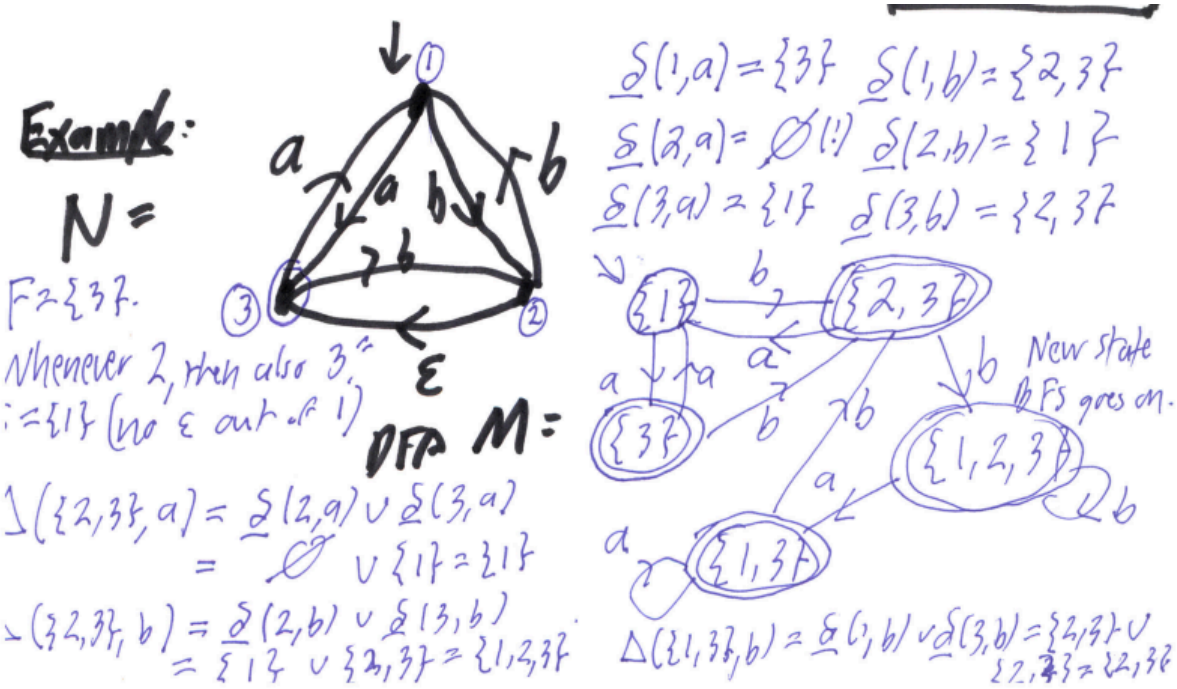
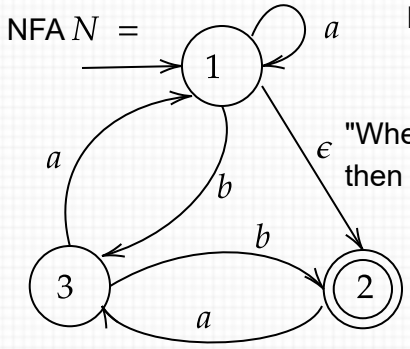
CSE396 Spring 2026 Week 4 Tue.: NFA-to-DFA Examples

[Lecture will first finish the example begun on Thu. 2/5.]



More examples:

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



"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."

$\underline{\delta}$	$a\epsilon^*$	$b\epsilon^*$
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)

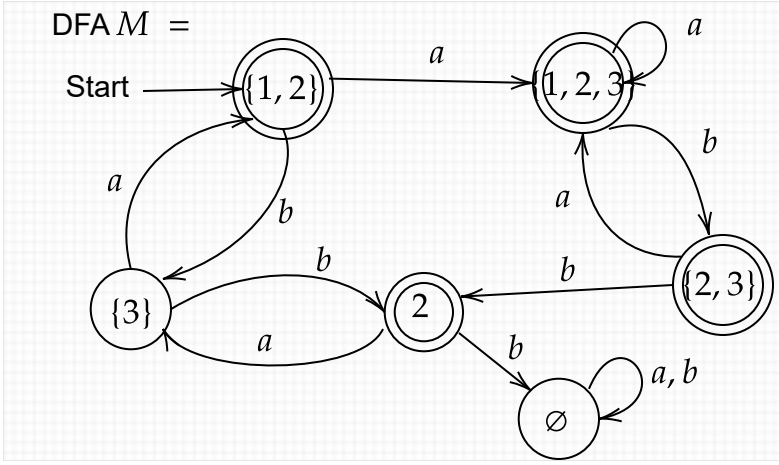
$\Delta(\{2,3\},a) = \{3\} \cup \{1,2\} = \{1,2,3\}$

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



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.

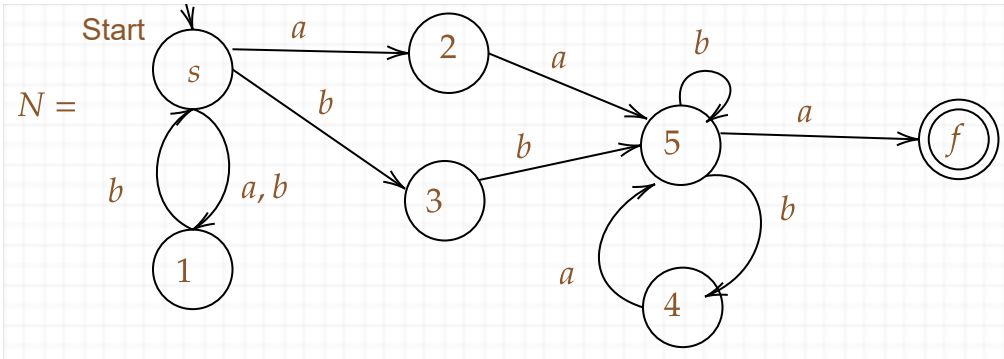
This DFA M has a dead state but does not have any eternally accepting (cluster of) states.

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.

[*** I will skim/skip this big example in lecture, but it is good to work it out for study. ***]



δ	a	b
s	$\{1, 2\}$	$\{1, 3\}$
1	$\varnothing$	$\{s\}$
2	$\{5\}$	$\varnothing$
3	$\varnothing$	$\{5\}$
4	$\{5\}$	$\varnothing$
5	$\{f\}$	$\{4, 5\}$
f	$\varnothing$	$\varnothing$

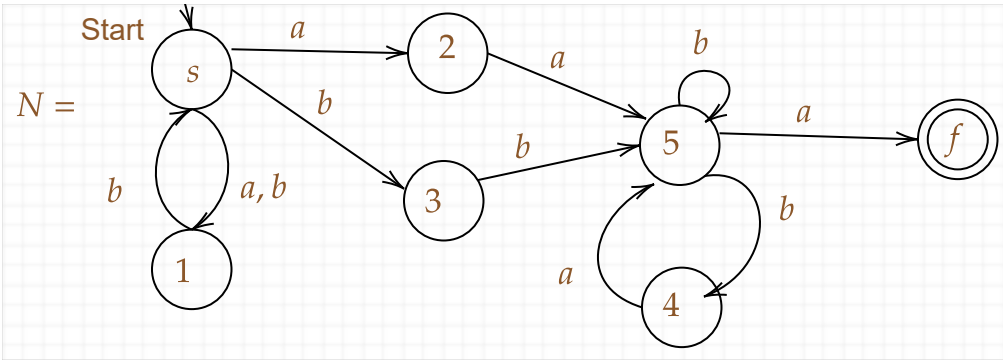
$$\Delta(P, c) = \bigcup_{p \in P} \underline{\delta}(p, c).$$

Since there are no ϵ's out of the start state (or at all), S is just $\{s\}$.
$\Delta(S, a) = \Delta(\{s\}, a) = \{1, 2\}$ (new set-state)
$\Delta(S, b) = \{1, 3\}$ (new set-state). Expand $\{1, 2\}$ first:

$\Delta(\{1,2\},a) = \delta(1,a) \cup \delta(2,a) = \varnothing \cup \{5\} = \{5\}$ (new set-state, append to expansion queue)
$\Delta(\{1,2\},b) = \delta(1,b) \cup \delta(2,b) = \{s\} \cup \varnothing = \{s\}$. Not new--back to the start state of the DFA.

$\Delta(\{1,3\},a) = \delta(1,a) \cup \delta(3,a) = \varnothing \cup \varnothing = \varnothing$. This means that the DFA has a reachable dead state. We can say $\Delta(\varnothing,a) = \Delta(\varnothing,b) = \varnothing$ right off the bat, no need to expand further.
$\Delta(\{1,3\},b) = \delta(1,b) \cup \delta(3,b) = \{s\} \cup \{5\} = \{s,5\}$. New state. But expand $\{5\}$ next.

$\Delta(\{5\},a) = \{f\}$ (new)
$\Delta(\{5\},b) = \{4,5\}$ (new). Now we come to expand $\{s,5\}$ but we have more stuff in the queue.



δ	a	b
s	$\{1,2\}$	$\{1,3\}$
1	$\varnothing$	$\{s\}$
2	$\{5\}$	$\varnothing$
3	$\varnothing$	$\{5\}$
4	$\{5\}$	$\varnothing$
5	$\{f\}$	$\{4,5\}$
f	$\varnothing$	$\varnothing$

$\Delta(\{s,5\},a) = \{1,2\} \cup \{f\} = \{1,2,f\}$. New again---this might worry us about blowup.
$\Delta(\{s,5\},b) = \{1,3\} \cup \{4,5\} = \{1,3,4,5\}$. Even more uh-oh...

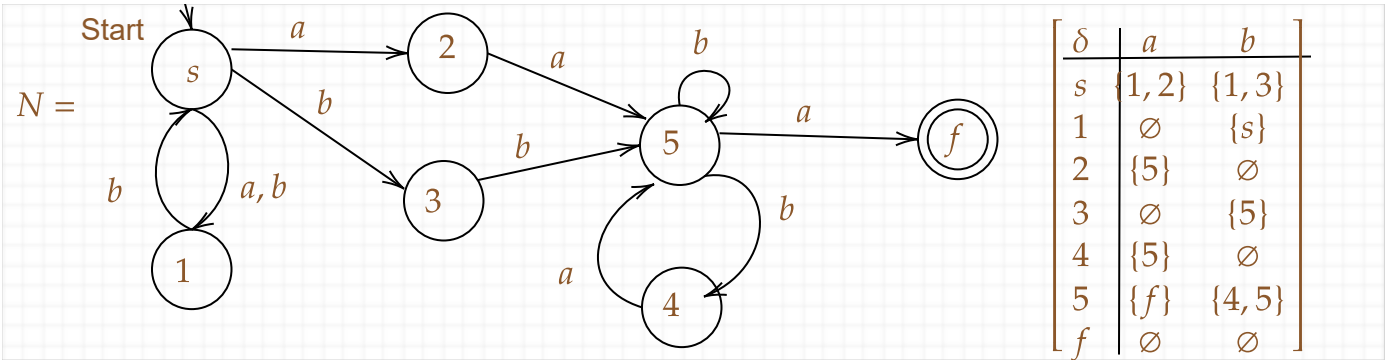
What this means is that the string bbb can be processed from s to four different states! Keep going:
$\Delta(\{f\},a) = \Delta(\{f\},b) = \varnothing$. OK, that one was quick.
$\Delta(\{4,5\},a) = \delta(4,a) \cup \delta(5,a) = \{5\} \cup \{f\} = \{5,f\}$. New, but adding f to $\{5\}$ is no biggie.
$\Delta(\{4,5\},b) = \delta(4,b) \cup \delta(5,b) = \varnothing \cup \{4,5\} = \{4,5\}$. Not new---we just looped back.

$\Delta(\{1,2,f\},a) = \delta(1,a) \cup \delta(2,a) \cup \delta(f,a) = \varnothing \cup \{5\} \cup \varnothing = \{5\}$. Old, $= \Delta(\{1,2\},a)$.
$\Delta(\{1,2,f\},b) = \delta(1,b) \cup \delta(2,b) \cup \delta(f,b) = \{s\} \cup \varnothing \cup \varnothing = \{s\}$. Of course, $= \Delta(\{1,2\},b)$.

Drumroll: because $\{5,f\}$ will work out the same as $\{5\}$ we really have just the one big state to go.

$\Delta(\{1,3,4,5\},a) =$ eyeball rows 1,3,4,5 in column a of the table $= \{5,f\}$. Almost home...
$\Delta(\{1,3,4,5\},b) =$ eyeball column b, we see $\{s,4,5\}$. *Thunderation!*---this is another new state.

$\Delta(\{5,f\},a) = \Delta(\{5\},a)$ since nothing comes out of f, so it $= \{f\}$.
$\Delta(\{5,f\},b) = \Delta(\{5\},b) = \{4,5\}$ which is now old, so all eyes now on expanding $\{s,4,5\}$.



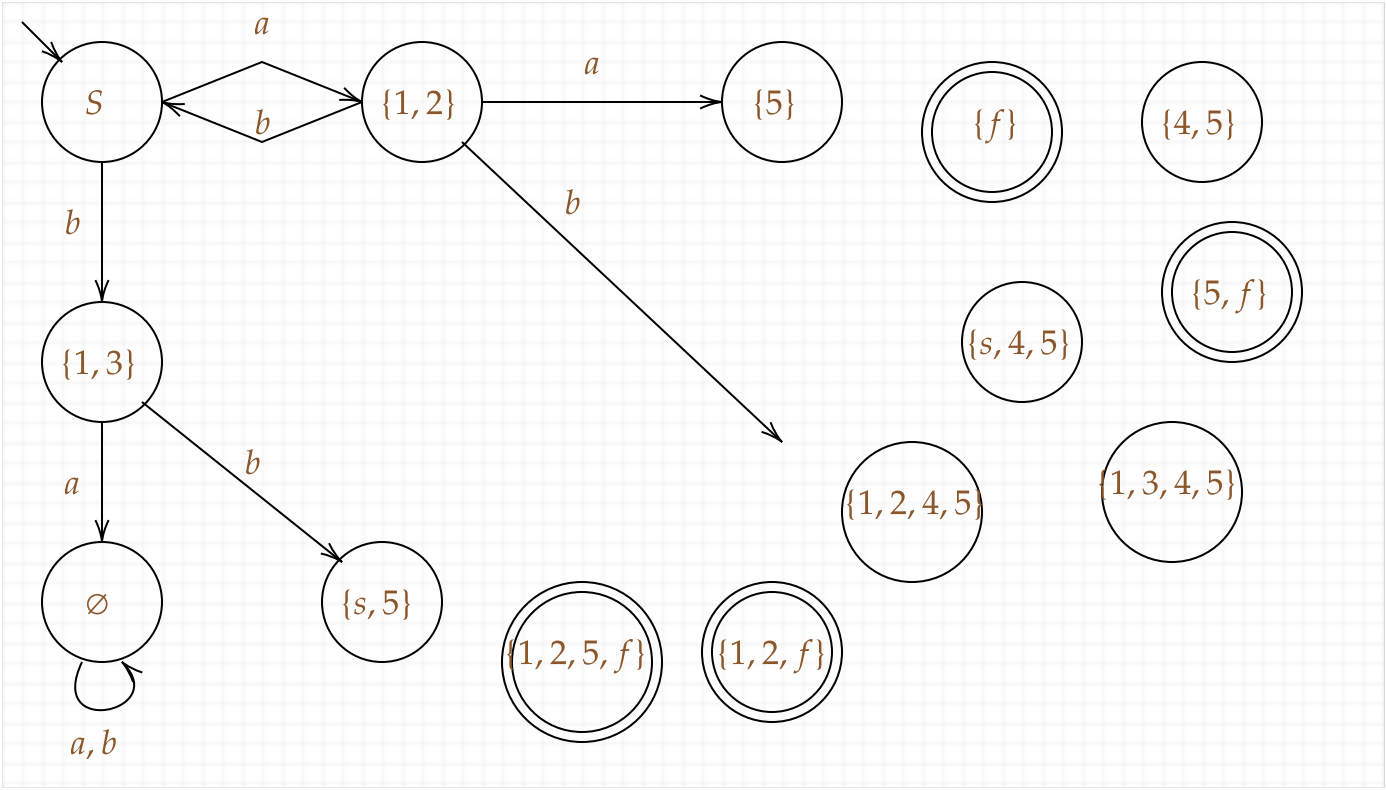
$\Delta(\{s,4,5\},a) = \{1,2\} \cup \{5\} \cup \{f\} = \{1,2,5,f\}$. New

$\Delta(\{s,4,5\},b) = \{1,3\} \cup \varnothing \cup \{4,5\} = \{1,3,4,5\}$. Not New. Keep Going...

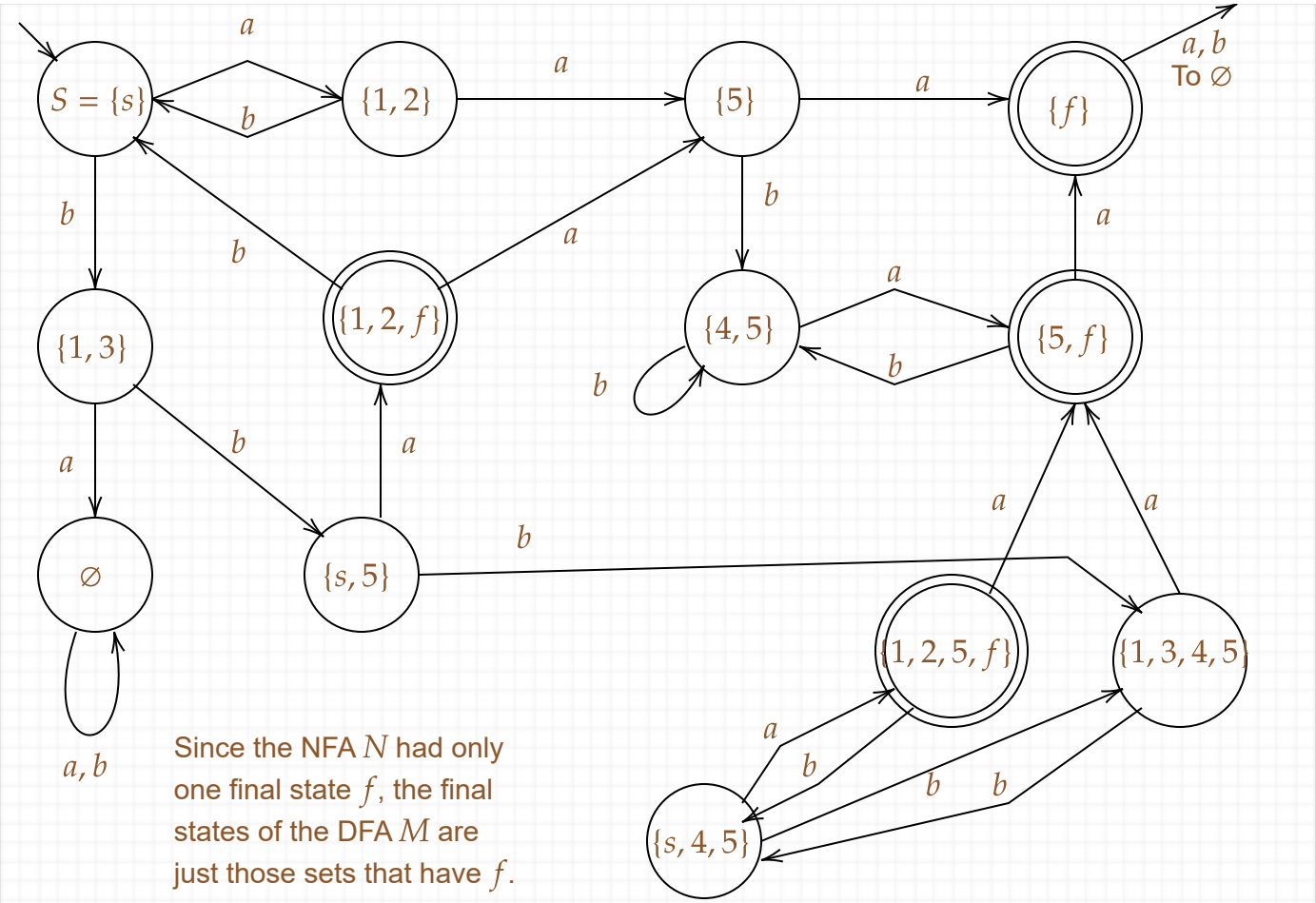
$\Delta(\{1,2,5,f\},a) = \{5,f\}$. Not new.

$\Delta(\{1,2,5,f\},b) = \{s,4,5\}$. Also not new. Just like that, the DFA has closed!

Once you get used to inspecting the δ or $\underline{\delta}$ table, you can draw the DFA as you go without writing out so many sets. Here it is after the first two generations:



The whole DFA:



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 state---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 [In 2026, I may skip this too.]

First recall these definitions and notations:

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;
- $S = E(s)$;
- $\mathcal{F} = \{R \in \mathbb{Q} : R \cap F \neq \emptyset\}$.

  $\underline{\delta}(p, c) = \{r :$ 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:

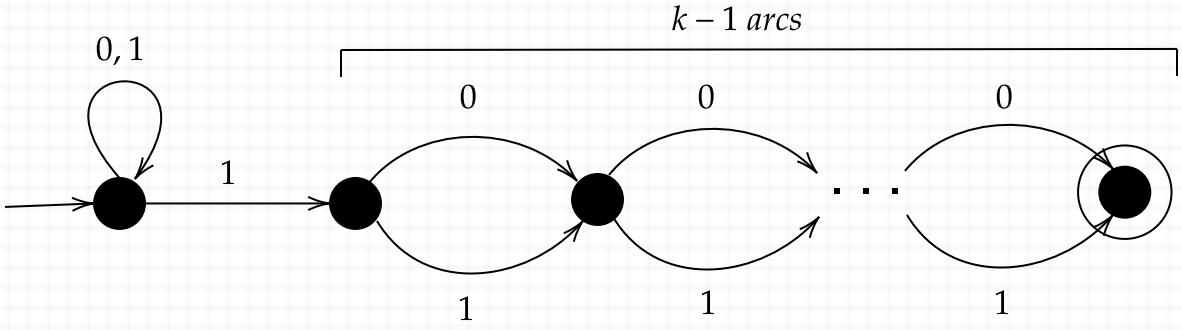
N can process $x_1 x_2 \cdots x_{i-1} x_i$ *from* s *to* r if and only if there is a state p such that N can process $x_1 x_2 \cdots x_{i-1}$ from s to p (which by IH $G(i-1)$ includes p into R_{i-1}) and such that N can process the char x_i from p to r.

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. ☒

Another example: 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 the week after next): 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 in Linux/UNIX.

Kleene's Equivalence Theorem

Stephen Kleene (pronounced clay-nee) proved most of the following in the mid-1950s:

Theorem. For any language A over any alphabet Σ, the following statements are equivalent:

1. There is a regular expression r such that $L(r) = A$.
2. There is an NFA N (allowing ϵ-transitions) such that $L(N) = A$.
3. There is a DFA M such that $L(M) = A$.

We proved $1 \implies 2$ simultaneously with formally defining regular expressions last Thursday, and we've exemplified $2 \implies 3$. Kleene actually did $1 \implies 3$ and $3 \implies 1$ directly, though with a neural-net version of DFAs that carried a whiff of NFAs. The full definition of NFA and the NFA-to-DFA step came in a 1959 paper by Michael Rabin and Dana Scott (who are both still living in 2/26). Since a DFA "Is-A" NFA, we can complete the cycle by showing $2 \implies 1$ just as well. Actually, we will start from an even more general kind of finite "machine".

[Lecture ended here. This $\sim$ where the Week 4 Tuesday lecture ended in 2021, anyway.]