Takeaway Notes: Closure and Nondeterminism

Contents

1 Introduction

This writeup is meant to provide key takeaways regarding language derivations that are *closed* over the set of regular languages (i.e., the languages that finite state machines can recognize)—thus the inclusion of "closure" in the title. Said closure is accomplished primarily via construction of NFAs from a base DFA or NFA, and thus this serves as a further exploration of nondeterministic finite state automata.

This is largely a distillation of the textbook and online video material, but seen through my individual lens in terms of emphasis and approach. Think of the textbook, the videos, this writeup, and the class meetings as if they were four portholes into the same subject. By exposing yourself to the same material in four different ways, the hope is that you walk away from this in a way that fulfills the objectives of the course.

2 Key Concepts

Here, we will start using the name "regular language" to denote a certain category of languages. This category is the set of all languages that finite state machines can recognize. For now it's just a name; we'll eventually dig into it to say exactly what it is about them that makes them "regular."

The notion of *closure* comes from set theory and abstract algebra, describing how some given set of entities can produce other entities strictly within the same set. Numbers beget numbers, strings beget strings, and here, languages of a certain category (*regular* languages) beget other languages within that category.

The following transformations/variations are said to be *closed* within the set of regular languages:

- The reverse of a regular language
- The complement of a regular language
- The union of regular languages
- The intersection of regular languages
- The concatenation of regular languages

In other words, the reverse or complement of a regular language L is also a regular language. The union, intersection, or concatenation of multiple regular languages $L_1 \dots L_n$ is also a regular language.

2.1 Proof Approach

The process for proving these closures remains along the same lines as the approach for showing that NFAs can handle the same languages as DFAs: show that there is a DFA that recognizes the said transformation/variation of the language. The existence of such a DFA shows that this transformation/variation is also regular, thus upholding closure.

As it turns out, there *is* an algorithm that constructs a finite state machine for each of these languages, given that the DFA for the original language(s) L or $L_1 \dots L_n$ has been defined.

2.2 "ϵ Moves"

Notably, some of the above algorithms don't produce a DFA directly, but instead produce an NFA. But that's OK, because we already know that a DFA can always be constructed from an NFA.

However, some of these constructions require one last tweak to our definition of an NFA: we now need so-called ϵ *moves*. ϵ moves are state transitions *on the empty string*: that means a machine can change state without consuming a character from the input string.

The good news is that ϵ moves aren't that hard to add to our formal definition. Since this involves transitions, the change affects δ: for NFAs that have ϵ moves (i.e., ϵ-NFAs), we can now say that δ may accept ϵ for its second argument (the one traditionally reserved for a character in Σ). Non-ϵ-NFAs are easily represented by saying that $\delta(q, \epsilon) = \emptyset$.

A clear-cut motivating example for ϵ moves is when constructing a finite state machine that involves switching start and accepting states (e.g., reversing a language). There can be only one start state, but any given finite state machine may have multiple accepting states. These multiple accepting states are easily converted into a single start state by designating this start state but giving it multiple ϵ state changes to the original accepting states. Note also that this swap holds true whether you are starting with a DFA or NFA. In both cases, you end up with an ϵ-NFA.

3 Important Skills

What should one be able to *do* with these concepts? Here are some key skillz:

- Work out how to construct a new NFA that recognizes a language that is derivable from an original language L—this knowledge will fade over time (unless you become a theoretical computer scientist!), so let's just say that over the long haul, you just don't forget that there's *some* process for deriving an NFA from another state machine which corresponds to deriving some regular language from an original one
- Apply ϵ moves correctly, whether across a single or multiple transitions
- Construct a DFA from an ϵ-NFA (which mainly means that you can do the above item correctly, over and over again)

Note that these correspond to what you may be asked to submit as course work for this general topic.

4 Big Picture Points

Things you shouldn't forget even years after taking this course:

- Many operations that "convert" one or more regular languages to another are *closed* over the set of regular languages—i.e., such operations themselves produce regular languages.
- In the most general sense, NFAs can be defined which can change state *without* consuming an input character. As with the original expansion into NFAs, this adds convenience but no additional power to finite state automata in general.
- Yet again, we define "power" here as the ability to recognize a language that cannot be recognized by an earlier, "lesser" machine. So far, we have not yet added a feature to our machines which constitutes this upgrade in power.

We will hold off on the extra power a little bit more, as we explore an idea that lives in parallel to finite state automata: that of *regular languages*.