CSE596 Lecture Fri 9/14 Fall 2018

Def^n: A GNFA is a 5-tuple $N = (Q, \Sigma, \delta, s, F)$, where now δ is a set of "generalized instructions"

$$(p, \alpha, q) \quad \text{where } \alpha \text{ is a regexp over } \Sigma$$

Note: An NFA_ε is the same as a GNFA in which α is a basic regexp $\underset{\sim}{c}$ or $\underset{\sim}{\varepsilon}$ (or $\underset{\sim}{\emptyset}$ for non-edges).

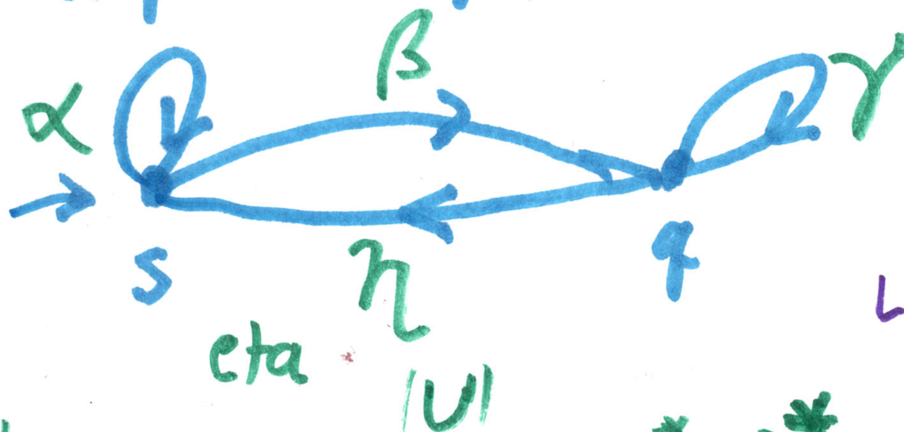
Def^n: A valid computation trace of N on x has the form $(p, u_1, q_1, u_2, q_2, u_3, \ldots, q_{m-1}, u_m, q)$

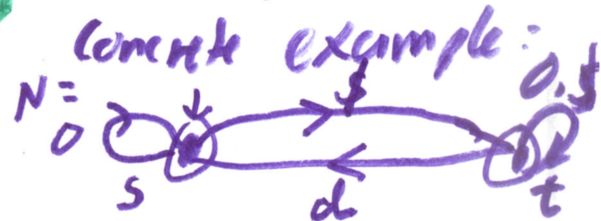
where $u_1 \cdots u_m = x$ and for each j, $1 \le j \le m$, there is an instruction (q_{j-1}, α_j, q_j) s.t. $u_j \in L(\alpha_j)$.

Then we say $x \in L_{pq}$, and $L(N) =_{def} \bigcup_{f \in F} L_{sf}$ as before.

Note: For every regexp α we can build the trivial equivalent GNFA $s \xrightarrow{\alpha} (f)$. The conversion from GNFA to regexp has this as the end case:

Examples and Practical Time-Saver: 2-state GNFA





$$L_{ss} = (\alpha + \beta\gamma^*\eta)^*$$

$$L_{sq} = L_{ss} \cdot \beta\gamma^*$$

also $= \alpha^*\beta \cdot L_{qq} = \alpha^*\beta \cdot (\gamma + \eta\alpha^*\beta)^*$

$L(N) = L_{ss} \cup L_{st}$

$= L_{ss} \cdot (\varepsilon + \$(0+\$)^*)$

where $L_{ss} = (0 + \$(0+\$)^* d)^*$

This explains the regular expression in the "Turing Kit" DragonSh example

Without adding an extra start and final state, we can use this as a base case whenever the original N has at most 1 acc. state besides s (different from s).

Then, let us assume states 3, ..., l are nonaccepting.

Algm: For k = l down to 3:

Bypass and eliminate state k by:

for each incoming edge (i, β, k):

for each outgoing edge (k, η, j):

update any existing instr. (i, α, j)

to $\alpha + \beta\gamma^*\eta$ where (k, γ, k) is the loop at k

<u>Extra</u>: In UNIX, $\alpha? \equiv \alpha + \epsilon$. Best used only with single chars and other short exprs, eg c? or [a-z]? for an optional lowercase letter.

IMHO the precedence of ? is hard to remember — whereas the precedence of +, ·, * is the same as for plus, times, and exponents in math.

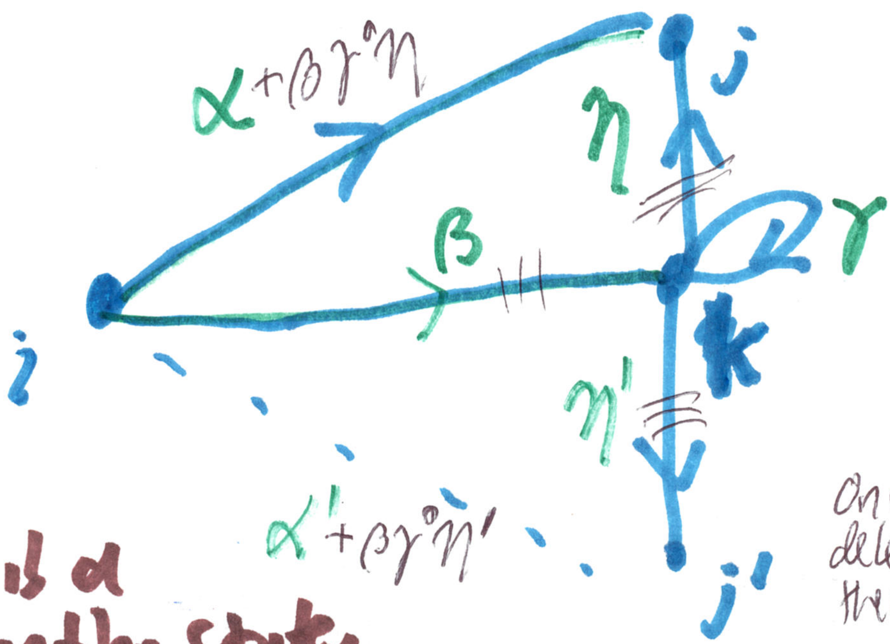
Note how this lecture showcased nested *s as in $(ba^*a + aa^*(b+a^*a$

The combination of allowing ~ as an extra operator on nested-* parts is what can blow up the running time of egrep: extended regular expressions. But if negation is limited to char level (as in [^a-z] for not a lowercase char), then it doesn't blow up. Several NP-hard problems become tractable when a parameter related to allowable nesting is held to a fixed bound. "Fixed-Parameter Tractable"

This also shows why although the regular languages are closed under ~, we hesitate to allow it as a basic operation. Our brains are wired for $\cup$, ·, and *, but not for ~ (and sometimes not for $\cap$ either).

Picture

$\alpha = \emptyset$ possible

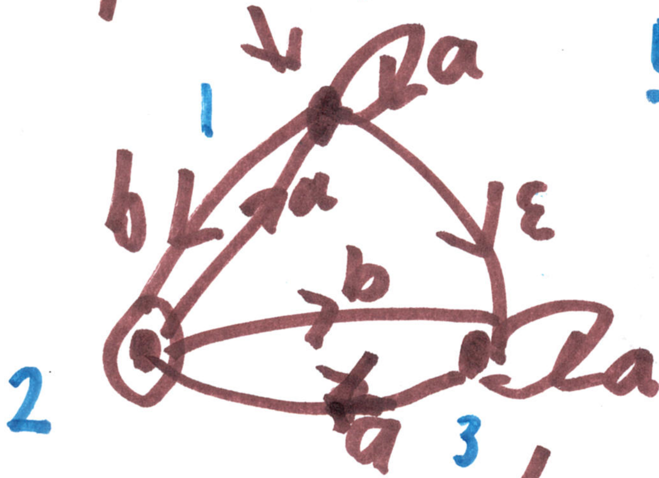


$\gamma = \emptyset$ possible then $\gamma^* = \epsilon$, since $\emptyset^* = \epsilon$. $\beta \neq \emptyset$, $\eta \neq \emptyset$.

Once all outedges are handled delete all incoming edges and then delete k.

Since k is a non-accepting state, any processing that goes in to k from some node i must come out at some node j, having matched $\beta\gamma^*\eta$. It could equivalently match $\alpha + \beta\gamma^*\eta$ directly.

In practice, can shortcut examples



Elim 3:

Inc: (1, ε, 3), (2, b, 3)

Out. (3, a, 2)

∴ update (1, —, 2), (2, —, 2)

$$\alpha + \beta \gamma^* \eta$$

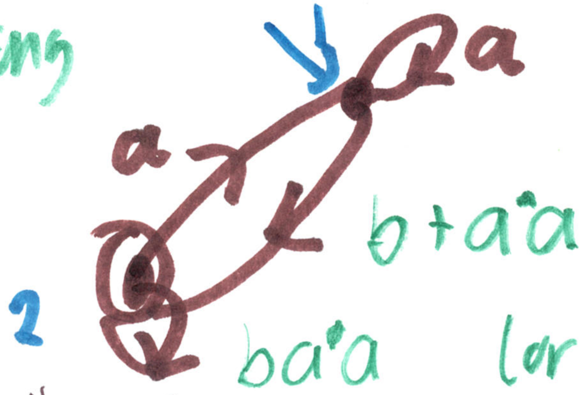
$$(1-2)_{new} = (1-2)_{old} + (1-3)(3-3)^*(3-2)$$

$$= b \quad \varepsilon \cdot a^* \quad a$$

$$(2-2)_{new} = (2-2)_{old} + (2-3)(3-3)^*(3-2)$$

$$\emptyset + b \quad a^* \quad a$$

or ε → won't be wrong



Final answer: $L_{12} = a^*(b+a^*a) \cdot L_{22}$ ("once")

where $L_{22} = (ba^*a + aa^*(b+a^*a))^*$

An extra +ε inside the outer * would not change L_{22} and the final answer.