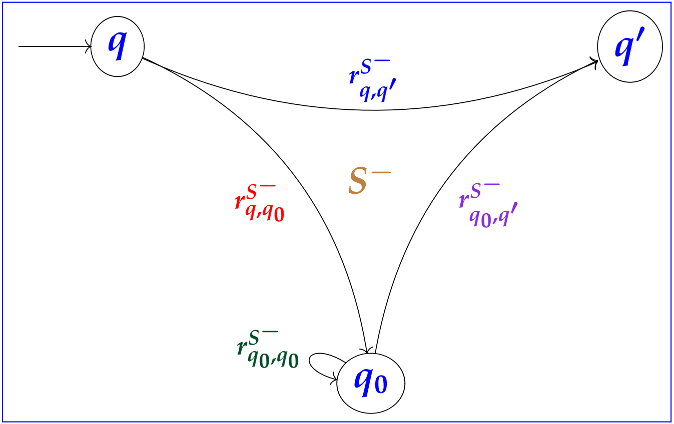
$$r^{S}_{q,q'} = r^{S^-}_{q,q'} | (r^{S^-}_{q,q_0}[r^{S^-}_{q_0,q_0}]^* r^{S^-}_{q_0,q'})$$



An Example

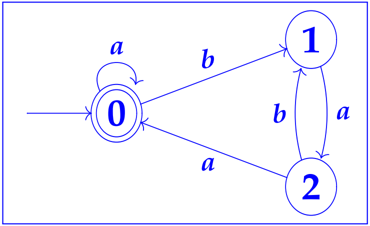
Demonstrates don't always have to follow induction to bitter end (but when in doubt...)

Construction works backwards to the induction; we start with all the states and remove one at a time.

We get to choose the state to remove in each step.

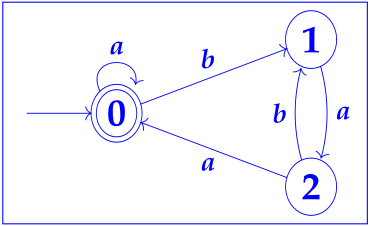
Strategy: choose a state that disconnects the automaton as much as possible

$M \triangleq$



Looking for $r_{0,0}^{\{0,1,2\}}$

$M \triangleq$



Looking for $r_{0,0}^{\{0,1,2\}}$

By direct inspection we have:

$r_{i,j}^{\{0\}}$	0	1	2
0			
1	$\emptyset$	ε	a
2	aa^*	a^*b	ε

$r_{i,j}^{\{0,2\}}$	0	1	2
0	a^*	a^*b	
1			
2			

(we don't need the unfilled entries in the tables)

We want $r_{0,0}^{\{0,1,2\}}$

We want $r_{0,0}^{\{0,1,2\}}$

Remove 1 from {0, 1, 2}

We want $r_{0,0}^{\{0,1,2\}}$

Remove 1 from {0, 1, 2}

$$r_{0,0}^{\{0,1,2\}} \triangleq r_{0,0}^{\{0,2\}} \quad | \quad (r_{0,1}^{\{0,2\}} \; [r_{1,1}^{\{0,2\}}]^* \; r_{1,0}^{\{0,2\}})$$

We want $r_{0,0}^{\{0,1,2\}}$

Remove 1 from {0, 1, 2}

$$r_{0,0}^{\{0,1,2\}} \triangleq \underset{a^*}{r_{0,0}^{\{0,2\}}} \quad | \quad (\underset{a^*b}{r_{0,1}^{\{0,2\}}} \; [r_{1,1}^{\{0,2\}}]^* \; r_{1,0}^{\{0,2\}})$$

We want $r_{0,0}^{\{0,1,2\}}$

Remove 1 from {0, 1, 2}

$$\begin{aligned} r_{0,0}^{\{0,1,2\}} &\triangleq r_{0,0}^{\{0,2\}} \quad | \quad (r_{0,1}^{\{0,2\}} \; [r_{1,1}^{\{0,2\}}]^* \; r_{1,0}^{\{0,2\}}) \\ &= a^* \quad | \quad (a^*b \; [r_{1,1}^{\{0,2\}}]^* \; r_{1,0,}^{\{0,2\}}) \end{aligned}$$

We want $r_{0,0}^{\{0,1,2\}}$

Remove 2 from {0, 2}

$$\begin{aligned} r_{0,0}^{\{0,1,2\}} &\triangleq r_{0,0}^{\{0,2\}} \quad \mid \quad \left(r_{0,1}^{\{0,2\}} \; \left[r_{1,1}^{\{0,2\}}\right]^* \; r_{1,0}^{\{0,2\}}\right) \\ &= a^* \quad \mid \quad \left(a^*b \; \left[r_{1,1}^{\{0,2\}}\right]^* \; r_{1,0,}^{\{0,2\}}\right) \end{aligned}$$

$$r_{1,1}^{\{0,2\}} \triangleq r_{1,1}^{\{0\}} \quad \mid \quad \left(r_{0,2}^{\{0\}} \; \left[r_{2,2}^{\{0\}}\right]^* \; r_{2,1}^{\{0\}}\right)$$

We want $r_{0,0}^{\{0,1,2\}}$

Remove 2 from $\{0, 2\}$

$$
\begin{aligned}
r_{0,0}^{\{0,1,2\}} &\triangleq r_{0,0}^{\{0,2\}} \quad | \quad (r_{0,1}^{\{0,2\}} \; [r_{1,1}^{\{0,2\}}]^* \; r_{1,0}^{\{0,2\}}) \\
&= a^* \quad | \quad (a^*b \; [r_{1,1}^{\{0,2\}}]^* \; r_{1,0,}^{\{0,2\}})
\end{aligned}
$$

$$
\begin{aligned}
r_{1,1}^{\{0,2\}} &\triangleq r_{1,1}^{\{0\}} \quad | \quad (r_{0,2}^{\{0\}} \; [r_{2,2}^{\{0\}}]^* \; r_{2,1}^{\{0\}}) \\
&= \varepsilon \quad | \quad (a \; [\varepsilon]^* \; a^*b)
\end{aligned}
$$

We want $r_{0,0}^{\{0,1,2\}}$

Remove 2 from {0, 2}

$$\begin{aligned} r_{0,0}^{\{0,1,2\}} &\triangleq r_{0,0}^{\{0,2\}} \mid (r_{0,1}^{\{0,2\}} \; [r_{1,1}^{\{0,2\}}]^* \; r_{1,0}^{\{0,2\}}) \\ &= a^* \mid (a^*b \; [r_{1,1}^{\{0,2\}}]^* \; r_{1,0,}^{\{0,2\}}) \end{aligned}$$

$$\begin{aligned} r_{1,1}^{\{0,2\}} &\triangleq r_{1,1}^{\{0\}} \mid (r_{0,2}^{\{0\}} \; [r_{2,2}^{\{0\}}]^* \; r_{2,1}^{\{0\}}) \\ &= \varepsilon \mid (a \; [\varepsilon]^* \; a^*b) \\ &= \varepsilon \mid (a a * b) \end{aligned}$$

We want $r_{0,0}^{\{0,1,2\}}$

Remove 2 from {0, 2}

$$\begin{aligned}
r_{0,0}^{\{0,1,2\}} &\triangleq r_{0,0}^{\{0,2\}} \;\Big|\; \left(r_{0,1}^{\{0,2\}} \left[r_{1,1}^{\{0,2\}}\right]^* r_{1,0}^{\{0,2\}}\right) \\
&= a^* \;\Big|\; \left(a^*b \;\left[\varepsilon \middle| (aa*b)\right]^* r_{1,0}^{\{0,2\}}\right)
\end{aligned}$$

$$\begin{aligned}
r_{1,1}^{\{0,2\}} &\triangleq r_{1,1}^{\{0\}} \;\Big|\; \left(r_{0,2}^{\{0\}} \left[r_{2,2}^{\{0\}}\right]^* r_{2,1}^{\{0\}}\right) \\
&= \varepsilon \;\Big|\; \left(a \;[\varepsilon]^* \; a^*b\right) \\
&= \varepsilon \mid (aa*b)
\end{aligned}$$

We want $r_{0,0}^{\{0,1,2\}}$

Remove 2 from {0, 2}

$$\begin{aligned} r_{0,0}^{\{0,1,2\}} &\triangleq r_{0,0}^{\{0,2\}} \quad | \quad (r_{0,1}^{\{0,2\}} \quad [r_{1,1}^{\{0,2\}}]^* \quad r_{1,0}^{\{0,2\}}) \\ &= a^* \quad | \quad (a^*b \quad [\varepsilon|(aa*b)]^* \quad r_{1,0}^{\{0,2\}}) \end{aligned}$$

We want $r_{0,0}^{\{0,1,2\}}$

Remove 2 from {0, 2}

$$\begin{aligned} r_{0,0}^{\{0,1,2\}} &\triangleq r_{0,0}^{\{0,2\}} \quad\Big|\quad \left(r_{0,1}^{\{0,2\}} \quad \left[r_{1,1}^{\{0,2\}}\right]^* \quad r_{1,0}^{\{0,2\}}\right) \\ &= a^* \quad\Big|\quad \left(a^*b \quad \left[\varepsilon\middle|\left(aa*b\right)\right]^* \quad r_{1,0}^{\{0,2\}}\right) \end{aligned}$$

$$r_{1,0}^{\{0,2\}} \triangleq r_{1,0}^{\{0\}} \quad\Big|\quad \left(r_{1,2}^{\{0\}} \quad \left[r_{2,2}^{\{0\}}\right]^* \quad r_{2,0}^{\{0\}}\right)$$

We want $r_{0,0}^{\{0,1,2\}}$

Remove 2 from {0, 2}

$$\begin{aligned} r_{0,0}^{\{0,1,2\}} &\triangleq r_{0,0}^{\{0,2\}} \mid (r_{0,1}^{\{0,2\}} \, [r_{1,1}^{\{0,2\}}]^* \, r_{1,0}^{\{0,2\}}) \\ &= a^* \mid (a^*b \, [\varepsilon|(aa*b)]^* \, r_{1,0}^{\{0,2\}}) \end{aligned}$$

$$\begin{aligned} r_{1,0}^{\{0,2\}} &\triangleq r_{1,0}^{\{0\}} \mid (r_{1,2}^{\{0\}} \, [r_{2,2}^{\{0\}}]^* \, r_{2,0}^{\{0\}}) \\ &= \emptyset \mid a*(\epsilon)^* \, aa^* \end{aligned}$$

We want $r_{0,0}^{\{0,1,2\}}$

Remove 2 from {0, 2}

$$\begin{aligned}
r_{0,0}^{\{0,1,2\}} &\triangleq r_{0,0}^{\{0,2\}} \mid (r_{0,1}^{\{0,2\}} \; [r_{1,1}^{\{0,2\}}]^* \; r_{1,0}^{\{0,2\}}) \\
&= a^* \mid (a^*b \; [\varepsilon \mid (aa \ast b)]^* \; r_{1,0}^{\{0,2\}})
\end{aligned}$$

$$\begin{aligned}
r_{1,0}^{\{0,2\}} &\triangleq r_{1,0}^{\{0\}} \mid (r_{1,2}^{\{0\}} \; [r_{2,2}^{\{0\}}]^* \; r_{2,0}^{\{0\}}) \\
&= \emptyset \mid a \ast (\epsilon)^* \; aa^* \\
&= aaa^*
\end{aligned}$$

We want $r_{0,0}^{\{0,1,2\}}$

Remove 2 from {0, 2}

$$\begin{aligned}
r_{0,0}^{\{0,1,2\}} &\triangleq r_{0,0}^{\{0,2\}} \quad | \quad (r_{0,1}^{\{0,2\}} \quad [r_{1,1}^{\{0,2\}}]^* \quad r_{1,0}^{\{0,2\}}) \\
&= a^* \quad | \quad (a^* b \quad [\varepsilon|(aa \ast b)]^* \quad aaa^*)
\end{aligned}$$

$$\begin{aligned}
r_{1,0}^{\{0,2\}} &\triangleq r_{1,0}^{\{0\}} \quad | \quad (r_{1,2}^{\{0\}} \quad [r_{2,2}^{\{0\}}]^* \quad r_{2,0}^{\{0\}}) \\
&= \emptyset \quad | \quad a \ast (\epsilon)^* \quad aa^* \\
&= aaa^*
\end{aligned}$$

We want $r_{0,0}^{\{0,1,2\}}$

Remove 2 from $\{0, 2\}$

$$\begin{aligned} r_{0,0}^{\{0,1,2\}} &\triangleq r_{0,0}^{\{0,2\}} \quad \mid \quad \left(r_{0,1}^{\{0,2\}} \quad \left[r_{1,1}^{\{0,2\}}\right]^* \quad r_{1,0}^{\{0,2\}}\right) \\ &= a^* \quad \mid \quad \left(a^*b \quad \left[\varepsilon \middle| \left(a a * b\right)\right]^* \quad a a a^*\right) \end{aligned}$$

We want $r_{0,0}^{\{0,1,2\}}$

Remove 2 from {0, 2}

$$\begin{aligned} r_{0,0}^{\{0,1,2\}} &\triangleq r_{0,0}^{\{0,2\}} \quad | \quad (r_{0,1}^{\{0,2\}} \quad [r_{1,1}^{\{0,2\}}]^* \quad r_{1,0}^{\{0,2\}}) \\ &= a^* \quad | \quad (a^*b \quad [\varepsilon|(aa*b)]^* \quad aaa^*) \end{aligned}$$

Which might have a simpler form...

Some questions

(a) Is there an algorithm which, given a string u and a regular expression r, computes whether or not u matches r?

(b) In formulating the definition of regular expressions, have we missed out some practically useful notions of pattern?

(c) Is there an algorithm which, given two regular expressions r and s, computes whether or not they are **equivalent**, in the sense that $L(r)$ and $L(s)$ are equal sets?

(d) Is every language (subset of Σ^*) of the form $L(r)$ for some r?

$Not(M)$

Given DFA $M = (Q, \Sigma, \delta, s, F)$,
then $Not(M)$ is the DFA with

- set of states $= Q$
- input alphabet $= \Sigma$
- next-state function $= \delta$
- start state $= s$
- accepting states $= \{q \in Q \mid q \notin F\}$.

(i.e. we just reverse the role of accepting/non-accepting and leave everything else the same)

Because M is a *deterministic* finite automaton, then u is accepted by $Not(M)$ iff it is not accepted by M:

$$L(Not(M)) = \{u \in \Sigma^* \mid u \notin L(M)\}$$

So regular languages are closed under complementation:

- given a regular expression r

$$\boxed{L(\sim r) = \{u \in \Sigma^* | u \notin L(r)\}}$$

So regular languages are closed under complementation:

- given a regular expression r
- build DFA M such that $L(M) = L(r)$ (Kleene (a))

$$\boxed{L(\sim r) = \{u \in \Sigma^* | u \notin L(r)\}}$$

So regular languages are closed under complementation:

- Given a regular expression r
- Build DFA M such that $L(M) = L(r)$ (Kleene (a))
- Build $Not(M)$ from M (just defined)

$$\boxed{L(\sim r) = \{u \in \Sigma^* | u \notin L(r)\}}$$

So regular languages are closed under complementation:

- given a regular expression r
- build DFA M such that $L(M) = L(r)$ (Kleene (a))
- build $Not(M)$ from M (just defined)
- find $\sim r$ such that $L(\sim r) = L(Not(M))$ (Kleene (b))

$$\boxed{L(\sim r) = \{u \in \Sigma^* | u \notin L(r)\}}$$

Regular languages are closed under intersection

Theorem. If L_1 and L_2 are a regular languages over an alphabet Σ, then their intersection $L_1 \cap L_2 = \{u \in \Sigma^* \mid u \in L_1 \,\&\, u \in L_2\}$ is also regular.

Regular languages are closed under intersection

Theorem. If L_1 and L_2 are a regular languages over an alphabet Σ, then their intersection $L_1 \cap L_2 = \{u \in \Sigma^* \mid u \in L_1 \mathbin{\&} u \in L_2\}$ is also regular.

Proof. Note that $L_1 \cap L_2 = \Sigma^* \setminus ((\Sigma^* \setminus L_1) \cup (\Sigma^* \setminus L_2))$

(*cf.* de Morgan's Law: $p \mathbin{\&} q = \neg(\neg p \vee \neg q)$).

Regular languages are closed under intersection

Theorem. If L_1 and L_2 are a regular languages over an alphabet Σ, then their intersection $L_1 \cap L_2 = \{u \in \Sigma^* \mid u \in L_1 \,\&\, u \in L_2\}$ is also regular.

Proof. Note that $L_1 \cap L_2 = \Sigma^* \setminus ((\Sigma^* \setminus L_1) \cup (\Sigma^* \setminus L_2))$

(*cf.* de Morgan's Law: $p \,\&\, q = \neg(\neg p \vee \neg q)$).

So if $L_1 = L(M_1)$ and $L_2 = L(M_2)$ for DFAs M_1 and M_2,

Regular languages are closed under intersection

Theorem. If L_1 and L_2 are a regular languages over an alphabet Σ, then their intersection $L_1 \cap L_2 = \{u \in \Sigma^* \mid u \in L_1 \,\&\, u \in L_2\}$ is also regular.

Proof. Note that $L_1 \cap L_2 = \Sigma^* \setminus ((\Sigma^* \setminus L_1) \cup (\Sigma^* \setminus L_2))$

(*cf.* de Morgan's Law: $p \,\&\, q = \neg(\neg p \vee \neg q)$).

So if $L_1 = L(M_1)$ and $L_2 = L(M_2)$ for DFAs M_1 and M_2, then $L_1 \cap L_2 = L(Not(PM))$, PM subset-constructed from M,

Regular languages are closed under intersection

Theorem. If L_1 and L_2 are a regular languages over an alphabet Σ, then their intersection $L_1 \cap L_2 = \{u \in \Sigma^* \mid u \in L_1 \,\&\, u \in L_2\}$ is also regular.

Proof. Note that $L_1 \cap L_2 = \Sigma^* \setminus ((\Sigma^* \setminus L_1) \cup (\Sigma^* \setminus L_2))$

(*cf.* de Morgan's Law: $p \,\&\, q = \neg(\neg p \vee \neg q)$).

So if $L_1 = L(M_1)$ and $L_2 = L(M_2)$ for DFAs M_1 and M_2, then $L_1 \cap L_2 = L(Not(PM))$, PM subset-constructed from M, where M is the NFA$^\varepsilon$ $Union(Not(M_1), Not(M_2))$. □

Regular languages are closed under intersection

Theorem. If L_1 and L_2 are a regular languages over an alphabet Σ, then their intersection $L_1 \cap L_2 = \{u \in \Sigma^* \mid u \in L_1 \,\&\, u \in L_2\}$ is also regular.

Proof. Note that $L_1 \cap L_2 = \Sigma^* \setminus ((\Sigma^* \setminus L_1) \cup (\Sigma^* \setminus L_2))$

(*cf.* de Morgan's Law: $p \,\&\, q = \neg(\neg p \vee \neg q)$).

So if $L_1 = L(M_1)$ and $L_2 = L(M_2)$ for DFAs M_1 and M_2, then $L_1 \cap L_2 = L(Not(PM))$, PM subset-constructed from M, where M is the NFA$^\varepsilon$ $Union(Not(M_1), Not(M_2))$. ☐

[It is not hard to directly construct a DFA $And(M_1, M_2)$ from M_1 and M_2 such that $L(And(M_1, M_2)) = L(M_1) \cap L(M_2)$ – see Exercise 4.7.]

Regular languages are closed under intersection

Corollary: given regular expressions r_1 and r_2,there is a regular expression, which we write as $r_1 \,\&\, r_2$, such that

> a string u matches $r_1 \,\&\, r_2$ iff it matches both r_1 and r_2.

Proof. By Kleene (a), $L(r_1)$ and $L(r_2)$ are regular languages and hence by the theorem, so is $L(r_1) \cap L(r_2)$. Then we can use Kleene (b) to construct a regular expression $r_1 \,\&\, r_2$ with $L(r_1 \,\&\, r_2) = L(r_1) \cap L(r_2)$. ☐

Some questions

(a) Is there an algorithm which, given a string u and a regular expression r, computes whether or not u matches r?

(b) In formulating the definition of regular expressions, have we missed out some practically useful notions of pattern?

(c) Is there an algorithm which, given two regular expressions r and s, computes whether or not they are **equivalent**, in the sense that $L(r)$ and $L(s)$ are equal sets?

(d) Is every language (subset of Σ^*) of the form $L(r)$ for some r?

Equivalent regular expressions

Definition. Two regular expressions r and s are said to be **equivalent** if $L(r) = L(s)$, that is, they determine exactly the same sets of strings via matching.

For example, are $b^*a(b^*a)^*$ and $(a|b)^*a$ equivalent?

Equivalent regular expressions

Definition. Two regular expressions r and s are said to be **equivalent** if $L(r) = L(s)$, that is, they determine exactly the same sets of strings via matching.

For example, are $b^*a(b^*a)^*$ and $(a|b)^*a$ equivalent?

Answer: yes (Exercise 2.3)

How can we decide all such questions?

Note that $L(r) = L(s)$

iff $L(r) \subseteq L(s)$ and $L(s) \subseteq L(r)$

Note that $L(r) = L(s)$

iff $L(r) \subseteq L(s)$ and $L(s) \subseteq L(r)$
iff $(\Sigma^* \setminus L(r)) \cap L(s) = \emptyset = (\Sigma^* \setminus L(s)) \cap L(r)$

Note that $L(r) = L(s)$

iff $L(r) \subseteq L(s)$ and $L(s) \subseteq L(r)$
iff $(\Sigma^* \setminus L(r)) \cap L(s) = \emptyset = (\Sigma^* \setminus L(s)) \cap L(r)$
iff $L((\sim r) \,\&\, s) = \emptyset = L((\sim s) \,\&\, r)$

Note that $L(r) = L(s)$

$$\begin{aligned}
&\text{iff } L(r) \subseteq L(s) \text{ and } L(s) \subseteq L(r) \\
&\text{iff } (\Sigma^* \setminus L(r)) \cap L(s) = \emptyset = (\Sigma^* \setminus L(s)) \cap L(r) \\
&\text{iff } L((\sim r) \mathbin{\&} s) = \emptyset = L((\sim s) \mathbin{\&} r) \\
&\text{iff } L(M) = \emptyset = L(N)
\end{aligned}$$

where M and N are DFAs accepting the sets of strings matched by the regular expressions $(\sim r) \mathbin{\&} s$ and $(\sim s) \mathbin{\&} r$ respectively.

Note that $L(r) = L(s)$

$$\begin{aligned}
&\text{iff } L(r) \subseteq L(s) \text{ and } L(s) \subseteq L(r)\\
&\text{iff } (\Sigma^* \setminus L(r)) \cap L(s) = \emptyset = (\Sigma^* \setminus L(s)) \cap L(r)\\
&\text{iff } L((\sim r)\,\&\,s) = \emptyset = L((\sim s)\,\&\,r)\\
&\text{iff } L(M) = \emptyset = L(N)
\end{aligned}$$

where M and N are DFAs accepting the sets of strings matched by the regular expressions $(\sim r)\,\&\,s$ and $(\sim s)\,\&\,r$ respectively.

So to decide equivalence for regular expressions it suffices to

check, given any DFA M, whether or not it accepts *any string at all*.

Note that $L(r) = L(s)$

iff $L(r) \subseteq L(s)$ and $L(s) \subseteq L(r)$
iff $(\Sigma^* \setminus L(r)) \cap L(s) = \emptyset = (\Sigma^* \setminus L(s)) \cap L(r)$
iff $L((\sim r) \mathbin{\&} s) = \emptyset = L((\sim s) \mathbin{\&} r)$
iff $L(M) = \emptyset = L(N)$

where M and N are DFAs accepting the sets of strings matched by the regular expressions $(\sim r) \mathbin{\&} s$ and $(\sim s) \mathbin{\&} r$ respectively.

So to decide equivalence for regular expressions it suffices to

check, given any DFA M, whether or not it accepts *any string at all*.

Note that the number of transitions needed to reach an accepting state in a finite automaton is bounded by the number of states (we can remove loops from longer paths). So we only have to check finitely many strings to see whether or not $L(M)$ is empty.

That gives us our answer to question (c) (which is yes).

Now onto the last of our questions...

The Pumping Lemma

Some questions

(a) Is there an algorithm which, given a string u and a regular expression r, computes whether or not u matches r?

(b) In formulating the definition of regular expressions, have we missed out some practically useful notions of pattern?

(c) Is there an algorithm which, given two regular expressions r and s, computes whether or not they are **equivalent**, in the sense that $L(r)$ and $L(s)$ are equal sets?

(d) Is every language (subset of Σ^*) of the form $L(r)$ for some r?

Examples of languages that are not regular

- The set of strings over $\{(,),a,b,\ldots,z\}$ in which the parentheses '$($' and '$)$' occur well-nested.
- The set of strings over $\{a,b,\ldots,z\}$ which are palindromes, i.e. which read the same backwards as forwards.
- $\{a^n b^n \mid n \geq 0\}$

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. $v \neq \varepsilon$)

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. $v \neq \varepsilon$)
- $|u_1 v| \leq \ell$

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.)

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], $u_1 vv u_2 \in L$, $u_1 vvv u_2 \in L$, 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

$$\begin{tikzcd} s = q_0 \arrow[r, "u_1"] & {}^{*}q_i \arrow[r, phantom, "="] \arrow[r, bend left=60, "v"] & q_j^{*} \arrow[r, "u_2"] & {}^{*}q_n \in F \end{tikzcd}$$

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

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