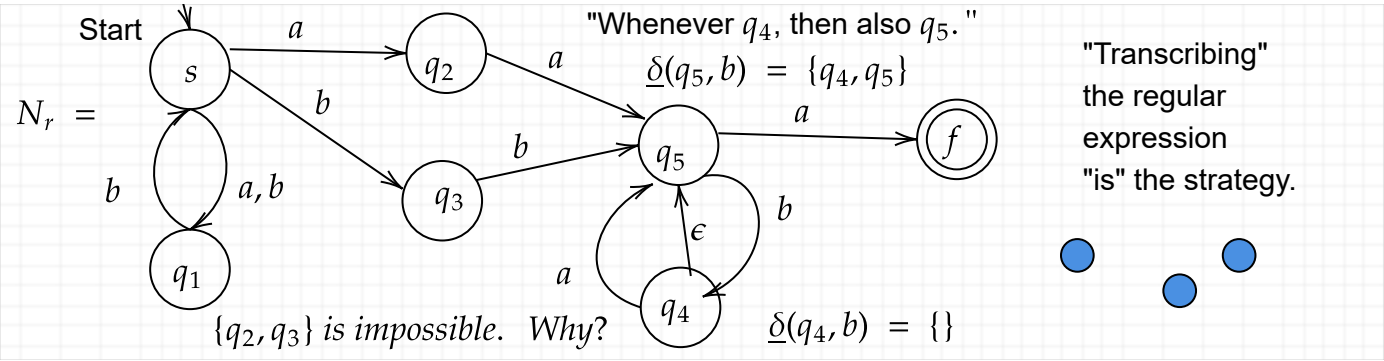
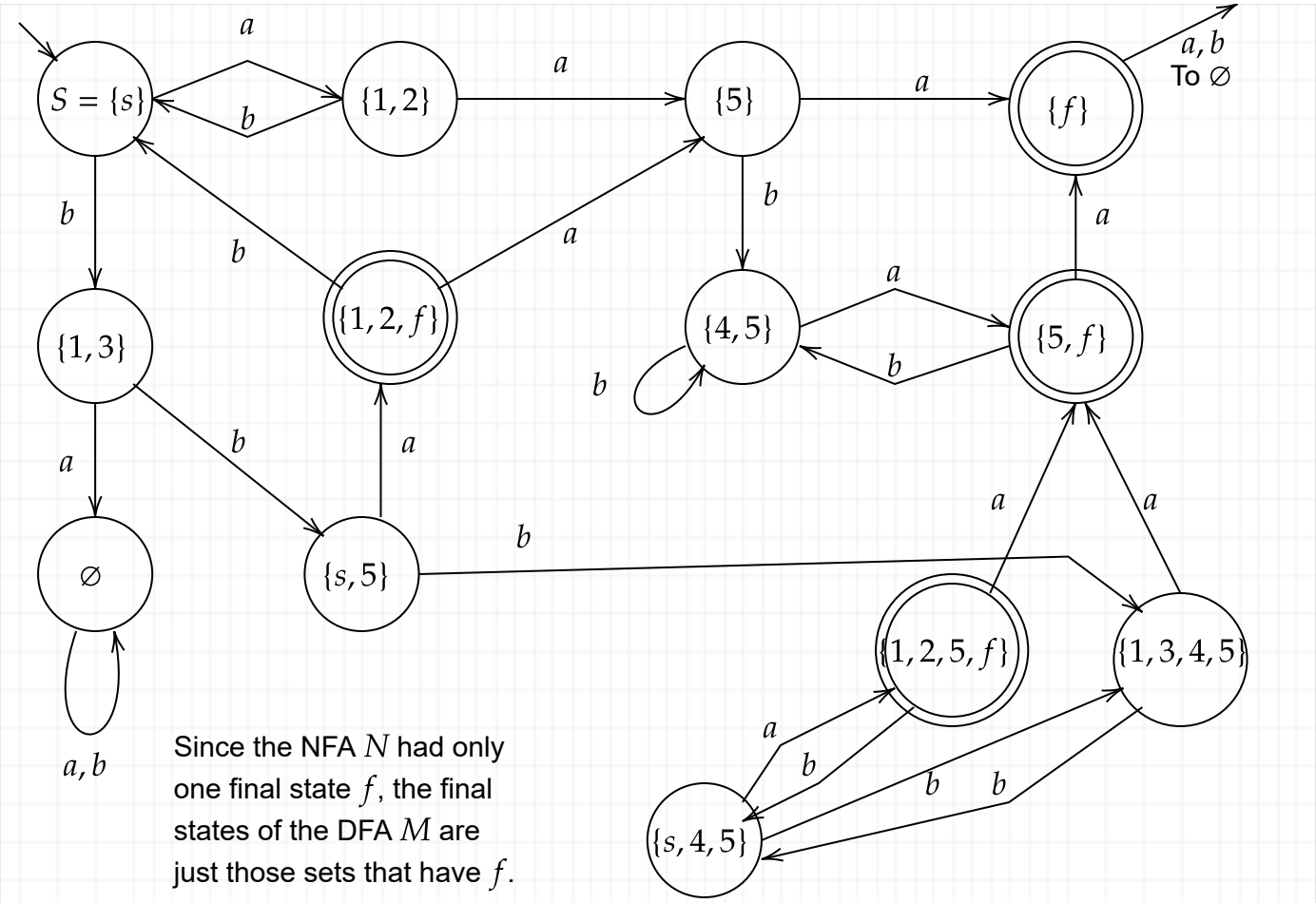
CSE396 Lecture Tue. 2/23: NFAs, DFAs, and GNFAs

Picking up with the long NFA-to-DFA example:



The whole DFA:



[Tue. 2/23 will pick up here, then do the example with ϵ's.]

For any string x, the set-state of the DFA after processing x equals the set of states that N can process x to. Thus, for instance:

- N can process the string bba to any of its states 1, 2, and all the way across to f.

- N can process $bbab$, however, only back to its start state s.
- N accepts aaa but cannot process $aaaa$.
- The shortest string that N can process to four different states is bbb.
- The shortest string that goes to 4 states, one of which is f, however, is $bbbba$.
- There is no string that N can process to more than four different states---in particular, there is no string that "lights up" every state, because the "omni" set-state $\{s, 1, 2, 3, 4, 5, f\} = Q$ was never encountered in the breadth-first search.
- There is no state that guarantees acceptance: every state can reach a rejecting state with more chars. In fact, every state has a path to the dead state.

In other cases, the DFA M may never reach a dead state. It might (also) have an "eternal state", meaning an accepting state that loops to itself. The "omni" state, even when reached, need not be eternal (though if M has any eternal state, "omni" is eternal). M can even have a cluster of accepting states that cycle amongst themselves without ever going to a rejecting state---though such a cluster can then be "condensed" into a single eternal state. This last possibility also tells you that the DFA cranked out by the algorithm is not necessarily optimal in size.

Proof of the Theorem

Recall definitions and specifications of the DFA $M = (\mathbb{Q}, \Sigma, \Delta, S, \mathcal{F})$ from before:

Epsilon closure: $E(R) = \{r : \textit{for some } q \in R,\ N \textit{ can process } \epsilon \textit{ from } q \textit{ to } r\}$

- $\mathbb{Q} = \{\textit{possible } R \subseteq Q\}$;
- Σ is the same;
- $S = E(s)$;
- $\mathcal{F} = \{R \in \mathbb{Q} : R \cap F \neq \varnothing\}$.

$\underline{\delta}(p, c) = \{r :$ you can get from p to r by first processing c at p, then doing any ϵ-arcs$\}$.

- For any $P \in \mathbb{Q}$ (i.e., $P \subseteq Q$ and P is possible) and $c \in \Sigma$ define

$$\Delta(P, c) = \bigcup_{p \in P} \underline{\delta}(p, c).$$

How do we prove $L(M) = L(N)$? What we want to prove is that for every string x, the state R_x that M is in equals the set of states r such that N can process x from s to r. Then the definition of the final states $\mathcal{F}$ of M kicks in to say that the languages are equal.

- Define $G(i)$ to be the statement that this holds for all strings x of length i.
- Then $G(0)$ says that the start state of M should equal the set of states r such that N can process ϵ from s to r. Since this is exactly the meaning of $E(s)$, which is made the start state S of M, the base case $G(0)$ holds.
- To prove $L(M) = L(N)$, then, we only need to show $G(i-1) \implies G(i)$ for each i.

In the step $i = 1$, the fact that S is ϵ-closed sets up the assumption that P in $\Delta(P, c)$ is ϵ-closed. The value $\Delta(P, c)$ is automatically ϵ-closed, since $c \cdot \epsilon^* = c$ so any trailing ϵ-arcs can count as part of processing c. If we---

- assume $G(i-1)$ as our induction hypothesis,
- take the set R_{i-1} which the property $G(i-1)$ refers to, and
- define $R_i = \Delta(R_{i-1}, x_i)$,

---then we only need to show that R_i has the property required for the conclusion $G(i)$. This is that R_i equals the set of states that N can process the bits $x_1 \cdots x_{i-1}x_i$ to. The core of the proof observes that:

> N can process $x_1x_2 \cdots x_{i-1}x_i$ *from s to r* if and only if there is a state p such that N can process $x_1x_2 \cdots x_{i-1}$ from s to p (which by IH $G(i-1)$ includes p into R_{i-1}) and such that N can process the char x_i from p to r.

Then by the inductive hypothesis $G(i-1)$, R_{i-1} equals the set of states q such that N can process $x_1 \cdots x_{i-1}$ from s to q. Now put $R_i = \Delta(R_{i-1}, x_i)$.

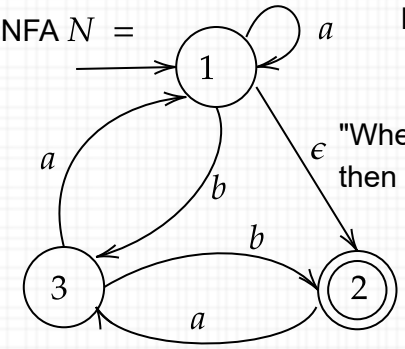
- Let $r \in R_i$. Then $r \in \underline{\delta}(q, x_i)$ for some $q \in R_{i-1}$. By IH $G(i-1)$, N can process $x_1 \cdots x_{i-1}$ from s to q. And N can process x_i from q to r by definition of $r \in \underline{\delta}(q, x_i)$. So N can process $x_1 \cdots x_i$ from s to r.
- Suppose N can process $x_1 \cdots x_i$ from s to r. Then---and this is the key point---the processing goes to some state q just before the char x_i is processed. By IH $G(i-1)$, q belongs to R_{i-1}. Moreover, $r \in \underline{\delta}(q, x_i)$ because we first do the step that processed the char x_i at q, then any trailing ϵ-arcs. Thus $r \in \Delta(R_{i-1}, x_i)$, which means $r \in R_i$.

Thus we have established that R_i equals the set of states r such that N can process $x_1 \cdots x_i$ from s to r. This is the statement $G(i)$, which is what we had to prove to make the induction go through. This finally proves the NFA-to-DFA part of Kleene's Theorem. ☒

More examples:

The first one differs from the hand-drawn example after it in having 2 not 3 be the accepting state.

NFA N =



"Whenever 1 then also 2."

We could have made state 1 accepting too.

States $\{1\}$ and $\{1,3\}$ are not possible in M.

$\underline{\delta}(p,c)$ = "first do c then any ϵ's."

$\underline{\delta}$	a	b
1	$\{1,2\}$	$\{3\}$
2	$\{3\}$	$\varnothing$
3	$\{1,2\}$	$\{2\}$

$\mathcal{F}$ = "anything with 2" = $\{\{1,2,3\},\{1,2\},\{2,3\},\{2\}\}$

DFA M = $S = E(\{1\}) = \{1,2\}$ This is an accepting state of M

$\Delta(S,a) = \underline{\delta}(1,a) \cup \underline{\delta}(2,a) = \{1,2\} \cup \{3\} = \{1,2,3\}$

$\Delta(S,b) = \underline{\delta}(1,b) \cup \underline{\delta}(2,b) = \{3\} \cup \{\} = \{3\}$

State $\{3\}$ counts as "new" state even though N has it.

$\Delta(\{1,2,3\},a) = \{1,2\} \cup \{3\} \cup \{1,2\} = \{1,2,3\}$

Must be $\{1,2,3\}$ since we got to the "omni" state on a.

$\Delta(\{1,2,3\},b) = \{3\} \cup \varnothing \cup \{2\} = \{2,3\}$

Not "omni" but is new. Doing state {3} next:

$\Delta(\{3\},a) = \underline{\delta}(3,a) = \{1,2\}$. Not new, back to S.

$\Delta(\{3\},b) = \underline{\delta}(3,b) = \{2\}$. Is new. (And is trouble)

$\Delta(\{2,3\},a) = \{3\} \cup \{1,2\} = \{1,2,3\}$

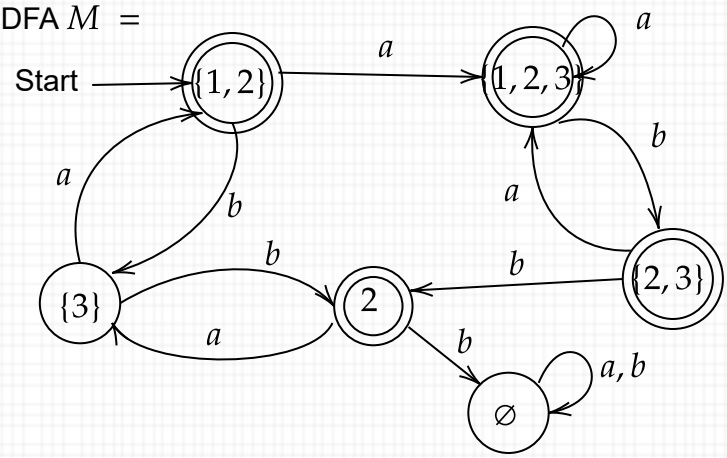
$\Delta(\{2,3\},b) = \varnothing \cup \{2\} = \{2\}$ Just {2} left now.

$\Delta(\{2\},a) = \underline{\delta}(2,a) = \{3\}$ Not new.

$\Delta(\{2\},b) = \underline{\delta}(2,b) = \varnothing$ So M has a dead state.

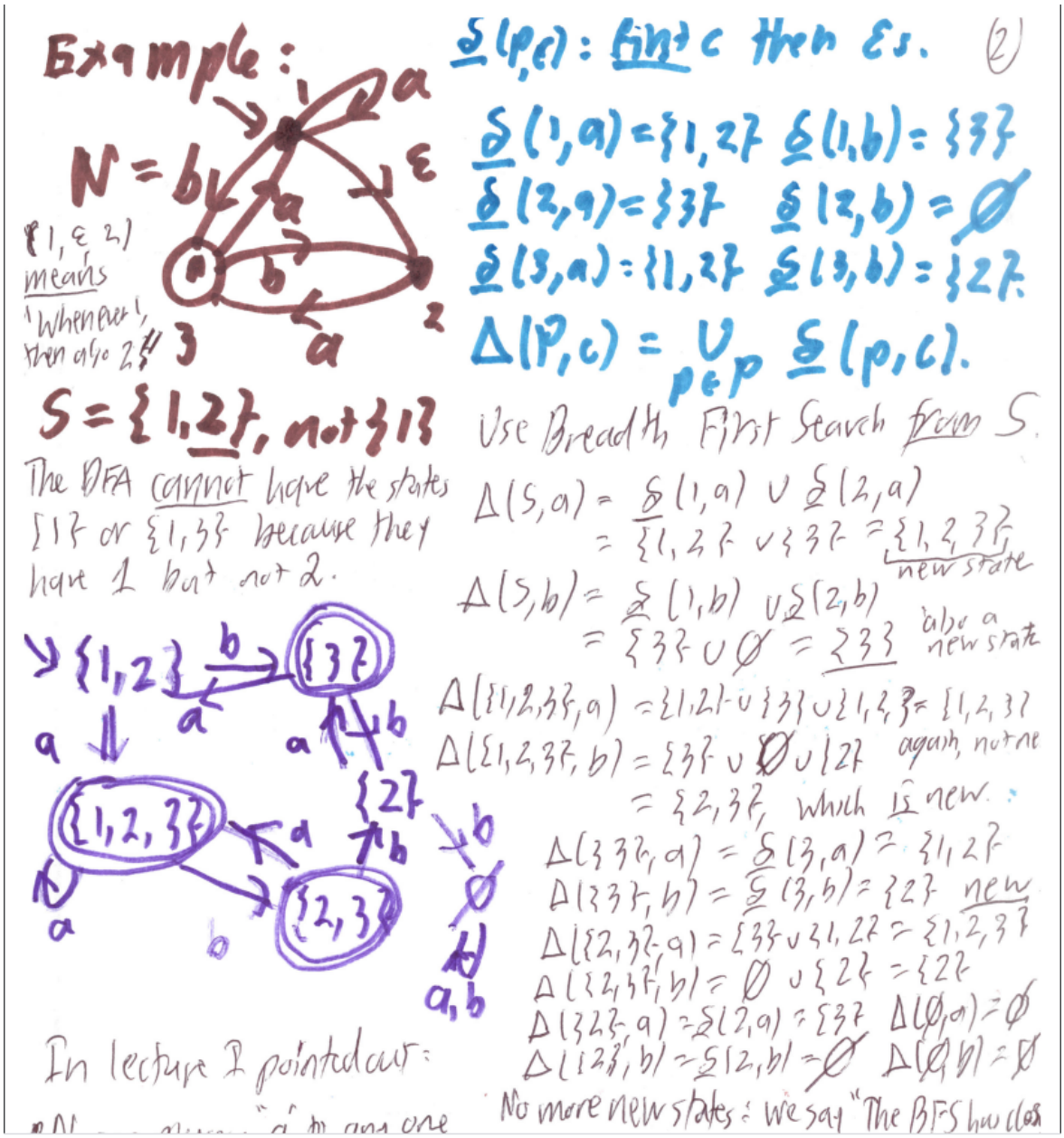
$\Delta(\varnothing,a) = \varnothing, \Delta(\varnothing,b) = \varnothing$. BFS has closed: done.

DFA M =



Note that the original NFA N can process a from s to any one of its three states. But N can't process bbb from any of its three states.

This DFA M has a dead state but does not have any eternally accepting (cluster of) states.

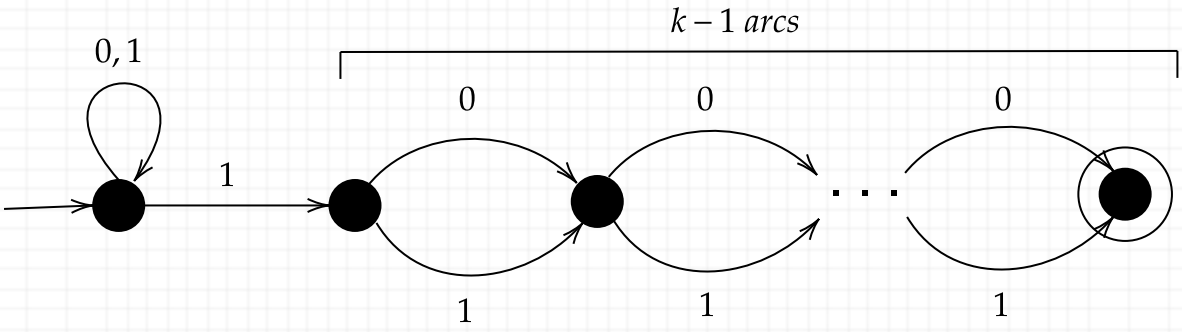


More on how the states of the DFA tell what the NFA can and cannot process:

- The NFA cannot process the string bbb from its start state at all. However you try, you come to the NFA state 2 being unable to process a b. Nor can it process bbb from any other state.
- However, N can process a from start to any one of its three states:
  - $(1, a, 1)$
  - $(1, a, 1)(1, \epsilon, 2)$
  - $(1, \epsilon, 2)(2, a, 3)$.

    This is shown in the DFA by the single arc $(S, a, \{1, 2, 3\})$.
- But in the string $x = abbb$, even though the initial a "turns on all three lightbulbs of N", the final bbb still cannot be processed by N. The DFA M does process it via the computation $(S, a, \{1, 2, 3\})(\{1, 2, 3\}, b, \{2, 3\})(\{2, 3\}, b, \{2\})(\{2\}, b, \varnothing)$, but that computation ends at $\varnothing$, which---when present at all---is always a dead state.

Another example [try for $k = 2$ *or* 3]: The "Leap of Faith" NFAs N_k for any $k > 1$:



$$L(N_k) = (0+1)^*1(0+1)^{k-1}$$
$$= \{x \in \{0,1\}^* : \textit{the kth bit of } x \textit{ from the end is a } 1\}.$$

Fact (will be proved next week): Whereas the NFA N_k has only $k+1$ states, the smallest DFA M_k such that $L(M_k) = L(N_k)$ requires 2^k states. This is a case of **exponential blowup** in the NFA-to-DFA algorithm.

Now here is a simple algorithm for telling whether a given string x *matches* a given regexp α:

1. Convert α into an equivalent NFA N_α.
2. Convert N_α into an equivalent DFA M_α.
3. Run M_α on x. If it accepts, say "*yes*, it matches", else say "no match".

This algorithm is *correct*, but it is *not efficient*. The reason is that step 2 can blow up. An intuitive reason for the gross inefficieincy is that step 2 makes you create in advance all the "set states" that would ever be used on all possible strings x, but most of them are unnecessary for the particular x that was given.

There is, however, a better way that builds just the set-states $R_1, \ldots, R_i, \ldots, R_n$ that are actually encountered in the particular computation on the particular x. We have $R_0 = S = E(s)$ to begin with. To build each R_i from the previous R_{i-1}, iterate through every $q \in R_{i-1}$ and union together all the sets $\underline{\delta}(q, x_i)$. If N_α has k states---which roughly equals the number of operations in α---then that takes order $n \cdot k \cdot k$ steps. This is at worst cubic in the length $\widetilde{O}(n+k)$ of x and α together, so this counts as a **polynomial-time algorithm**. It is in fact the algorithm actually used by the `grep` command on Linux/UNIX. (The tilde above the O means ignoring factors of $\log n$.)

Generalized NFAs (GNFAs)

I view these as mathematical bookkeeping devices, not as "real" as NFAs, let alone DFAs. The meaning of an arc from a state p to a state q is "all ways we know so far to get from state p to state q, in

however many steps." By getting from state p to state q we mean a string processed along the way, so our notation $L_{p,q} = \{w \in \Sigma^* : N \text{ can process } w \text{ from } p \text{ to } q\}$ comes into play. We will see that this language is always regular "in the final analysis." Moreover, insofar as ϵ or one single character a or b (etc.) "is-a" basic regular expression, we start off with arcs labeled by regular expressions with an NFA anyway. And a loop or edge labeled a, b could really be the non-basic regular expression $a \cup b$ anyway.

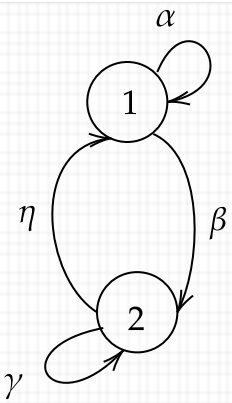
So let us generalize arcs to any regular expressions over the alphabet Σ, letting Regexp(Σ) stand for that.

Definition: A generalized NFA (GNFA) is a 5-tuple $G = (Q, \Sigma, \delta, s, F)$ where Q, Σ, s, F are the same as in an NFA but now $\delta \subseteq (Q \times \text{Regexp}(\Sigma)) \times Q$.

Definition: A sequence $\vec{c} = [q_0, u_1, q_1][q_1, u_2, q_2][q_2, u_3, q_3] \cdots [q_{t-2}, u_{t-1}q_{t-1}][q_{t-1}, u_t, q_t]$ is a **valid computation** if for each i, $1 \leq i \leq t$, there is an instruction $(q_{i-1}, \alpha, q_i) \in \delta$ such that the string u_i matches the regular expression α. The **string processed** by the computation is the string $u = u_1 \cdot u_2 \cdot u_3 \cdots u_{t-1} \cdot u_t$, which might be shorter or longer than t. Then we say the GNFA G can process u from q_0 to q_t and write $u \in L_{q_0,q_t}$ (with G understood). Then as before,

$$L(G) = \bigcup_{f \in F} L_{s,f}.$$

The idea can be put across less technically even when we use abstract regular expressions $\alpha, \beta, \gamma, \eta$ (alpha, beta, gamma, eta).



$$L_{1,1} = (\alpha + \beta\gamma^*\eta)^*$$

$$L_{1,2} = L_{1,1} \cdot \beta\gamma^* = (\alpha + \beta\gamma^*\eta)^*\beta\gamma^*$$

$$L_{2,2} = (\gamma + \eta\alpha^*\beta)^*$$

$$L_{2,1} = L_{2,2} \cdot \eta\alpha^* = (\gamma + \eta\alpha^*\beta)^*\eta\alpha^*$$

$L_{1,2}$ also equals $\alpha^*\beta \cdot L_{2,2} = \alpha^*\beta(\gamma + \eta\alpha^*\beta)^*$.

These become more user-friendly base cases than what rthe text does for the FA-to-regexp proof in section 1.3, which will complete the proof of Kleene's Theorem of equivalence on Thursday.

Notes for Thursday (now posted separately, will be edited up or down a bit)

[I didn't quite cover the above in time on Tue. 2/23, but my notes were planning to reiterate it anyway. The purpose of using 2-state GNFAs as the base case is that (i) they save you the trouble of having to make a new state state, (ii), they often save having to make a new single final state as well, and (iii) they always save the final step of the text's algorithm which often involves having to cut and paste excruciatingly long regular expressions. On HWs and exams you'll be able to save that pencil-pushing by just saying the single Greek letters and saying whether the language is $L_{s,s}$, $L_{s,f}$, or $L_{s,s} \cup L_{s,f}$.]