂ Context Free Languages
  ⊂ Context Sensitive Languages
  ⊂ 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

\[ G \rightarrow E \quad \Delta_0 \]
\[ E \rightarrow E + T \quad \Delta_1 \]
\[ E \rightarrow T \quad \Delta_2 \]
\[ T \rightarrow T \ast P \quad \Delta_3 \]
\[ T \rightarrow P \quad \Delta_4 \]
\[ P \rightarrow (E) \quad \Delta_5 \]
\[ P \rightarrow x \quad \Delta_6 \]
Everybody's favourite grammar

\[
\begin{align*}
G & \rightarrow E \quad \Delta_0 \\
E & \rightarrow E + T \quad \Delta_1 \\
E & \rightarrow T \quad \Delta_2 \\
T & \rightarrow T \ast P \quad \Delta_3 \\
T & \rightarrow P \quad \Delta_4 \\
P & \rightarrow (E) \quad \Delta_5 \\
P & \rightarrow x \quad \Delta_6
\end{align*}
\]

so, e.g. \( G \xrightarrow{\Delta_0} E \)
Everybody’s favourite grammar

\[ G \rightarrow E \quad \Delta_0 \]
\[ E \rightarrow E + T \quad \Delta_1 \]
\[ E \rightarrow T \quad \Delta_2 \]
\[ T \rightarrow T \ast P \quad \Delta_3 \]
\[ T \rightarrow P \quad \Delta_4 \]
\[ P \rightarrow (E) \quad \Delta_5 \]
\[ P \rightarrow x \quad \Delta_6 \]

so, e.g. \[ G \xrightarrow{\Delta_0} E \xrightarrow{\Delta_1} E + T \]
Everybody's favourite grammar

\[ G \rightarrow E \quad \Delta_0 \]
\[ E \rightarrow E + T \quad \Delta_1 \]
\[ E \rightarrow T \quad \Delta_2 \]
\[ T \rightarrow T * P \quad \Delta_3 \]
\[ T \rightarrow P \quad \Delta_4 \]
\[ P \rightarrow (E) \quad \Delta_5 \]
\[ P \rightarrow x \quad \Delta_6 \]

so, e.g. \[ G \xrightarrow{\Delta_0} E \xrightarrow{\Delta_1} E + T \xrightarrow{\Delta_4} E + P \]
Everybody's favourite grammar

\[ G \rightarrow E \quad \Delta_0 \]
\[ E \rightarrow E + T \quad \Delta_1 \]
\[ E \rightarrow T \quad \Delta_2 \]
\[ T \rightarrow T \ast P \quad \Delta_3 \]
\[ T \rightarrow P \quad \Delta_4 \]
\[ P \rightarrow (E) \quad \Delta_5 \]
\[ P \rightarrow x \quad \Delta_6 \]

so, e.g. \[ G \stackrel{\Delta_0}{\rightarrow} E \stackrel{\Delta_1}{\rightarrow} E + T \stackrel{\Delta_4}{\rightarrow} E + P \stackrel{\Delta_6}{\rightarrow} E + x \]
Everybody’s favourite grammar

\[
\begin{align*}
G &\rightarrow E \quad \Delta_0 \\
E &\rightarrow E + T \quad \Delta_1 \\
E &\rightarrow T \quad \Delta_2 \\
T &\rightarrow T \ast P \quad \Delta_3 \\
T &\rightarrow P \quad \Delta_4 \\
P &\rightarrow (E) \quad \Delta_5 \\
P &\rightarrow x \quad \Delta_6 \\
\end{align*}
\]

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

\[ G \rightarrow E \quad \Delta_0 \]
\[ E \rightarrow E + T \quad \Delta_1 \]
\[ E \rightarrow T \quad \Delta_2 \]
\[ T \rightarrow T \ast P \quad \Delta_3 \]
\[ T \rightarrow P \quad \Delta_4 \]
\[ P \rightarrow (E) \quad \Delta_5 \]
\[ P \rightarrow x \quad \Delta_6 \]

so, e.g.

\[ G \overset{\Delta_0}{\rightarrow} E \overset{\Delta_1}{\rightarrow} E + T \overset{\Delta_4}{\rightarrow} E + P \overset{\Delta_6}{\rightarrow} E + x \overset{\Delta_2}{\rightarrow} \]
\[ T + x \overset{\Delta_4}{\rightarrow} P + x \]
Everybody’s favourite grammar

\[ G \rightarrow E \quad \Delta_0 \]
\[ E \rightarrow E + T \quad \Delta_1 \]
\[ E \rightarrow T \quad \Delta_2 \]
\[ T \rightarrow T * P \quad \Delta_3 \]
\[ T \rightarrow P \quad \Delta_4 \]
\[ P \rightarrow (E) \quad \Delta_5 \]
\[ P \rightarrow x \quad \Delta_6 \]

so, e.g. \[ G \xrightarrow{\Delta_0} E \xrightarrow{\Delta_1} E + T \xrightarrow{\Delta_4} E + P \xrightarrow{\Delta_6} E + x \xrightarrow{\Delta_2} \]
\[ T + x \xrightarrow{\Delta_4} P + x \xrightarrow{\Delta_5} (E) + x \]
Everybody's favourite grammar

\[ G \to E \quad \Delta_0 \]
\[ E \to E + T \quad \Delta_1 \]
\[ E \to T \quad \Delta_2 \]
\[ T \to T \ast P \quad \Delta_3 \]
\[ T \to P \quad \Delta_4 \]
\[ P \to (E) \quad \Delta_5 \]
\[ P \to x \quad \Delta_6 \]

so, e.g.

\[ G \xrightarrow{\Delta_0} E \xrightarrow{\Delta_1} E + T \xrightarrow{\Delta_4} E + P \xrightarrow{\Delta_6} E + x \xrightarrow{\Delta_2} \]
\[ T + x \xrightarrow{\Delta_4} P + x \xrightarrow{\Delta_5} (E) + x \xrightarrow{\ldots} (x + x) + x \]
Everybody’s favourite grammar

\[
G \rightarrow E \quad \Delta_0 \\
E \rightarrow E + T \quad \Delta_1 \\
E \rightarrow T \quad \Delta_2 \\
T \rightarrow T \ast P \quad \Delta_3 \\
T \rightarrow P \quad \Delta_4 \\
P \rightarrow (E) \quad \Delta_5 \\
P \rightarrow x \quad \Delta_6
\]

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

is a derivation of \((x + x) + x\)
Language classes by forms of production

\( \alpha, \beta, \gamma \) any strings of terminals and non-terminals
Language classes by forms of production

\[ \alpha, \beta, \gamma \] any strings of terminals and non-terminals

Context Free Languages:
productions of form \( N \rightarrow \beta \)
Language classes by forms of production

\( \alpha, \beta, \gamma \) any strings of terminals and non-terminals

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

Context Sensitive Languages: productions of the form \( \alpha N \beta \rightarrow \alpha \gamma \beta \) [Type 1]
Language classes by forms of production

\[ \alpha, \beta, \gamma \text{ any strings of terminals and non-terminals} \]

Context Free Languages: \[ \text{[ Type 2]} \]
productions of form \[ N \rightarrow \beta \]

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

Recursively Enumerable Languages: \[ \text{[ Type 0]} \]
productions of the form \[ \alpha \rightarrow \beta \]
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 $A \rightarrow a$ or $S \rightarrow \varepsilon$ or $A \rightarrow aB$ (right regular)
Language classes by forms of production

How about Regular Languages?  

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

production of the form $A \rightarrow a$ or $S \rightarrow \epsilon$ or $A \rightarrow aB$ (right regular)

or of the form $A \rightarrow a$ or $S \rightarrow \epsilon$ or $A \rightarrow Ba$ (left regular)
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 $A \rightarrow a$ or $S \rightarrow \varepsilon$ or $A \rightarrow aB$ (right regular)

or of the form $A \rightarrow a$ or $S \rightarrow \varepsilon$ or $A \rightarrow Ba$ (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 $L(M_1) = L(M_2)$?

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 $L(M_1) = L(M_2)$?


But for NPDA, the question of equivalence is
What about our "questions"?

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


But for NPDA, the question of equivalence is undecidable.