COSC341 TUTORIAL 5, SOLUTIONS

Checking to make sure we understand how to convert between all the different versions of regular languages and a bit more on closure properties. Unless otherwise specified use the alphabet $\Sigma = \{a, b\}$.

1. *A closure property for the regular languages is a construction that applies to a regular language (or languages) and is guaranteed to produce another regular language. For example "the union of two regular languages is regular". What closure properties have we mentioned so far? What others can you think of?*

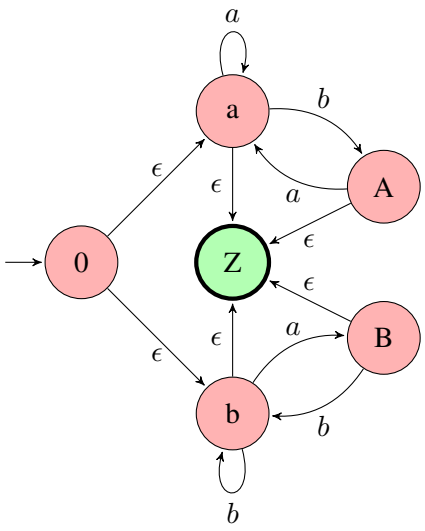
   The regular languages are closed under all of:

   - union,
   - intersection,
   - complement,
   - Kleene-star,
   - concatenation,
   - substitution (meaning, take a regular language and allow the substitution of any words from other regular languages for its letters, e.g., replace a's by words from ab^* and b's by words not containing consecutive a's),
   - subwords (any word that can be produced by deleting any or all letters),
   - prefixes,
   - suffixes,
   - reversal.

2. *You told a friend about the result that the language of strings of a's whose length is a power of two is not regular. They said "I wonder whether there is a regular language of strings from a 17-letter alphabet where the length of every string in the language is a power of two, and all powers of two occur?" What do you think the answer to their question is? More importantly, can you prove it?*

   Using the result on subsitution above – if this language were regular so would be the language where we just substituted a for every letter (or in terms of the underlying automaton, replaced all label arrows with a). That would make the original language regular and it's not. Therefore, there can be no such language.

3. *Design an NFA for the language of strings that contain either no pair of consecutive b's or no pair of consecutive a's. Convert to a DFA.*

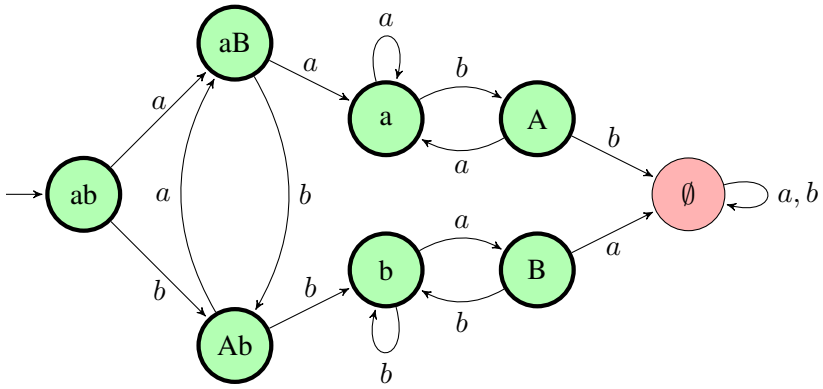


The intention of the labels is that in lower-case labelled states we're happy to take either letter (but happiest with the label) while in upper-case labelled states we *only* want the corresponding letter.

To convert this to a DFA we note for convenience that the ϵ closure of 0 is 0abZ and that Z is in the ϵ-closure of all states except the garbage state , meaning that all the non-garbage states will be accepting, and we can discount Z as a "place we might be" (so long as we're still somewhere, i.e., have not seen both aa and bb). We can also ignore 0 since we can't get back there.

So, our initial state is ab, which has an a transition to aB and a b transition to Ab. From aB, a takes us to a, and b takes us to Ab. Similarly, from Ab, b takes us to b, and a takes us to aB. From a, a loops and b takes us to A, while from b, b loops and a takes us to B. Finally, from A, b takes us to the empty set (i.e., garbage) and a takes us to a, while from B, a takes us to garbage and b takes us to b.

In a picture:



4. *Tutorial 4 asked you to develop a DFA that accepts the language* EVEN-EVEN*, which consists of all strings containing both an even number of a's and an even number of b's. ... Now design an NFA in standard form that accepts this language and then use the state elimination technique to derive a regular expression for it. Compare your result to the one presented (without use of the state elimination technique) in the solution to Tutorial 4, Question 4. Was this surprising?*

   The following regular expression was presented in the solution to Tutorial 4, Question 4:

$$(aa + bb + (ab + ba)\,(aa + bb)^*\,(ab + ba))^*$$

   This was generated by thinking about paths through the four state automaton, and specifically thinking about the fact that EVEN-EVEN is closed under concatenation so we can just consider the minimal prefix of a word in EVEN-EVEN that also belongs to EVEN-EVEN. Two such minimal prefixes are aa and bb. If we start ab or ba (reaching a point where we have an odd number of both symbols) then we we can bounce back and forth between the odd-odd state and the odd-even or even-odd states (via aa or bb) but, at some point must return to even-even via a second ab or ba).

   As the regular expressions generated by state elimination can be much longer than those designed carefully using human intuition, it is somewhat surprising to find that the state elimination algorithm reproduces the regular expression above, at least when eliminating states 01 and 10 before 11—elimating 11 first has not been tried.

5. *Recall the language* MULTI-6-b *(introduced in Tutorial 4), which consists of all strings with the following property: for some a in the string, the number*

of b's that follow it is a multiple of 6. We saw an NFA with seven states that recognises it (9 states in "standard form"). Convince yourself that to convert it to DFA will result in something like 64 states ("off by several" errors are possible here) and that this is somehow necessary.

This foreshadowed the Myhill-Nerode theorem. The idea is the following – for any subset, X, of $\{0, 1, 2, 3, 4, 5\}$ we can easily construct a string where for each and only $x \in X$ there is an a in the string such that the number of b's following it is x. For instance with $X = \{0, 3, 5\}$ we could use $abbabbba$. The powers of b that can be appended to such a string in order to belong to MULTI-6-b determine the set X (basically, k b's can be appended if and only if $6 - k$ is in X). So, two such words corresponding to different sets must correspond to different states of the DFA (else they'd allow the same continuations). Since there are 64 such sets, the DFA must have at least 64 states.