Langages formels, calculabilité, complexité - L3S1

Regular expressions

`[A-Z][a-z]*[ ][A-Z][A-Z]` is an example of regexp.

Formal definition

1. $\varepsilon, \emptyset$ are regular expressions.
2. $\forall x \in \Sigma$, x is a regexp.
3. If R_1 and R_2 are two regexps, $R_1 + R_2$, $R_1 R_2$, R_1^* are regexps.

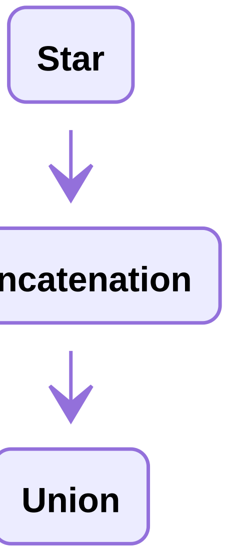
Language of a regexp

- $L(a) = \{a\}$, for $a \in \Sigma$
- $L(\varepsilon) = \{\varepsilon\}$
- $L(\emptyset) = \emptyset$
- $L(R_1 + R_2) = L(R_1) \cup L(R_2)$
- $L(R_1 R_2) = \{xy, (x, y) \in R_1 \times R_2\}$
- $L(R_1^*) = L(R_1)^*$

Examples

- $\{0 + 1\}^* 1$: any string that ends with a 1.
- Strings with alternating 0 and 1 : $(\varepsilon + 1)(01)^*(\varepsilon + 0)$

Order of operations



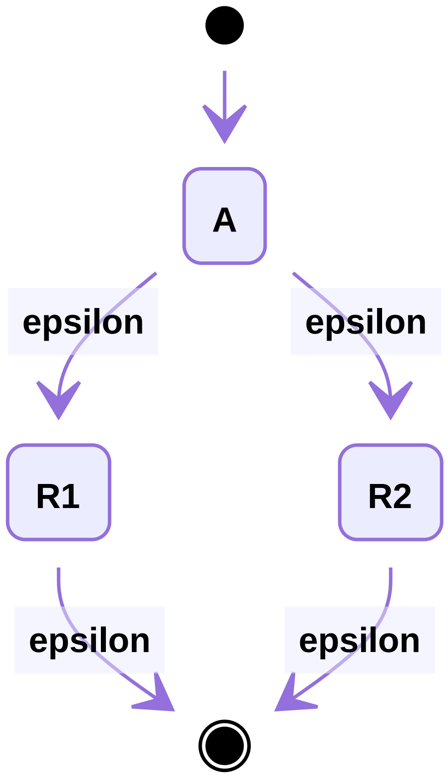
Equivalence with DFAs

Theorem : $L = L(R)$ for some regexp $R \Leftrightarrow L = L(D)$ for some DFA D.

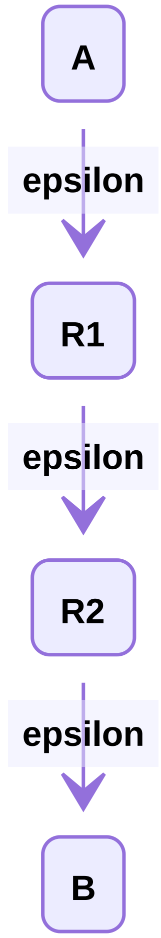
Dem :

$L = L(R) \Rightarrow L = L(D)$

Idea: do it recursively.
It is trivial to do the base cases.
Union : with ε-transitions.



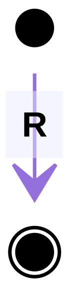
Concatenation : with ε-transitions.



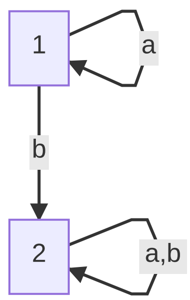
Converting a DFA in a regexp :

1. Assume only one q_{start} and one q_{accept}, $q_{start} \neq q_{accept}$.
2. No incoming transition to q_{start}

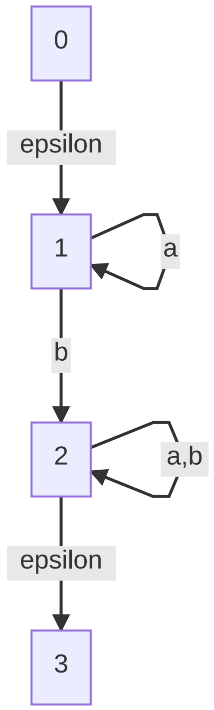
3. No outcoming transition from q_{accept}
4. Initially, k states, other than q_{start} and q_{accept}.
5. For $i = 1, \ldots, k$, remove node i
6. In the end :



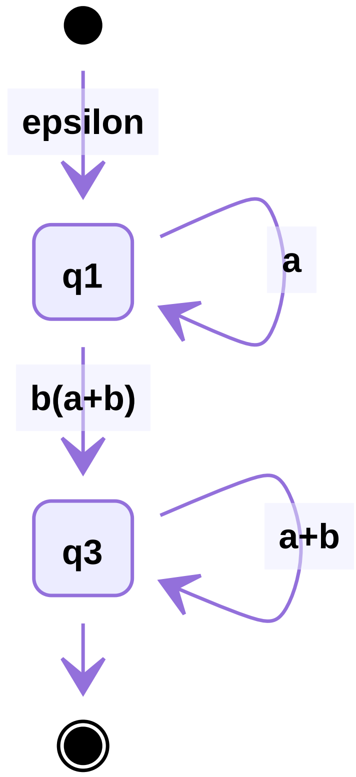
Example :



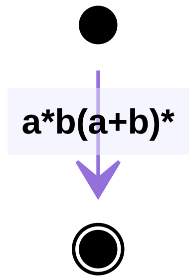
Then



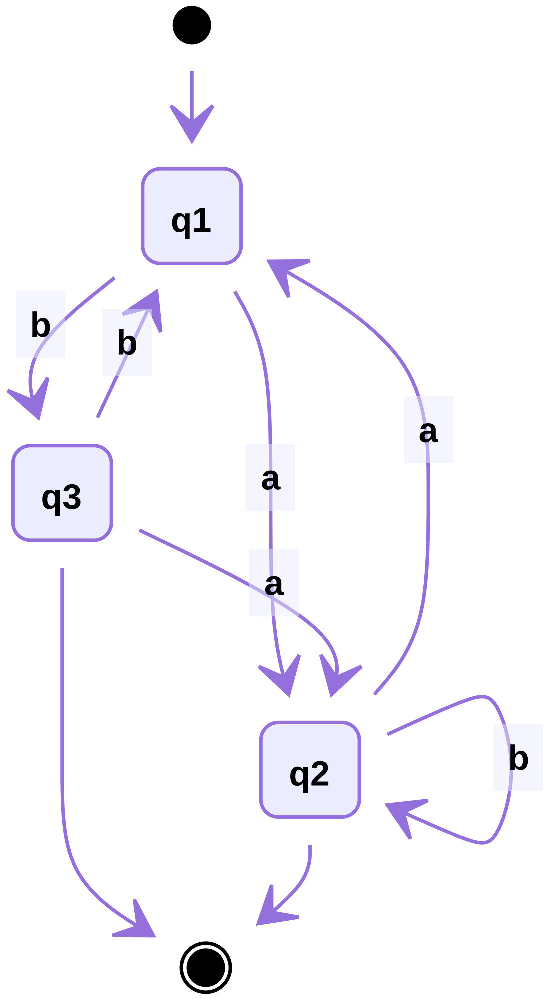
Then : (rip 2)



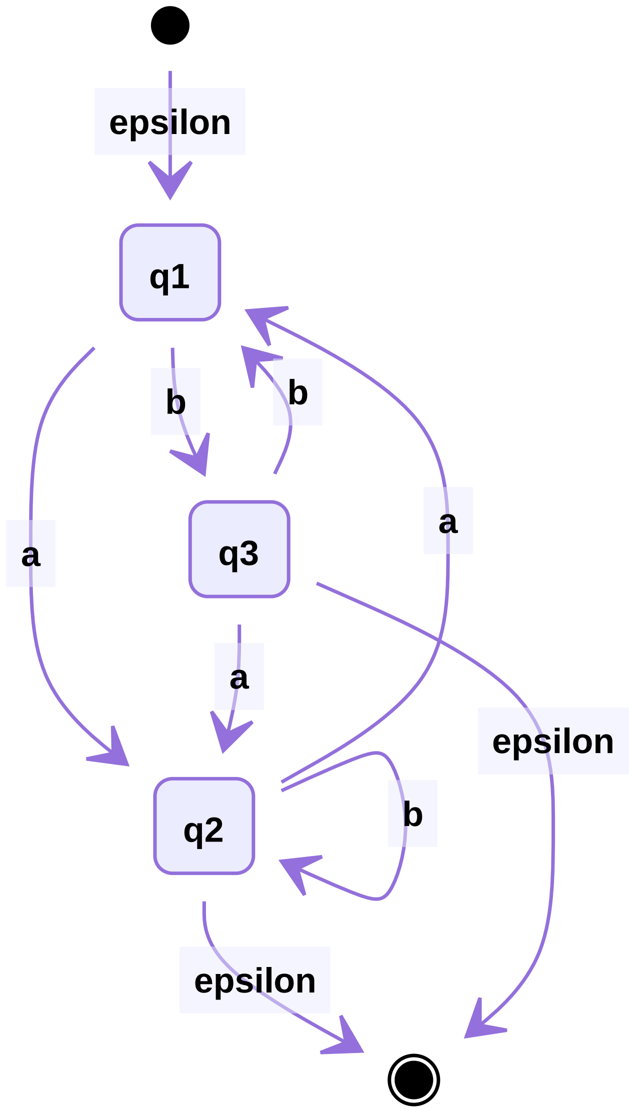
Finally, (rip 1)



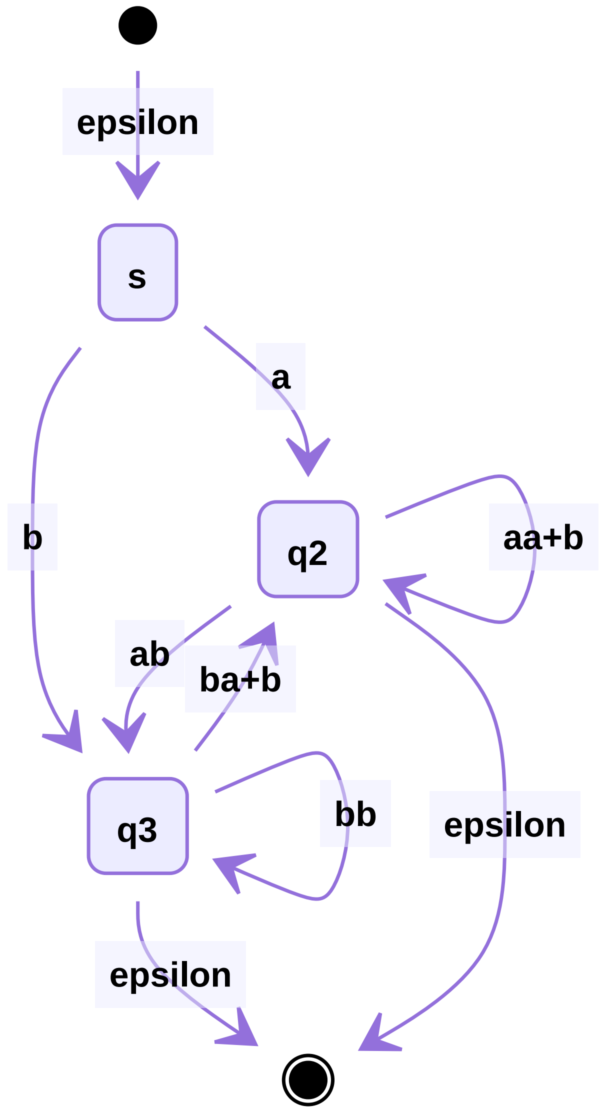
Another example : (accepting : 2 and 3)



Then :



Rip 1 :



And so on and so forth...

Proprieties of regexps

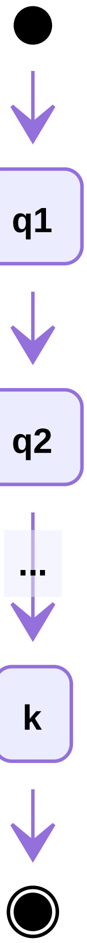
$L+M=M+L,\ (L+M)+N=L+(M+N),\ (LM)N=L(MN),\ \emptyset+L=L+\emptyset=L,\ \varepsilon L=L\varepsilon=L,\ \emptyset L=L\emptyset=\emptyset,$ $\emptyset^*=\varepsilon$, $\varepsilon^*=\varepsilon$, $L(M+N)=LM+LN$, $(M+N)L=ML+NL$, $L+L=L$, $(L^*))^*=L^*$, $L^+=LL^*=L^*L$.

Pumping lemma

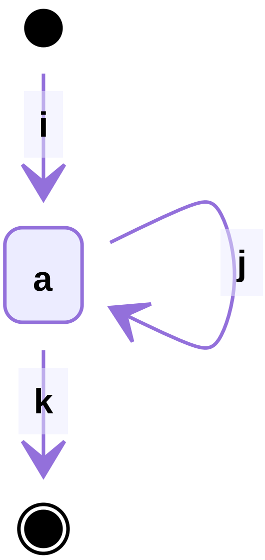
Is $L_{01}=\{0^n1^n, n\in\mathbb{N}\}$ regular ?

Suppose it is. It exists a DFA that recognizes it with a finite number of states k such that $L(D)=L_{01}$.

What about 0^k1^k ?



After reading $k-1$ elements, we have exhausted the different states. After the k^{th} element, a state will be repeated. Therefore, if we read k 0s, the repeated state, a, will be reached.



We then know that $j \neq 0$. Then if $0^k 1^k$ is accepted, then $0^{k-j}1^k$ should also be accepted. Contradiction.

Pumping lemma : let $L = L(D)$ for a DFA D. Then $\exists p \in \mathbb{N}$ (the pumping length) such that $\forall w \in L$ with $|w| \geq p$, we can break w into $w = xyz$ such that :

1. $y \neq \varepsilon$ (non trivial)
2. $|xy| \leq p$ (xy "near the begining")
3. $\forall i \geq 0, \quad xy^i z \in L$ (w can be "pumped")

Proof:

Let $p = |Q|$, $D = (Q, \Sigma, \delta, q_0, F)$. Let $w = a_1 \dots a_m$ with $m \geq p$ (if no such w exists, the proof is done).

Let $q_i := \hat{\delta}(q_0, a_1 \dots a_i) \quad i = 0, \dots, m$ (at least $p + 1$ of these).

By pigeonhole : $\exists i \neq j \leq p$ such that $q_i = q_j$.

Let $x := a_1 \dots a_i$; $y = a_{i+1} \dots a_j$; $z = a_{j+1} \dots a_m, \Rightarrow |xy| \leq p$.

x takes us from q_0 to q_i, y from q_i tp $q_j = q_i$, z from q_j to q_m, an accept state. (x or z may be ε, but $y \neq \varepsilon$ since $i < j$). Hence $xz \in L$ and $xy^k z \in L$ for any $k > 0$.

Examples

$L_{01} = \{0^n 1^n, n \geq 0\}$ is not regular. Suppose it is. Let p be the pumping length. Let $w = 0^p 1^p$. Not $|w| \geq p$.

Then $\exists x, y, z$ such that $w = xyz$, $|xy| \leq p$, $y \neq \varepsilon$ and $xy^i z \in L$ for $i \geq 0$

Since $|xy| \leq p$, y only has 0s. All the ones are in z. But $xz \in L$ by pumping lemma. Absurd.

Let $L = \{1^{n^2}, n \geq 0\}$.

Let $s := 1^{p^2}$. $y = 1^i$, with $0 < i \leq p$.

Then $\forall i \in \mathbb{N}, p^2 + i$ is a perfect square. Absurd.

Minimization

Given a DFA D, does D have redundant states ? How can we find a minimal DFA that is equivalent to D ?

Definition: we say that two states p, q are equivalent iff for all $w \in \Sigma^*$, $\hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F$. We write : $p \equiv q$.

Note: $\equiv$ is an equivalence relation : it partitions the states into equivalence classes.

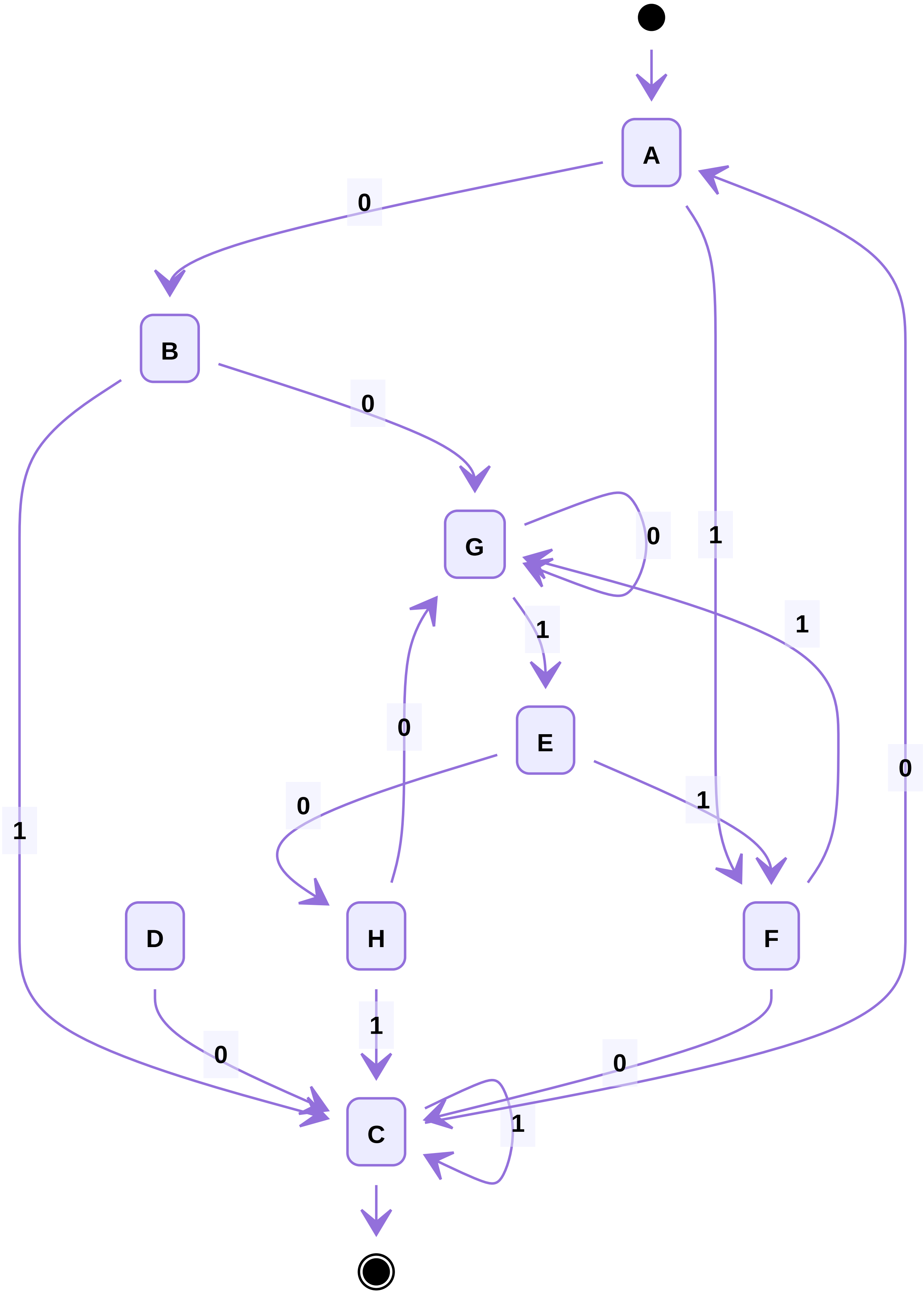
Note: if $p \equiv q$, $p \in F \Leftrightarrow q \in F$.

The equivalent states can be "merged into one".

Table filling algorithm

Find all distinguishable pairs : $p \not\equiv q \Rightarrow \exists w \in \Sigma^* \hat{\delta}(p, w) \in F \wedge \hat{\delta}(q, w) \notin F$, remove them recursively, conclude.

Idea: if for all $a \in \Sigma$, $\delta(r, a) = p, \delta(s, a) = q \ \wedge p \not\equiv q \Rightarrow r \not\equiv s$



Distinguishable pairs table (partly filled)

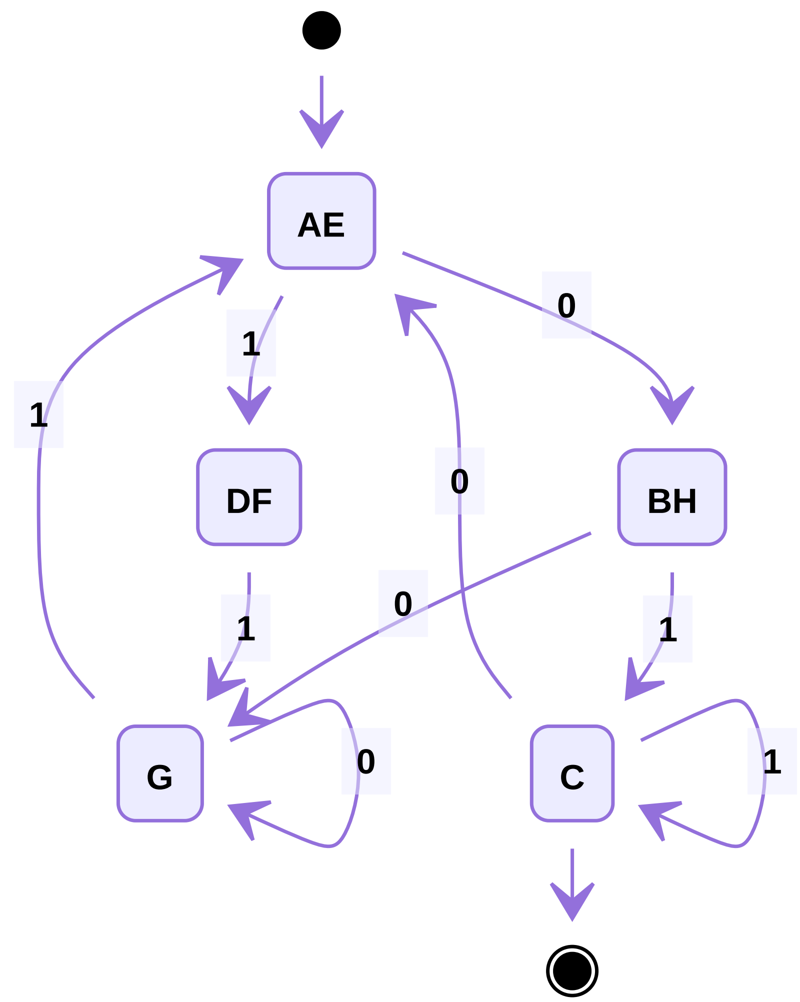
x = distinct	A	B	C	D	E	F	G
A							

x = distinct	A	B	C	D	E	F	G
B							
C	x	x					
D			x				
E			x				
F			x		x		
G			x			x	
H			x				

Ex : $\{C, H\}$ pair, + 0 input $\Rightarrow$ $\{G, F\}$ are distinguishable, +1 input : $\{E, F\}$ distinguishable.

We would see that $\{A, E\}, \{D, F\}, \{B, H\}$ are equivalent. By fusing the nodes, we can come with a smaller DFA that recognizes the same language.

Final DFA :



Theorem: if two states are not distinguished by the table filling algorithm, they are equivalent.

Proof: Call $\{p, q\}$ the bad pair. We consider the pair with the shortest $\omega = a_1 \dots a_n$.

$\hat{\delta}(p,\omega) \in F$, $\hat{\delta}(q,\omega) \notin F$. Let $r = \delta(p, a_1)$, $s = \delta(q, a_1)$. Therefore, r and s are distinguishable (otherwise, p and q wouldn't be). This contradicts the hypothesis of shortest ω. So $\{r, s\}$ is not a bad pair, so the algorithm finds this pair, and thus $\{p, q\}$ at the next step.

Context-free grammars

$L = \{0^N 1^N, N \geq 0\}$ is not a regular language. But it can be described recursively : $\varepsilon \in L$ and $\forall A \in L, 0A1 \in L$.

We note $\begin{matrix} A \to \varepsilon \\ A \to 0A1 \end{matrix}$, or $A \to 0A1|\varepsilon$. Another example is : $\begin{matrix} A \to a \\ A \to A + A | A * A \end{matrix}$.This defines a **context-free grammar**.

Formally, a **context-free grammar (CFG)** is a 4-tu~~ple~~ $\cancel{G = (V, \Sigma, R, S}$), with V the set of variables, Σ the set of terminals (alphabet), R the set of rules/deductions, which are $A \to \omega$, with $\omega \in \{V \cup \Sigma\}^*$, $A \in V$ and S the start variable.

Definition : language recognized by a CFG

Let $A \to \omega$ be a rule, uAv yields $u\omega v$. More generally, u derives v ($u \overset{*}{\to} v$) if $u = v$ or $u \to u_1 \to \cdots \to u_k \to v$.

We define $L(G) = \{\omega \in \Sigma^*, s \overset{*}{\to} \omega\}$ the language recognized by the CFG G.

Example : Let $D = (Q, \Sigma, \delta, q_0, F)$ be a DFA. Then there exists a CFG G that recognizes $L(D)$.

If the transition function is of the form $\delta(q_i, a) = q_j$, we put $R_i \to aR_j$, with R_0 the start variable.

$R_0 \to aR_1 \to abR_2 \to ab$
$R_1 \to \varepsilon$ if $q_i \in F$

This CFG recognizes the same language as the automaton.

Therefore, we've proven that CFGs recognize strictly more languages than DFAs.

Derivation, parse trees

Leftmost derivations

Idea : in each step, we will replace the *leftmost* variable.

Example : $S \to SS \mid (S) \mid \varepsilon$
For instance, we want to generate `(())()` :

$S \to SS \to (S)S \to ((S))S \to ((\varepsilon))S \to (())(S) \to (())(\varepsilon) = (())()$.

Parse trees

```
      S
    /   \
   S     S
  /|\   /|\
 ( S )( S )
   |     |
 ( S )   ε
   |
   ε
```

Sometimes, there is not uniqueness of the parse tree. We call such grammars **ambiguous** (ex : $S \to S + S \mid S * S \mid a$)

Ambiguity

Definition: A CFG is ambiguous if $\exists \omega \in L(G)$ with two or more parse trees.

We can remove ambiguity : $\begin{matrix} B \to (RB \mid \varepsilon \\ R \to) \mid (RR \end{matrix}$ recognizes the same language of parenthesis as before, but is unambiguous.

We say that L is **inherently ambiguous** if there is no unambiguous G such that $L(G) = L$.

An example of such a language would be : $L = \{0^i 1^j 2^p | i = j \vee j = k\}$. The ambiguity comes from $i = j = k$ (which is not context-free, we'll see that later on).

1. For every parse tree, there exists a leftmost derivation
2. For every leftmost derivation, there exists a parse tree

Idea to prove 1. : (induction on the height of the tree)
Base case :

```
      A
    / | \
   a1 .. an
```

There is a rule $A \to a1 \ldots a_n$.

```
       A
     / | \
    /  |  \
   X1 ... Xn
   ~       ~
```

(with ~ a subtree) : $A \to X_1 \dots X_n$, and X_i and be derived in a leftmost fashion $X_i \xrightarrow{*} \omega_i$. By the induction hypothesis, we conclude.

Idea to prove 2. : induction on the number of steps of the derivation

Chomsky normal form (CNF)

A few rules are allowed:

$S \to \varepsilon$
$A \to BC$ (A, B, C are variables)
$A \to a$ where a is a terminal

$\omega = a_1 \dots a_N$ in $2N-1$ steps : $N-1$ steps to extend to N variables, then N steps to simplify the variables into terminals.

Theorem: context-free language (CNL) can be generated by a a grammar G in CNF.

Idea: convert a CFG G into normal form.

1. Add new S_0 : $S_0 \to S$
2. Remove ε-rules : assume $A \to \varepsilon$, $R \to uAv$. We add : $R \to uv$ (case $A \to \varepsilon$), we need to account all occurrences of A if $R \to uAvAw$. If $R \to A$, we add $R \to \varepsilon$ if $R \to \varepsilon$ have not been remove before.
3. Remove unit rules $A \to B$. For each $B \to u$, we add $A \to u$, if we had not deleted such a rule before, as previously.
4. We end up with $A \to u_1 \dots u_n$ with $u_i \in \Sigma \cup V$ or $S_0 \to \varepsilon$.
   $A \to u_1 \dots u_n$ goes to $A \to u_1 A_1$, $A_1 \to u_2 A_2$ ... $A_{n-2} \to u_{n-1} u_n$. We eventually need to change rules if $u_i \in \Sigma$: $A_i \to U_i A_{i+1}$ and $U_i \to u_i$.

Pushdown automata

Definition

Definition : a PDA is a 6-tuple $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$, with :

- Q the set of states
- Σ the input alphabet
- Γ the tape alphabet
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ the transition function
- q_0 the initial state
- F the set of accepting states

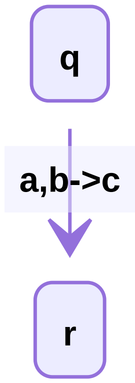
A PDA accepts $w = w_1 \dots w_M$ with $w_i \in \Sigma_\varepsilon$ if there are $r_0, \dots r_m \in Q$, $s_0, \dots, s_M \in \Gamma^*$ such that :

1. $r_0 = q_0$ and $s_0 = \varepsilon$

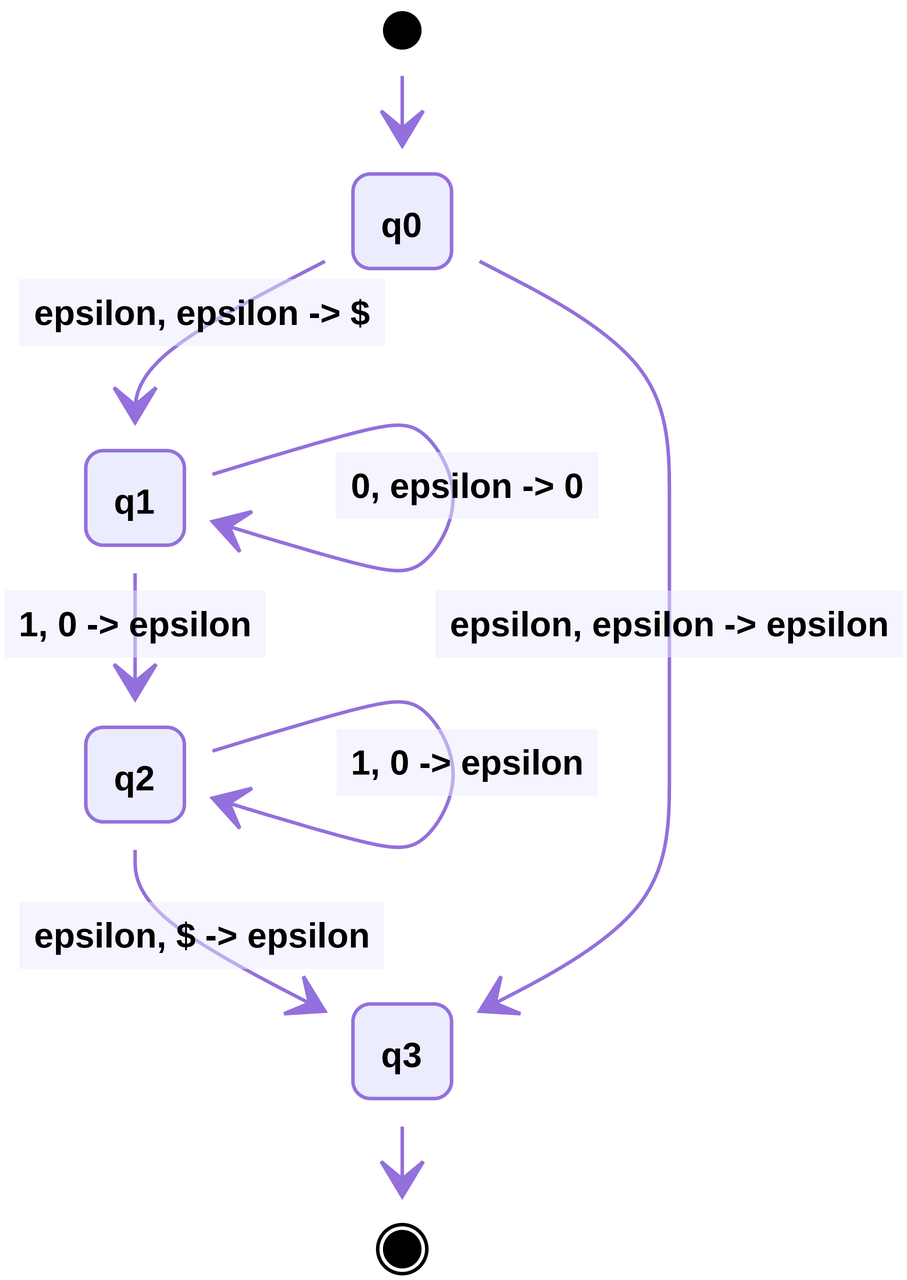
2. For $i = 0$ to $M - 1$:
   $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$
   $s_i = at, s_{i+1} = bt, \qquad a, b \in \Gamma_\varepsilon, t \in \Gamma^*$
3. $r_M \in F(, s_M = \varepsilon)$

Remark: this is not a deterministic PDA. We could define a deterministic PDA, but it will only recognize unambiguous languages.

We will draw :



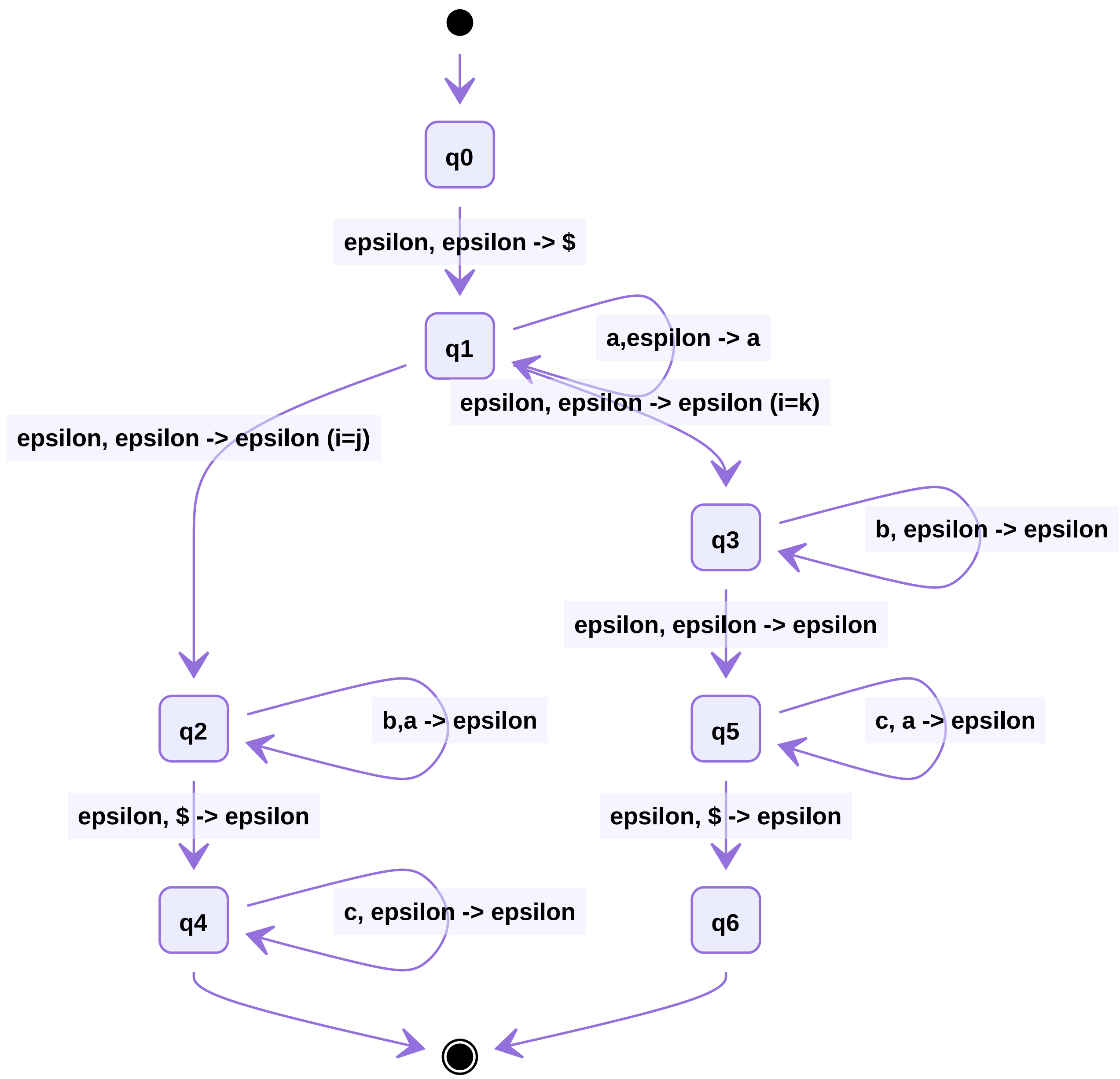
Meaning that we read a, and we replace b with c at the top of the stack ($\delta(q, a, b) \ni (r, c)$).



	0	1	ε
q_0			
q_1			
q_2			
q_3			

We can also put this information in a table like above, but this is harder to visualize.

Example : $L = \{a^i b^j c^k, i = j \text{ or } i = k\}$



Equivalence between PDA and CFG

From CFGs to PDAs

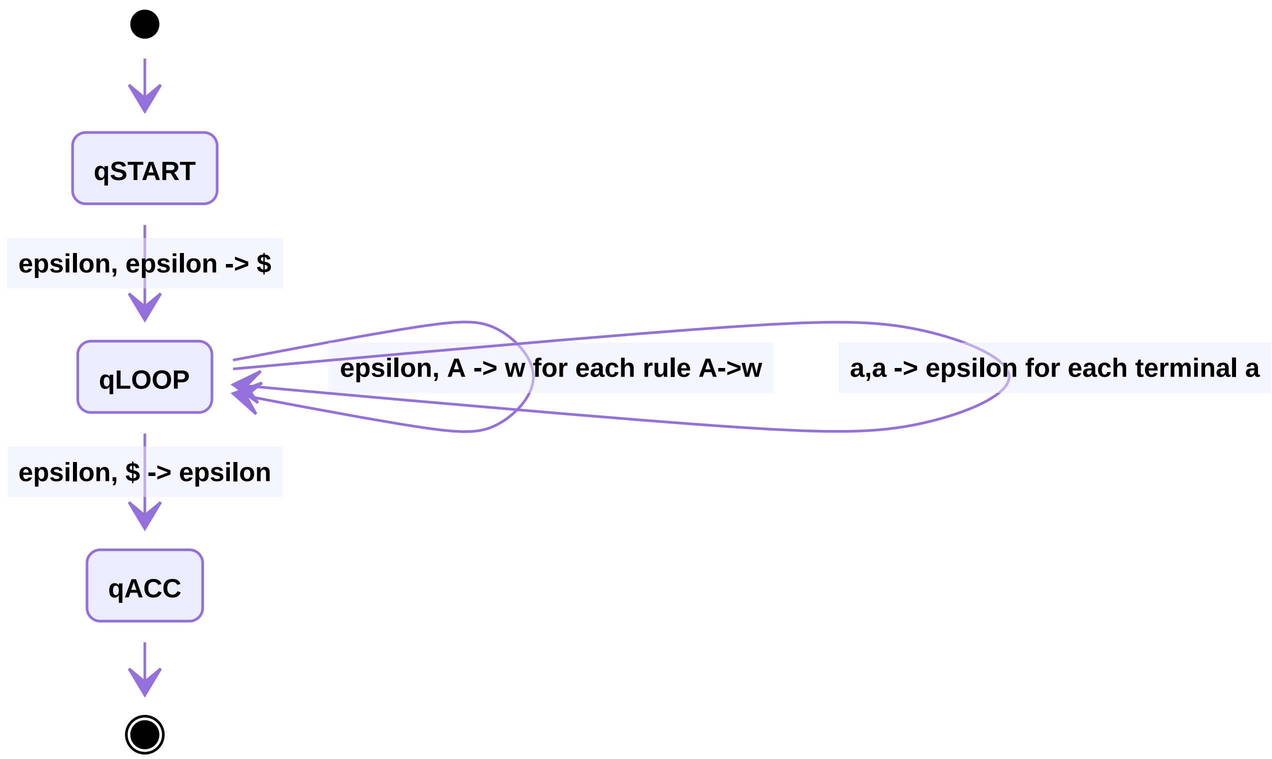
Claim: if L is a CFG, then some PDA recognizes it.

Proof: $\exists$ a CFG G such that $L(G) = L$. We want to convert G into a PDA P.

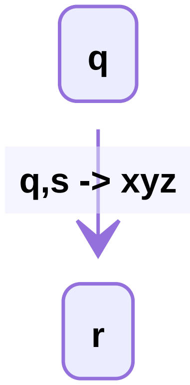
$S \to aTb|b$
$T \to Ta|\varepsilon$ => it converts into the stack as `[S;$]` `->` `[a;T;b;$]`

If we read a a, it would become `[T;b;$]` then `[b;$]`for instance ($T \to \varepsilon$). We then read a b : `[$]`. `$` is on top, so it is a word accepted.

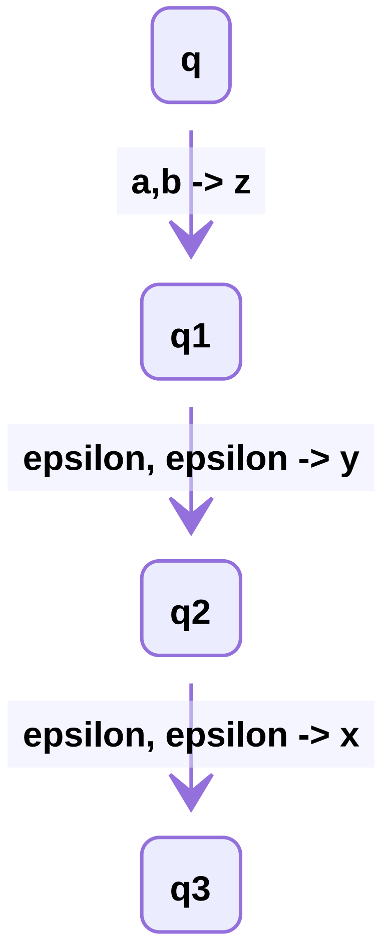
Thus, we can make the PDA like so :



We can then decompose each complex rule by using several states. For instance,



Can be decomposed into :

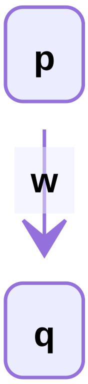


From PDAs to CGAs

Claim: if a PDA recognizes a language L, then there exists a CFG G such that $L(G) = L$.

Proof: given a PDA P, we build a CFG G for $L(P)$.

We want the stack to be empty at the beginning and at the end (or more generally, if the stack is unchanged at the end compared to the beginning).



Goal : create a variable A_{pq} that can take P from p to q on an empty stack. We have $A_{pp} \to \varepsilon$.

Two situations :

1. the stack never gets empty between p and q.
2. the stack gets empty between p and q.

Assumption:

1. there is a single accept state
2. it empties before accepting
3. each transition is either a push or a pop

$$
\begin{array}{lcr}
A_{pq} \to A_{pr}A_{rq} & & \forall p,q,r \in Q \\
A_{pq} \to aA_{rs}b & \text{if} & \begin{array}{l}\delta(p,a,\varepsilon) \ni (r,u) \\ \delta(s,b,u) \ni (q,\varepsilon)\end{array} \\
A_{p,p} \to \varepsilon & & \forall p \in Q
\end{array}
$$

Claim: if A_{pq} generates x then x can bring P from p to q with empty stacks.

Proof: base case : the derivation has 1 step. Then this rule is $A_{pp} \to \varepsilon$.

Induction step : assume this is true for k steps of derivation. Let's show that it is true for $k+1$ steps.

Suppose that A_{pq} generates x in $k+1$ steps.

Case 1: The first step is $\to A_{pq} \to aA_{rs}b$. $x = ayb$, $A_{rs} \xrightarrow{*} y$.

Then A_{rs} can bring P from r to s in k steps, by induction hypothesis.

Then the stack at r and s is `[u]`, so x can bring P from p to q on empty stacks.

Other case : see lecture notes.

Pumping lemma

Lemma: If L is a CFL, then there exists p such that for every $s \in L$, $|s| \geq p$, that can be broken into $s = uvwxyz$ such that

1. for each i : $uv^ixy^iz \in L$
2. $|vy| > 0$
3. $|vxy| \leq p$

Proof: let s be "very long" (in a sense to be defined later on). Then the parse tree must be "very tall".

cf lecture notes

Example: $L = \{a^N, b^N, c^N, N \geq 0\}$.

We assume L is a CFL. Then there exists p such that $a^pb^pc^p \in L$. There exists u, v, x, y, z such that $a^pb^pc^p = uvwxyz$ and $uv^ixy^iz \in L \quad \forall i \geq 0$. We then, by case enumeration, conclude that this is not possible.