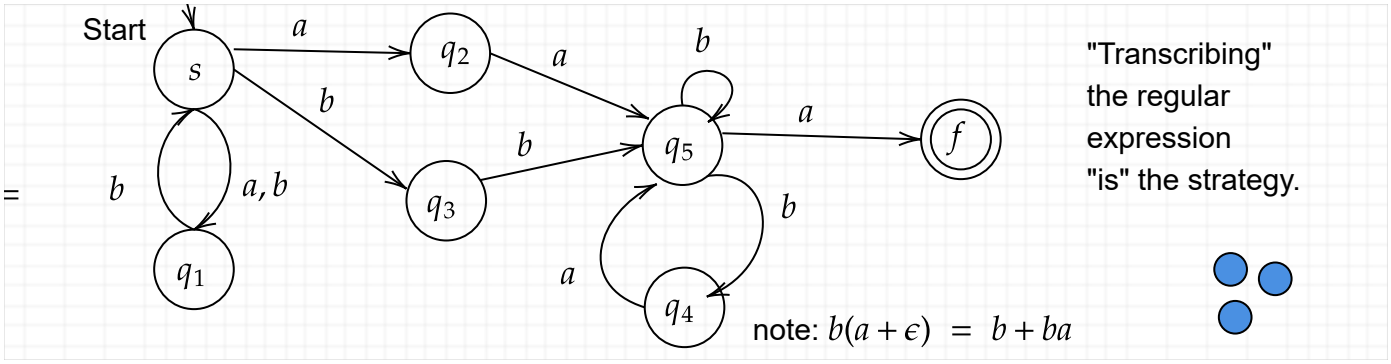
CSE396 Lecture Thu. 2/18: From NFA to DFA

Example: $r = (ab + bb)^*(aa + bb)(b(a + \epsilon))^*a$.



How can we track this machine on an input such as $x = bbaabbaa$? We can try individual computations by trial-and-error:

$(s, b, q_3, b, q_5, a, f, a,$ ---? Crash!
$(s, b, q_1, b, s, a, q_1, a,$ ---? Crash!
$(s, b, q_1, b, s, a, q_2, a, q_5, b, q_4, b,$ --- Crash!
$(s, b, q_1, b, s, a, q_2, a, q_5, b, q_5, b, q_5, a, f, a,$ --- Cannot process the final a, so Crash!
$(s, b, q_1, b, s, a, q_2, a, q_5, b, q_5, b, q_4, a, q_5, a, f)$: end of string, and state is f, so *accept*.

The idea of the DFA is to keep track of all the possibilities in-parallel:

$(\{s\}, b, \{q_1, q_3\}, b, \{s, q_5\}, a, \{f, q_1, q_2\}, a, \{q_5\}, b, \{q_4, q_5\}, b, \{q_4, q_5\}, a, \{f, q_5\}, a, \{f\})$.

Theorem (part two of Kleene's Theorem): Given any NFA $N = (Q, \Sigma, \delta, s, F)$ we can build a DFA $M = (\mathbb{Q}, \Sigma, \Delta, S, \mathcal{F})$ such that $L(M) = L(N)$.

Notice that s got capitalized to S, which hints that S is a *set* rather than a single element. And δ got capitalized to Δ. Q and F were already sets, but they got...curlier. What does that mean? Well, that they are "of an even higher order"---sets of sets, for instance. An important set of sets is:

$$\mathcal{P}(Q), \text{ also written } 2^Q, \text{ called the } \textit{power set} \text{ of } Q \text{ and defined as } \{R : R \subseteq Q\}.$$

Unlike what textbooks tend to say, we will not necessarily make $\mathbb{Q}$ be all of $\mathcal{P}(Q)$, just those subsets R that are *reachable* from S. What this means is that the states of the DFA will be sets of states of the NFA---the states that are *possible* upon *processing* a given part of the input string x.

This suggests the question, which states (of N) are possible **before** we process any chars in x? Obviously the start state s of N is possible, but are there any others? Yes, if there are ϵ-transitions out of s. Define $E(s)$ to be the set of states of N that are reachable this way. If N has no ϵ-arcs (out of s or overall), then $E(s)$ is just $\{s\}$. Thus we begin building M by taking

$$S = E(s).$$

We could have said "S" in place of "$E(s)$" to begin with, but the E notation in the textbook is useful because we can use it to define the following for any set $R \subseteq Q$ of states of N:

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

This is called the *epsilon-closure* of R. If $E(R) = R$ then R is already *epsilon-closed*. It sounds "weeny" technical, but we will only need to use subsets that are ϵ-closed. The insights are

- *The states of the DFA need only be the **possible** subsets of states of the NFA.*
- *A subset R is good if it contains at least one accepting state, i..e, if $R \cap F \neq \emptyset$, because that will mean it is possible for the NFA to accept the string.*

We are thus ready to specify this much of the DFA:

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

The only component of M left to define is Δ. For any $P \in \mathbf{Q}$ and $c \in \Sigma$ define

$$\Delta(P, c) = \{r : \textit{for some } p \in P, N \textit{ can process } c \textit{ from } p \textit{ to } r\}.$$

This means that (p, c, r) can be a virtual instruction, but it might not be a literal instruction in δ because we might have to process ϵ's before and after the character c. We can save half the trouble by realizing that any "ϵ's before" are taken care of by the possible set-states P already being ϵ-closed. The text doesn't say this, but when solving these problems, it is IMHO a help to use the following definition first to build a table from the given NFA:

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

More formally, $\underline{\delta}(p, c) = \{r : (\exists q)[(p, c, q) \in \delta \wedge r \in E(q)\}$. Then for **possible** $P \in \mathbf{Q}$ and $c \in \Sigma$, we get the equivalent definition

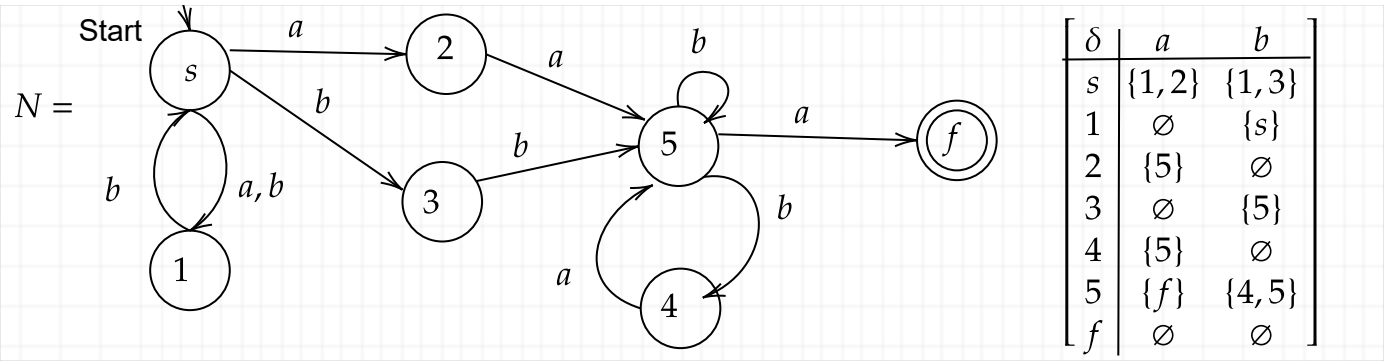
$$\Delta(P, c) = \bigcup_{p \in P} \underline{\delta}(p, c).$$

Even if there are no ϵ's, the idea of limiting to "possible" P often helps in a second way: we avoid having to define instructions for set-states that are never actually encountered. At the beginning, we *encounter* S. Then "expanding" S means computing $\Delta(S, c)$ for each char c. Thus, if $\Sigma = \{0, 1\}$ then the "first generation" are the states $P_0 = \Delta(S, 0)$ and $P_1 = \Delta(S, 1)$. One (or both) of these might equal S again, in which case we have nothing more to do with it. But whichever one(s) are new need to be expanded again to fill out the "second generation." We keep on expanding new set-states---"new" meaning we have not encountered that exact set before---until a generation turns up no new state. Then we say "the DFA has closed" and we're done.

[FYI: It is really a **breadth-first search** that has closed. If you've seen breadth-first search executed on graphs, this one is scaled up in a big way. It is not done on the graph G_N of N but rather on the potentially exponentially bigger graph $\mathcal{G}$ whose nodes are the sets of states. The graph $\mathcal{G}$ is given **implicitly** via the table $\underline{\delta}$ and the rule for Δ. Just don't think you necessarily have to write out all $2^4 = 16$ set-states when given a 4-state NFA like some sources show in diagrams.

If there are no ϵ's, then $\underline{\delta}$ is just the same as the text's set-valued δ function. But when there are ϵ's, writing out the $\underline{\delta}$ table (which cuts out the ϵ's) is a much better use of your time, IMPO, than just copying out the text's δ table with the ϵ column. Why just recopy information that is already explicitly present in the diagram? Whereas, IMHO, the step from N to $\underline{\delta}$ is not typo-prone when done on-the-fly and most usefully breaks your work in half.]

Example (first without any ϵ's)---a large example, in fact:

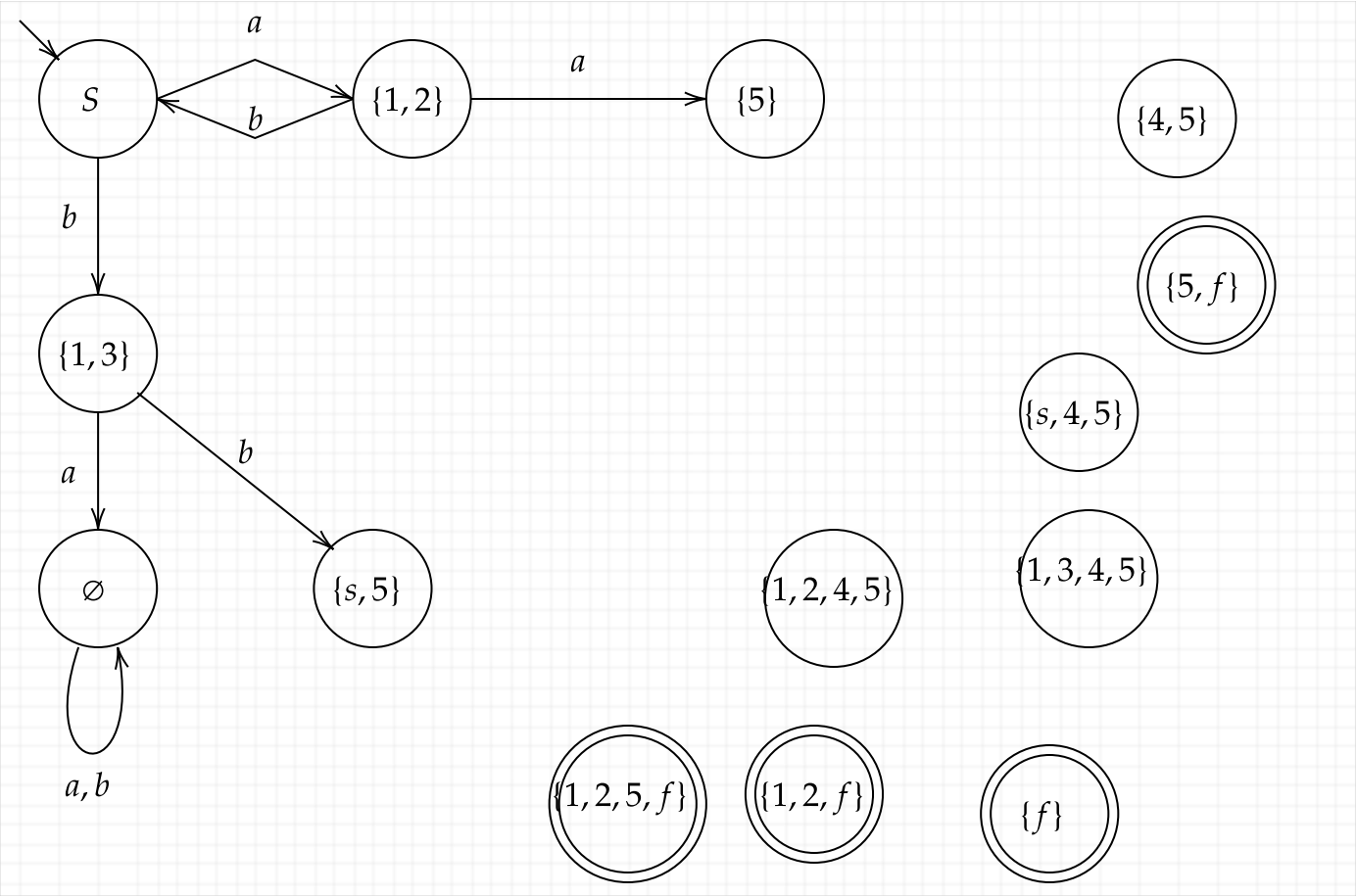


δ	a	b
s	$\{1,2\}$	$\{1,3\}$
1	$\varnothing$	$\{s\}$
2	$\{5\}$	$\varnothing$
3	$\varnothing$	$\{5\}$
4	$\{5\}$	$\varnothing$
5	$\{f\}$	$\{4,5\}$
f	$\varnothing$	$\varnothing$

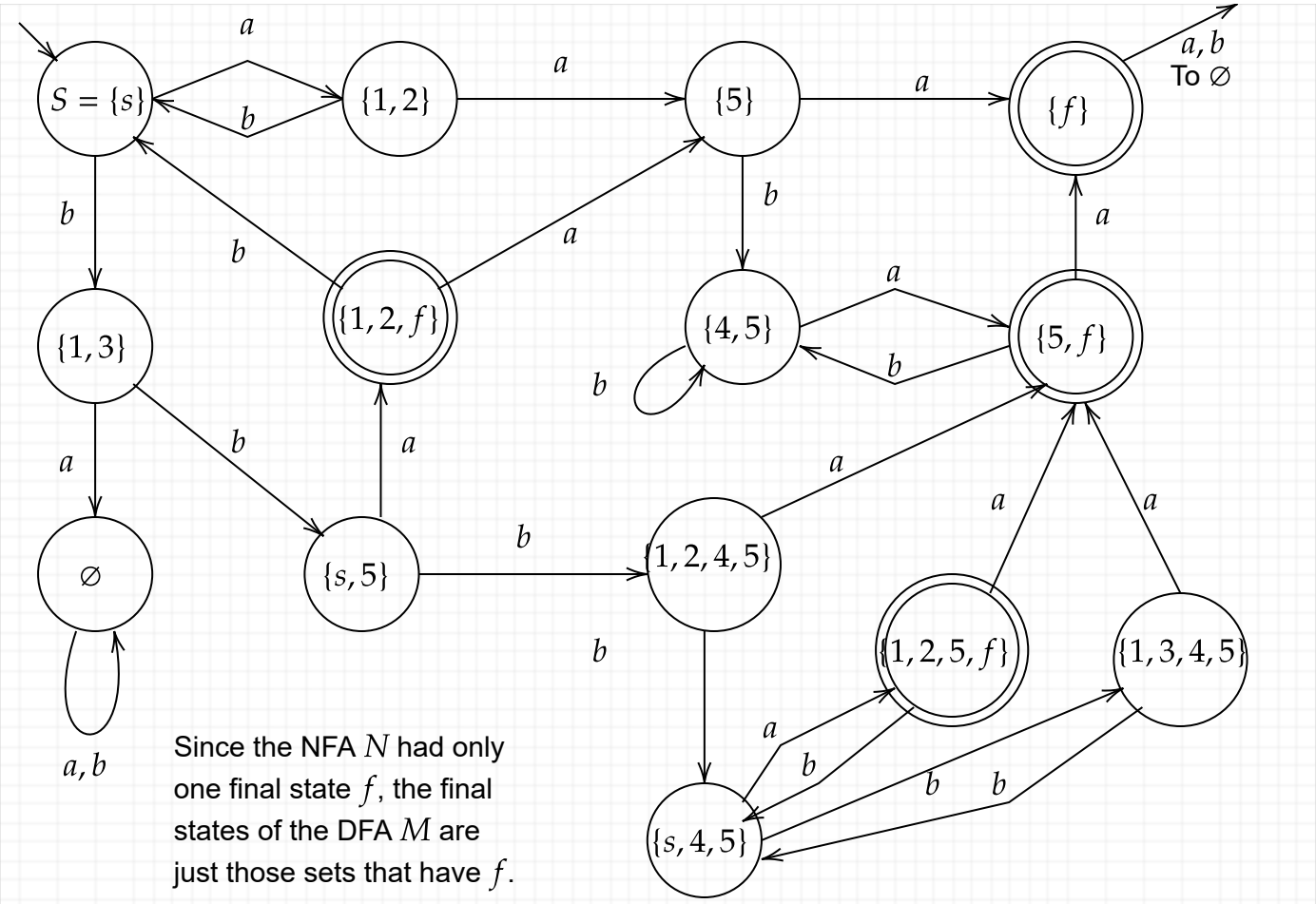
Since there are no ϵ's out of the start state (or at all), S is just $\{s\}$.
$\Delta(S, a) = \{1, 2\}$ (new set-state)
$\Delta(S, b) = \{1, 3\}$ (new set-state). Expand $\{1, 2\}$ first:
$\Delta(\{1, 2\}, a) = \delta(1, a) \cup \delta(2, a) = \varnothing \cup \{5\} = \{5\}$ (new set-state, append to expansion queue)
$\Delta(\{1, 2\}, b) = \delta(1, b) \cup \delta(2, b) = \{s\} \cup \varnothing = \{s\}$. Not new---back to the start stateof 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.
$\Delta(\{s, 5\}, a) = \{1, 2\} \cup \{f\} = \{1, 2, f\}$. New again---this might worry us about blowup.
$\Delta(\{s, 5\}, b) = \{1, 3\} \cup \{4, 5\} = \{1, 2, 4, 5\}$. 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.
$\Delta(\{1, 2, 4, 5\}, a) =$ eyeball rows 1,2,4,5 in column a of the table $= \{5, f\}$. Almost home...
$\Delta(\{1, 2, 4, 5\}, b) =$ 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 ÷(
$\Delta(\{s, 4, 5\}, b) = \{1, 3\} \cup \varnothing \cup \{4, 5\} = \{1, 3, 4, 5\}$. New but not unexpected. Keep Going...
$\Delta(\{1, 2, 5, f\}, a) = \{5, f\}$. Not new.
$\Delta(\{1, 2, 5, f\}, b) = \{s, 4, 5\}$. Also not new.
$\Delta(\{1, 3, 4, 5\}, a) = \{5, f\}$. The same.
$\Delta(\{1, 3, 4, 5\}, b) = \{s\} \cup \{5\} \cup \varnothing \cup \{4, 5\} = \{s, 4, 5\}$. 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 [lecture will piece more together]:



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

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 $x_1 \cdots x_i$ to. The core of the proof is finally to observe that:

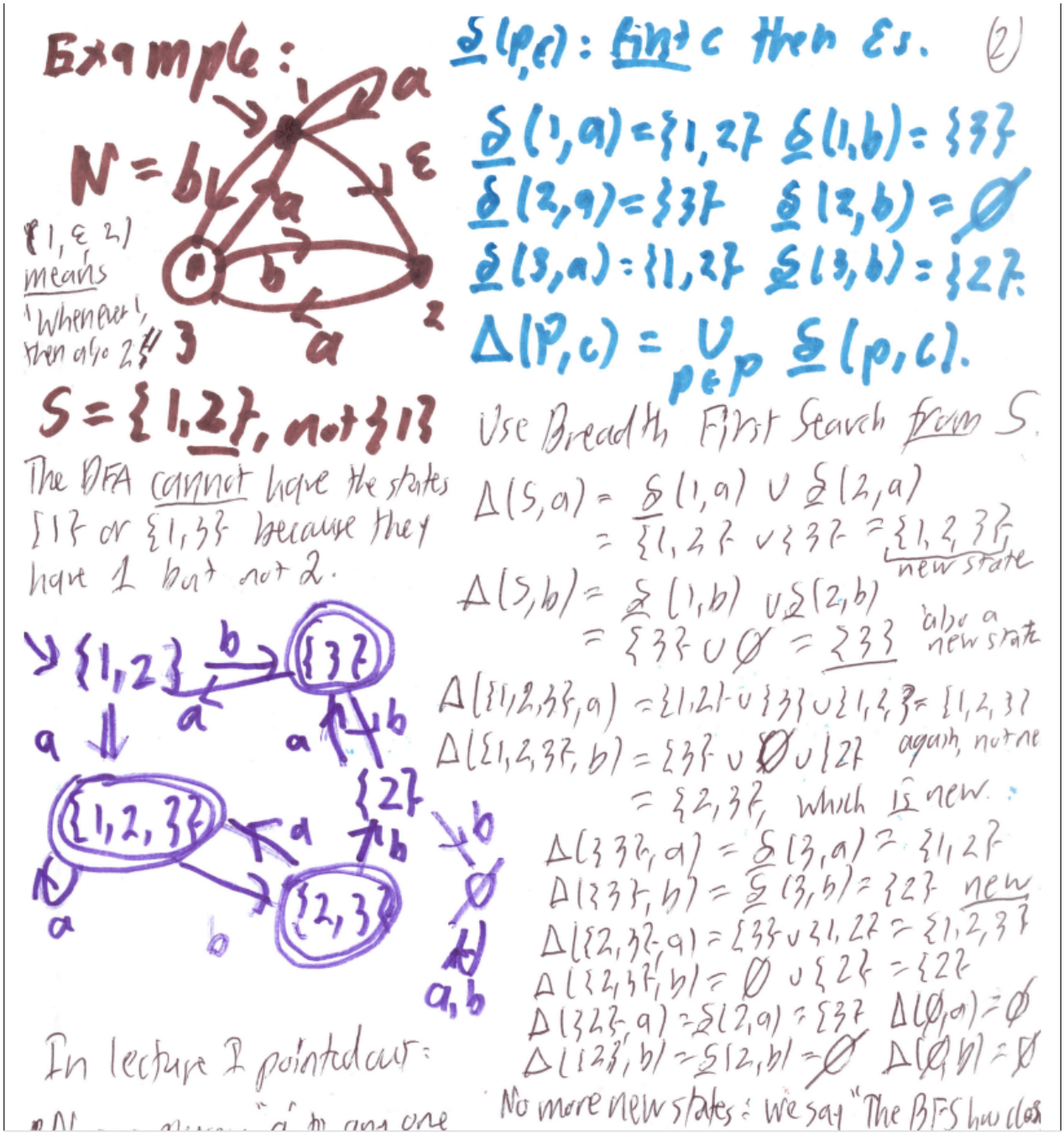
> N can process $x_1 x_2 \cdots x_{i-1} x_i$ *from s to r* if and only if there is a state p such that N can process $x_1 x_2 \cdots x_{i-1}$ from s to p (which by IH $G(i-1)$ includes p into R_{i-1}) and such that N can process the char x_i from p to r.

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. ☒

[More examples as-and-if time allows]



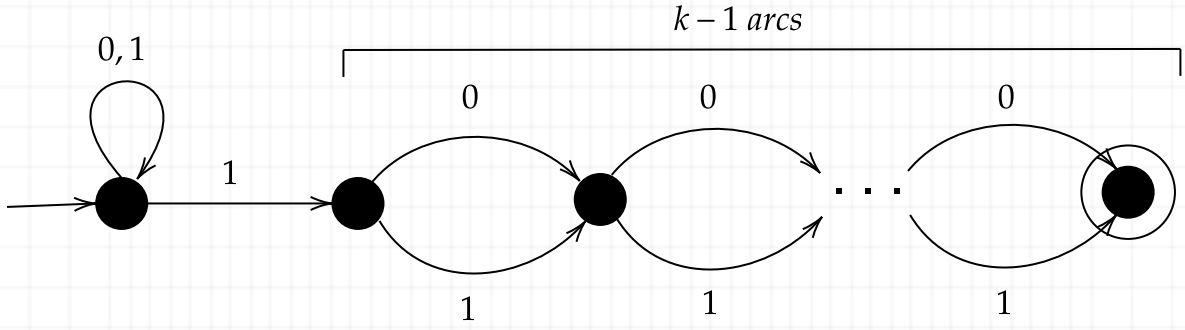
More on how the states of the DFA tell what the NFA can and cannot process:

- 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: The "Leap of Faith" NFAs N_k for any $k > 1$:



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

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.