Lecture 8: Midterm Review

Ryan Bernstein

1 Introductory Remarks

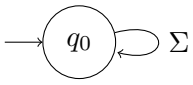
- As I'm sure you're aware, the midterm is on Thursday. Assignment 2 is due that same day.

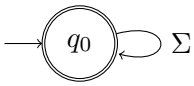
2 Remarks on the Assignment 1

2.1 Drawing Finite State Machines

2.1.1 $\emptyset$ and Σ^*

There seems to have been some confusion about the difference between the following two DFAs:





The DFA on the left has no accepting states. This means that it rejects all strings, and its language is therefore the empty language $\emptyset$. The DFA on the right, on the other hand, has *only* accepting states, which means that it *accepts* all strings. Its language is therefore Σ^*.

On the question about finite languages, I had a surprising number of people attempt to prove that $\emptyset$ was a finite language by drawing something akin to the machine on the right.

2.1.2 Using Σ-transitions

Σ-transitions—like those shown in the DFAs above—are a useful tool. They can make drawing finite automata easier, and they can also make those automata *alphabet-agnostic*. This is useful for languages like $\emptyset$ or Σ^*, because these languages behave the same way regardless of the alphabet used in their elements.

There are some times, though, when it's not appropriate to use Σ. Σ-transitions are a shorthand that we use to say that we follow this same transition for every character in the alphabet. This means that it's only really appropriate to use Σ-transitions in two cases:

1. We're trying to make a language alphabet-agnostic, in which case we know nothing about the contents of Σ. Since this means we need to follow the same transition on every character, this is usually only appropriate when we're:

- Drawing an automaton for a trivial language like $\emptyset$ or Σ^*, or
- Drawing a DFA for some language that is dependent on string length alone.

2. We actually want to take the same transition for every character in Σ, but this may not be the case for other transitions in the state machine. In this case, we must know *everything* about Σ.

For any language that is *not* alphabet-agnostic, you should not be using a Σ-transition unless either you or the question have fully specified your alphabet.

2.2 Things We Cannot Do

An NFA is a 5-tuple $(Q, \Sigma, \delta, q_{start}, F)$. Nowhere in this do we allow for logical gates, despite how convenient they might make our lives. If we want to achieve a similar effect, we need to simulate this using only states.

3 How to Proof

3.1 Constructive Proofs

If you're asked to show that some language is regular, I can think of exactly two ways to do this (although there may be more that I haven't considered):

1. Construct a finite automaton that decides it, or
2. Construct a regular expression that describes it

Similarly, if you're asked to show that some language is context-free, you can either:

1. Construct a pushdown automaton that decides it, or
2. Construct a context-free grammar that describes it

It is *not* enough to simply state that such a construction exists. By definition, a language is regular if there exists some DFA that decides it. Stating "There exists a DFA that decides L" is simply restating the fact that you are trying to prove.

You *must* show how such a machine can be built. This is the origin of the name "Proof by construction".

If I'm trying to show that context-free languages are closed under concatenation, I'm allowed to instantiate context-free grammars for my input languages A and B, since I have been told (or am able to safely assume) that they are context-free. I am *not* allowed to simply say "Since a context-free grammar exists that describes $A \circ B$ exists, context-free languages are closed under concatenation." This is the conclusion of my proof, but I have to show that I can construct that grammar by adding a new start variable S_0 and a new rule $S_0 \to S_A S_B$ to prove that this conclusion is valid.

3.2 Disproving Closure Properties

If we're attempting to show that some class of languages is *not* closed under some operation, we need only find a single counterexample.

Closure under an operation means that performing that operation on *any* two members of the set yields a result that is also a member of that set. If we want to disprove this, we only need a single pair of members for which this does not work. For instance, if we wanted to prove that the natural numbers are not closed under subtraction, it is sufficient to point out that $3 - 5 = -2$.

3.3 Applying the Pumping Lemmas

We can restate the regular language pumping lemma as follows:

> If a language A is regular, then **there exists** some length p such that **for any** string $s \in A$ with a length of at least p, **there exists** some x, **there exists** some y, and **there exists** some z such that $s = xyz$ and the following requirements are satisfied:
>
> 1. $xy^iz \in A$ **for any** $i \geq 0$
> 2. $|y| > 0$
> 3. $|xz| \leq p$

If we let P be some powerful predicate relation that encompasses the requirements of the pumping lemma, we can quantify all of this as follows:

$$\exists p \forall s \exists x \exists y \exists z \forall i P p s x y z i$$

Since we're showing that the pumping lemma *does not hold* for some non-regular language, we want to negate this:

$$\neg \exists p \forall s \exists x \exists y \exists z \forall i P p s x y z i$$

Moving the negation inside these quantifiers and attaching it to the pumping lemma P itself yields the following:

$$\forall p \exists s \forall x \forall y \forall z \exists i \neg P p s x y z i$$

If we want to show that this is true, we are allowed to pick any variable preceded by an existential quantifier. We have to show that the pumping lemma breaks for any value of the universally-quantified variables.

Fortunately, because of the relationship between s, x, y, and z, we are able to restrict the possible values of these universally-quantified variables. But we are *never* allowed to explicitly choose a value for x, y, or z.