

Supplementary Notes from the Tutorials for CS2MJ3, Fall 2009

Nick James

October 1, 2009

Tutorial 1

First, some notation. It seems that not everyone completely grasped the $*$ and $+$ notation as it applies to sets of strings or symbols. If X is a set of strings or symbols then,

$$\begin{aligned}X^0 &= \{\varepsilon\} \\X^{k+1} &= X^k X \quad \text{for } k \geq 0 \\X^* &= \bigcup_{i=0}^{\infty} X^i \\X^+ &= \bigcup_{i=1}^{\infty} X^i\end{aligned}$$

For example,

$$\{a, b\}^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, \dots\}$$

$$\{a, b\}^+ = \{a, b, aa, ab, ba, bb, aaa, aab, aba, \dots\}$$

With that out of the way, we can move on to something more interesting. During the first tutorial many people asked the following question:

If I have a DFA (or NFA), $M = (X, Q, q_0, \delta, F)$, how can I systematically produce a NFA, $N' = (X, Q', q'_0, \delta', F')$ such that $L(N') = L(M)^*$?

This question is actually deeper than it seems. To answer it is to effectively prove (depending on how pedantic you're feeling) that if a language, A , is regular, then so is A^* . Let us begin, however, by considering how we might construct an, N , such that $L(N) = L(M)^+$, for that task is somewhat simpler.

M accepts the string, $x \in X^*$ if and only if M is left in some final state, $q \in F$ after reading the last symbol of x . So if we wish our machine, N , to accept any nonempty concatenation of strings accepted by M , all we need to do is start with $N = M$ and add ε -transitions from every final state of N to the start state (of N). Once the machine finishes reading x , it can start over from the beginning using one of those fancy, new ε -transitions and read another copy

of x (or any other string in $L(M)$). It can keep going back like this as long as there remain strings from $L(M)$ in the concatenation. Hence, our modified machine, N , recognizes the language $L(N) = L(M)^+$. That's fairly clear, I think, just from this informal explanation, but to prove it rigorously would require an inductive proof (this is what I meant by, "depending on how pedantic you're feeling"). I'll do one if you'd like to see it, but I warn you that it will require the development of some notation that the course text lamentably neglects.

There are two ways we can construct a machine, N' , which recognizes the language, $L(N') = L(M)^*$. One way is to work from the idea that the only difference between N and N' is that N' must also accept ε (if it doesn't already). How can we modify a machine to make it behave exactly as it normally does but also accept ε ? One way is to add a new final state which will also serve as the new start state, q'_0 , and add an ε -transition from q'_0 to the former start state, q_0 . The machine then begins in a final state, so if the input is ε , it will accept it. If the input is not ε , it must proceed along the new ε -transition to the former start state (at which point it acts just like the original machine). There's no way to return to q'_0 once we leave it, so the only effect it has is to make the machine accept ε regardless of whether it did so already.

The second way to construct N' is to implement the following four changes to M :

- Add a new state, q'_0 , which will serve as the new start state (the old start state, q_0 , stays in the machine but is "demoted" to the status of a normal (non-start) state. This step is exactly what we did before with q'_0).
- Add an ε -transition from q'_0 to the old start state, q_0 (again, the same as before).
- Add ε -transitions from every final state to q'_0 (in the previous method, the ε -transitions went to q_0 instead of q'_0).
- Finally, make q'_0 the *only* final state in the machine (before, q'_0 merely became an additional final state).

At first glance, the addition of q'_0 seems unnecessary. Why not just add the ε -transitions from the final states to q_0 and then make q_0 the only final state? Two steps. Should work, right?

In many cases it really does work, but there are some exceptions for which it fails. The four-step procedure, however, never fails. To see an counterexample, consider the following NFA¹:

¹I was planning to use a diagram for this instead of a table, but I've never found any software I really like for drawing NFAs and DFAs. Lots of software *works*, but it all seems so much more awkward than it should be. I thought surely tonight was the night I'd finally find the perfect finite automaton drawing software, but two hours of searching and experimentation didn't turn up anything better L^AT_EX's "picture environment" (which is supremely unwieldy, but very pretty), Open Office's "Draw" (which is easier, but still clunky and hard to make it look quite right), and Xfig (which I think ends up being worse than "Draw"). So... Sorry. No time for pictures today. I'm hoping Prof. Soltys has covered state-transition tables. Otherwise this table is going to look like nonsense. It's not terribly important, though. Just a counterexample to illustrate the fact that the 2-step technique doesn't always work. You can just take my word for it and go do something more interesting instead. ;-)

| State | Symbol | $\delta(\text{State}, \text{Symbol})$ |
|-------------------|--------|---------------------------------------|
| $\rightarrow q_0$ | a | $\{q_1\}$ |
| $q_1 F$ | b | $\{q_0\}$ |

This machine recognizes the language, $L(a(ba)^*)$, so we want to make a new machine that recognizes $L((a(ba)^*)^*)$. The flawed, two-step method produces this:

| State | Symbol | $\delta(\text{State}, \text{Symbol})$ |
|---------------------|---------------|---------------------------------------|
| $\rightarrow q_0 F$ | a | $\{q_1\}$ |
| q_1 | b | $\{q_0\}$ |
| q_1 | ε | $\{q_0\}$ |

This machine recognizes the language, $L((a + ab)^*)$, which isn't quite what we wanted.

The very first method I discussed (the one based directly on the machine for $L(M)^+$) produces this:

| State | Symbol | $\delta(\text{State}, \text{Symbol})$ |
|----------------------|---------------|---------------------------------------|
| $\rightarrow q'_0 F$ | ε | $\{q_0\}$ |
| q_0 | a | $\{q_1\}$ |
| $q_1 F$ | b | $\{q_0\}$ |
| $q_1 F$ | ε | $\{q_0\}$ |

The four-step method produces this:

| State | Symbol | $\delta(\text{State}, \text{Symbol})$ |
|----------------------|---------------|---------------------------------------|
| $\rightarrow q'_0 F$ | ε | $\{q_0\}$ |
| q_0 | a | $\{q_1\}$ |
| q_1 | b | $\{q_0\}$ |
| q_1 | ε | $\{q'_0\}$ |

The latter two machines recognize the language we wanted: $L((a(ba)^*)^*)$.

These tables actually look needlessly complicated in retrospect. I guess I'll just suck it up and pick *some* kind of diagram software to make pictures for these things... But not today (I'm tired).

Please mail me (jamesnd2@mcmaster.ca) if you have questions or comments about this.