Craig Silverstein
Gates 470
Phone: (72)3-1787
Office Hours: Wed 4:30–5:30, Fri 1–3, Gates 470
Section: Monday 2:15–3:15, 380-380W

Computer Science 154
School of Engineering
Stanford University

Section Handout #2

DFAs, NFAs, and REs: conversions

We finally know how to convert back and forth from all of these to all the others of these:

From ↓ **To** →	DFA	NFA	RE
DFA	—	nothing	delete states, building RE along transitions.
NFA	subset construction.	—	delete states, building RE along transitions.
RE	build NFA recursively from rules for $*$, $\cup$, and concatenation, then convert to DFA.	build recursively from rules for $*$, $\cup$, and concatenation.	—

- **Subset construction** We replace the nodes of the NFA, which used to be states, with *sets* of states. So if our NFA had states that looked like 1, 2, 3, etc., our DFA will have states that look like $\{1\}$, $\{2\}$, $\{1,2,3\}$, $\emptyset$, etc. The interpretation of this is that we are in state $\{1,2\}$, for instance, of our NFA, if from the states we used to be in, we could transition to either state 1 or state 2 (using some symbol from the alphabet). Remember that for a DFA, all states must have transitions coming out of them. Our construction always starts with the states $\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$, etc. which represent states 1, 2, 3, 4, etc of our NFA. But as we get new sets (such as $\{1,2,3\}$), we have to add them to the DFA as new states. Since we have these new states, we need to figure out transitions for them. Only when we have figured out all the transitions and gotten no new states (don't forget the empty state!) can we declare ourselves finished.

- **RE building** We build regular expressions along the edges, and delete states until only the start and end state are left. For each state we delete, we need to figure out all the transitions that travel through our state. So if we are deleting state 2, we need to look at all transitions that go from 1 to 3 via 2, or 4 to 5 via 2, or even 1 back to itself via 2. We replace each of these roundabout transitions with direct transitions (i.e. replace the arrows from 1 to 2 to 3 with an arrow directly from 1 to 3) and put an RE along the new edge which looks like (RE-from-1-to-2) concatenated with (RE-from-2-to-itself)* concatenated with (RE-from-2-to-3). This is the RE we put along the new edge from 1 to 3. If there was already an edge from 1 to 3, then now there are two, so we combine the two edges to get a single new edge with RE (1st-RE-from-1-to-3) $\cup$ (2nd-RE-from-1-to-3).

- **FA building** This is the opposite of RE building. Building up FAs from the RE parts (kleene star, union, concatenation) involves drawing transitions out of all the final states — or into the start state — of the machine you're messing with. For instance, to concatenate two machines, you take all the final states of one and add arrows from each of those to the starting state of the second machine. We use a lot of ϵ transitions, which is why we defined NFAs to allow them.

Regular Language Summary

(Don't worry if you don't understand all of this; some things, like homomorphisms, we never studied in class. Others, like the pumping theorem, we haven't gotten to yet but will Very Soon Now.)

The following are true if a language is regular:

- There exists a DFA or NFA that accepts the language;
- There exists a regular expression that accepts the language;
- The language is closed under union, concatenation, intersection, kleene star, and reversal (that is, the union of two regular languages is regular, etc);
- The language is closed under homomorphism;
- The language is closed under taking suffixes (for every string in a regular language, take the last n characters and put it in a new set. This new set is a regular language);
- The language satisfies the pumping theorem.

The inverses are not, in general true: if a language satisfies the pumping theorem, it is *not* necessarily regular.

To prove a language is regular:

- Give a DFA or an NFA for it;
- Give a regular expression for it;
- Show it can be constructed from other regular languages using union, concatenation, intersection, homomorphism, reversal, suffixes, etc.

To prove a language is not regular:

- Use the strong form of the pumping theorem: pick a string in the language (you pick), know that it can be divided into 3 parts such that the size of the first two parts combined is less than some constant K (you don't pick, but pretend you're given K before you pick your string), and show that if you stick in extra copies of the second part you get a string that is not in the language.
- Simplify the problem by doing one of the following to a language known to be regular: intersection, union, homomorphisming, etc. If you get a non-regular language, then your original language was also non-regular (e.g., if $L_1 \cap L_2$ is not regular, then either L_1 or L_2 is not regular — but, of course, we picked L_2 to be a known regular language, so L_1 is the not-regular guy). The idea is do this so the result language is simpler than the first language. **Example:** $a^n b^n c^*$ is not regular because $a^n b^n c^* \cap a^* b^* = a^n b^n$ is not regular.

The following argument is *not* valid: "My language is not regular because a subset of it is not regular." For instance "$a^n b^n c^*$ is not regular because it includes $a^n b^n$ as a subset." Consider: $a^* b^*$ includes $a^n b^n$ as a subset, but $a^* b^*$ is certainly regular!