Johns Hopkins Math Tournament 2018

Proof Round: Automata

February 9, 2019

Problem	Points	Score
1	10	
2	5	
3	10	
4	20	
5	20	
6	15	
7	20	
Total	100	

Instructions

- The exam is worth 100 points; each part's point value is given in brackets next to the part.
- To receive full credit, the presentation must be legible, orderly, clear, and concise.
- If a problem says "list" or "compute," you need not justify your answer. If a problem says "determine," "find," or "show," then you must show your work or explain your reasoning to receive full credit, although such explanations do not have to be lengthy. If a problem says justify or prove, then you must prove your answer rigorously.
- Even if not proved, earlier numbered items may be used in solutions to later numbered items, but not vice versa. There is an exception for problems in two parts (for example 5a and 5b). You will not receive any credit for part b if your proof for part a is not correct.
- Pages submitted for credit should be **numbered in consecutive order at the top of each page** in what your team considers to be proper sequential order.
- **Please write on only one side of the answer papers.**
- Put the **team number** (NOT the team name) on the cover sheet used as the first page of the papers submitted. Do not identify the team in any other way.

Introduction

In this proof round, you will explore the mathematics underlying **finite automata**. These are simple constructs used by computer scientists to understand how computers work. While computers in real life are very complicated machines, their fundamental operations can be reduced to a small set, which enables us to understand programs and their algorithms from a theoretical point of view.

1 Regular Languages

Amanda has a very boring job at Acme, Inc.. Every day, from 9 AM to 5 PM, Amanda sits in front of a conveyor belt which carries black and white balls. Amanda's supervisor occasionally barges in, and hands her a description of a pattern of colors. A few minutes later, the conveyor belt starts up, and the balls start moving by. When the conveyor belt stops, Amanda has to decide whether the balls matched the pattern her supervisor gave her.

Definition 1.1. We call a subset of the set of all possible sequences of balls a language. A regular expression is a simple way of specifying a language; the set of patterns specified by the expression r is called the language of r and is denoted $L(r)$. If a language is specified by some regular expression, we call it a regular language.

To define regular languages, we start with an alphabet of symbols, which we will take to be $\{B, W\}$, since the conveyor belt carries black and white balls. Additionally, we use ϵ to denote the empty sequence. If $r = \alpha$, where α is in the alphabet, then $L(r) = \{\alpha\}$. We can combine regular languages recursively using the following operations on regular expressions:

- *Concatenation*: the language $L(rs)$ contains all patterns which consist of a pattern in $L(r)$ followed by a pattern in $L(s)$
- *Disjunction*: the language $L(r \mid s) = L(r) \cup L(s)$
- *Kleene star*: the language $L(r^\star) = \{\epsilon\} \cup \{s_1 \mid_1 \in L(r)\} \cup \{s_1 s_2 \mid s_1, s_2 \in L(r)\} \cup \cdots$, that is, all strings which can be formed by repeatedly taking elements of $L(r)$ and concatenating them

In the order of operations, Kleene star has priority over concatenation, which in turn has priority over disjunction. Additionally, we use parentheses to group symbols and expressions.

Example 1.1. *The regular expression $B^\star$ accepts any pattern which contains only black balls. The regular expression $(WW \mid B)^\star$ accepts any pattern for which consecutive white balls appear in even numbers.*

Problem 1:
(a) (2 points) Show that all finite languages are regular.
(b) (3 points) Describe the language specified by the regular expression

$$B^\star(B^\star \mid WW^\star)^\star WW^\star$$

(c) (5 points) Let r be a regular expression not containing any instances of ϵ. Show that a regular language $L(r)$ is infinite (as a set) if and only if r contains a Kleene star.

(a) Let L be finite, i.e., $L = \{s_1, \ldots, s_n\}$. Then $L = L(r)$, where

$$r = s_1 \mid \cdots \mid s_n.$$

This shows that L is regular.

(b) Since $L(B^\star) \subset L(B^\star \mid WW^\star)$, we can simplify by removing the concatenation with $B^\star$. Additionally, it is clear that

$$L\left((B^\star \mid WW^\star)^\star\right) = L\left((B \mid W)^\star\right).$$

Therefore, we are seeking to describe the language specified by $(B \mid W)^\star WW^\star$. But $L((B \mid W)^\star)$ is the largest possible language, containing all strings; so the language we are seeking contains all strings which end in W. (Full points should be given for this result, regardless of methods used).

(c) We first show, in the absence of the empty string ϵ, how each operation changes the size of the corresponding language. First, if $L(r)$ is already infinite, then $L(rs)$, $L(r \mid s)$, and $L(r^\star)$ are all infinite. If $|L(r)| < \infty$ and $|L(s)| < \infty$, then:

- $|L(rs)| = |L(r)| \cdot |L(s)| \geq |L(r)|$
- $|L(r \mid s)| = |L(r)| + |L(s)| \geq |L(r)|$
- $|L(r^\star)| = \infty$

Given these observations, the result easily follows. If r contains a Kleene star, then at some point in the recursive definition of $L(r)$, the cardinality grows to infinity as per the third point above. Conversely, if $L(r)$ contains only concatenations and disjunctions, then $|L(r)|$ can be written in terms of finite products and sums of the cardinalities of the base languages; but all the base languages (consisting of a single alphabet symbol) have cardinality 1, so $|L(r)|$ is finite.

Amanda has begun to suspect that some of the patterns represent a binary code. The white balls should be read as ones, and the black balls should be read as zeros. Recall that binary numerals are in a base two number system.

Example 1.2. *The string of colors WWBBWB represents the binary number* 110010_2. *This is equivalent to* $32 + 16 + 2 = 50$ *in decimal.*

Problem 2: Some regular languages have a numerical interpretation.

(a) (2 points) Write a regular expression which accepts strings corresponding to numbers divisible by two.

(b) (3 points) Describe how to write a regular expression which accepts strings corresponding to numbers divisible by 2^n for some fixed n.

(a) A binary integer is divisible by two if and only if it ends in a zero. Therefore, we are looking for patterns ending in a black ball. A suitable regular expression is $(B \mid W)^\star B$ (though this choice is not unique, and other equivalent answers should be accepted).

(b) A binary integer is divisible by 2^n if and only if the last n digits are zeroes. A suitable regular expression to describe the corresponding ball patterns is

$$(B \mid W)^\star \underbrace{B \cdots B}_{n \text{ times}}$$

2 Nondeterministic Finite Automata

Amanda is tired of sorting through the ball patterns, so she wants to build a machine that does the job for her. Luckily, Acme is not just a toy ball company; they also have a division working on *automata*. We will start by considering a special class of automata called nondeterministic finite automata, or NFAs. An NFA has a finite set of states, each of which can be active or inactive at any given time. The machine starts with only the designated initial state being active, and then follows rules for updating each state as data comes in.

The rules take the form of one state activating another, conditional on which color ball comes across the conveyor belt. Formally, for each state i, we define three sets:

$$\begin{aligned} S_i^{(\epsilon)} &= \{\text{states } j \text{ for which there is an } \epsilon\text{-channel } i \to j\} \\ S_i^{(B)} &= \{\text{states } j \text{ for which there is a } B \text{ channel } i \to j\} \\ S_i^{(W)} &= \{\text{states } j \text{ for which there is a } W \text{ channel } i \to j\} \end{aligned}$$

If state i is active at time t, then every state in $S_i^{(\epsilon)}$ will be active at time t as well. If a black ball comes across the conveyor belt, then every state in $S_i^{(B)}$ will be active at time $t+1$; and if a white ball comes across, every state in $S_i^{(W)}$ will be active at time $t+1$. Note that state i itself becomes inactive at time $t+1$, unless it is activated (either by itself or another state).

Amanda wants to use an NFA in order to recognize her patterns. As the colored balls come across the conveyor belt, different states will be activated at each time step. One state will be designated as the *accept state*. If the accept state is active when the conveyor belt stops moving, then the pattern is accepted.

Example 2.1. *We will use a simplified diagrammatic notation for NFAs. Figure 1 shows a machine comprised of four states, labeled* $1, 2, 3, 4$*. The arrow pointing at state 1 indicates that it is made to be active when the conveyor belt starts. The double circle around state 4 indicates that it is the accept state.*

We describe the sequence of events for the input pattern $BWWB$ *below.*

- *When the conveyor belt starts, state 1 is activated, and because of the* ϵ*-channel, so is state 3.*

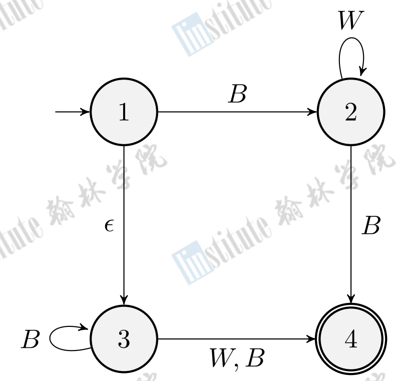


Figure 1: An example NFA.

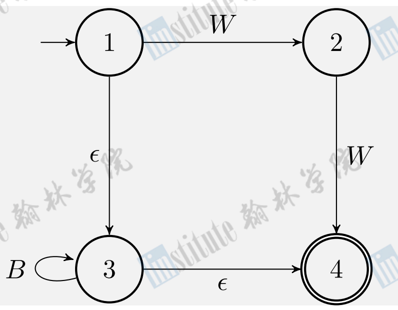
- *The first ball is black, so state 3 activates itself and state 4, while state 1 activates state 2, so now state 2, 3, 4 are active.*
- *The next ball is white, so state 2 activates itself, while state 3 activates state 4, so now states 2 and 4 are active.*
- *The next ball is white, so state 2 activates itself. Now only state 2 is active.*
- *The final ball is black, so state 2 fires a ball at state 4, leaving it active.*

Since state 4 is active when the pattern is finished, the pattern is accepted by the machine.

Problem 3:
(a) (3 points) Write a regular expression which describes the language of patterns accepted by the NFA in Figure 1.
(b) (7 points) Draw an NFA consisting of four states which accepts a pattern if and only if it is in the language described by the regular expression $B^* \mid (WW)$.

(a) There are two paths to activating the accept node. The path going through state 2 requires a black ball, followed by some number of white balls, followed by a black ball; the regular expression for this pattern is $BW^\star B$. The path going through state 3 requires some number of black balls, followed by any ball; the regular expression for this pattern is $B^\star(W \mid B)$. In total, our regular expression is $(BW^\star B) \mid (B^\star(W \mid B))$.

(b) With four states, there is essentially only one way to do this:



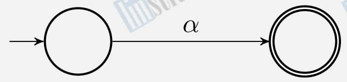
Amanda hopes to use NFAs to automate her entire job. First, she needs to verify that the machines can recognize any regular language. We can prove this using an inductive argument. First, we show that there is an NFA corresponding to any of the simplest regular languages, languages consisting of a single string which contains a single alphabet symbol. We then show that the three regular expression operations – concatenation, disjunction, and Kleene star – can all be represented with NFAs.

Problem 4:

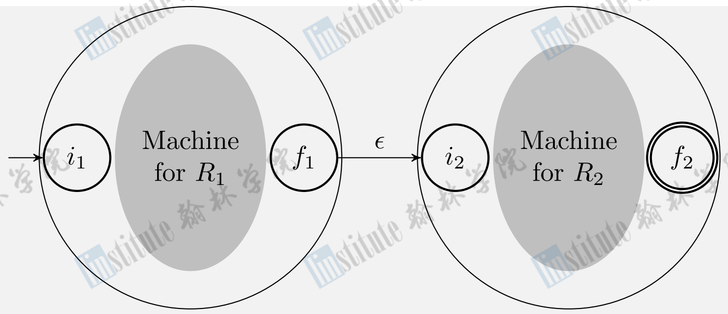
(a) (2 points) Show that regular languages consisting only of a single one-character string can be represented using an NFA.

Assume that two regular languages R_1 and R_2 can both be represented by NFAs.
(b) (4 points) Show that their concatenation R_1R_2 can be represented by an NFA.
(c) (4 points) Show that their disjunction $R_1 \mid R_2$ can be represented by an NFA.
(d) (6 points) Show that the Kleene star $R_1^\star$ can be represented by an NFA.
(e) (4 points) Using your constructions in parts (b)-(d), draw an NFA which accepts only the patterns in the regular language given by $(B \mid WB)^\star \mid (W \mid BW)^\star$.

(a) This is obvious; for every symbol α in the alphabet, we have an NFA

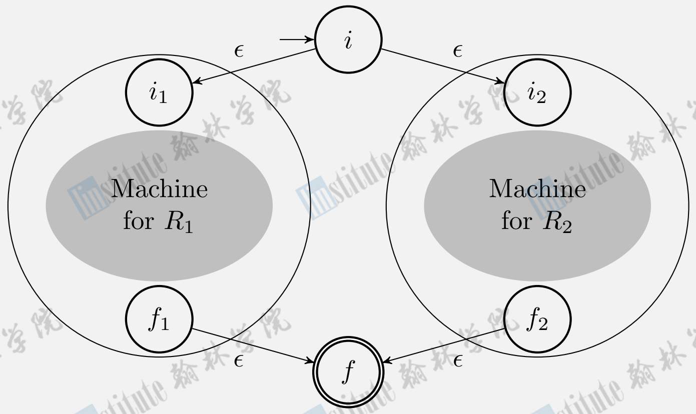


(b) Let i_1 and f_1 denote the initial and accepting states of the NFA for R_1, and likewise, i_2 and f_2 denote the initial and accepting states of the NFA for R_2. To build a machine for R_1R_2, we take all the states in each NFA, and let i_1 and f_2 be the initial and accepting states of the combined machine. We add a single arrow, an ϵ-channel from f_1 to i_2. This is shown diagrammatically below.

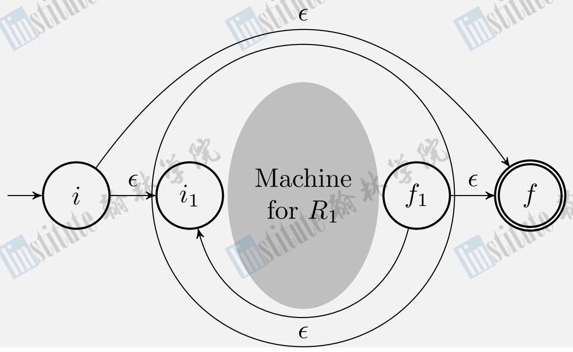


It is clear that this machine accepts exactly the strings in the language R_1R_2.

(c) Using the same notation as in part (b), we can construct a machine for the disjunction in a similar way, shown below. We have added two additional states, i and f, to be the initial and accepting states of the new machine.



(d) In order to accept $R_1^\star$, we need to be able to loop back to the initial state of the machine for R_1 once we reach the accepting state. Additionally, we have to accept the empty string. The following construction works, using the same notation as parts (b) and (c).



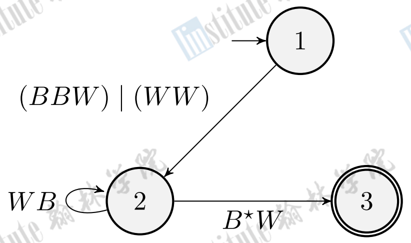


Figure 2: An example generalized NFA.

Now, whenever Amanda receives a pattern, she can build an NFA which recognizes that pattern. After she's done with it, she puts it in a closet with all her other old NFAs, without labeling it.

At the end of Acme, Inc.'s fiscal year, Amanda receives a notice from the higher-ups:

```
TO ALL PATTERN MATCHERS:

SUBMIT A LIST OF ALL PATTERNS YOU HAVE BEEN ASSIGNED THIS YEAR.

LIST MUST BE RECEIVED BY END OF DAY.
```

Amanda does not keep any kind of records. Her only way of recovering the patterns is to look at her NFAs in the closet and figure out which regular languages they specified.

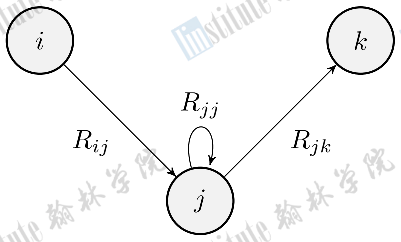
In order to do this, Amanda will reduce an NFA to a smaller *generalized* NFA. A generalized NFA can have its channels labeled by any regular expression, not just single symbols. An ϵ-channel is interpreted as a channel which accepts the empty string.

Example 2.2. *Figure 2 shows a generalized NFA. As usual, the initial state 1 starts out being active. It waits for a pattern matching the expression* $(BBW) \mid (WW)$*. If it sees such a pattern, it activates state 2. Then state 2 waits for a pattern matching* WB*, for which it activates itself; or a pattern matching* $B^\star W$*, for which it activates the accepting state 3. If state 3 is active when the conveyor belt stops, then the pattern is accepted.*

Problem 5: This problem outlines a method for reducing an arbitrary NFA to a generalized NFA with only two states (an initial and an accepting states) and a channel from the initial to the accepting state.

(a) (3 points) Show that any NFA can be converted to a machine that has no channels entering the initial state, no channels leaving the accepting state, and distinct initial and accepting states. We call such an NFA *rectified*.

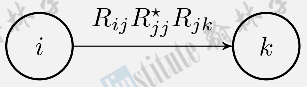
(b) (4 points) A configuration of states i, j, and k is shown below. The channels have regular expressions denoted by R_{ij}, R_{jj}, and R_{jk}. Show that this configuration is equivalent to one in which state j is removed, and a new channel is placed between states i and k.



(c) (3 points) Let j be an arbitrary state in a generalized NFA, having incoming channels from states $a_1, \ldots, a_n$ and outgoing channels to states $b_1, \ldots, b_m$. Show that an equivalent NFA without node j can be constructed by adding channels from a_i to b_k for $i = 1, \ldots, n$ and $k = 1, \ldots, m$.
(d) (2 points) Show that, if a generalized NFA has n channels between states i and j, it is equivalent to a generalized NFA with only a single channel between states i and j.
(e) (8 points) Using parts (a)-(d), establish that an arbitrary NFA is equivalent to a generalized NFA with only an initial and an accepting state, and a single channel between them. It follows that the original NFA accepts the regular language specified by the regular expression labeling the single channel in the reduced NFA.

(a) Let i and f be the initial and accepting states of some NFA, with possibly $i = f$. We can add additional, distinct states i' and f', and make these the new initial and accepting states. We also add ϵ-channels $i' \to i$ and $f \to f'$. The new NFA is equivalent to the original one, and is rectified.

(b) The given configuration of states is equivalent to



(c) If we add a channel with regular expression $R_{a_ij}R_{jj}^{\star}R_{jb_k}$ between a_i and b_k, for each $i = 1, \ldots, n$ and $k = 1, \ldots, m$, and then remove the state j, we have an equivalent NFA.

(d) If we have channels with regular expressions $R_1, \ldots, R_n$ between states i and j, then we can replace this by a single channel with regular expression $R_1 \mid \cdots \mid R_n$ between i and j.

(e) Let M be an NFA; by part (a), it is equivalent to a rectified NFA M'. Additionally, in the construction of part (a), two states are added to M to form M'. Thus, if M' has n states, then $n > 2$.

Now, pick a state j in M' which is neither the initial nor the accepting state (such a state exists because $n > 2$). By part (c), we can remove state j by adding additional channels between the other states. Thus, M' is equivalent to an NFA M'' with $n-1$ states. Additionally, in the construction of part (c), we only add incoming channels to states which already had incoming channels; we only add outgoing channels to states which already had outgoing channels; and we do not change which state is initial and which state is accepting. Thus, M'' is rectified.

By induction, M' (and therefore M) is equivalent to a rectified NFA with two states. A rectified NFA with two states can only consist of an initial state, an accepting state, and some number of channels between them. By part (d), we can collapse these channels into a single channel. This proves the result.

Now Amanda can build an NFA for any regular language, and she can take a machine and reconstruct the regular language which it accepts. This ability can be summarized in the following theorem.

Theorem 2.1. *Let L be a language. The following are equivalent:*

(a) L is regular

(b) There exists an NFA which decides L

3 Deterministic Finite Automata

Amanda's life is going pretty well now. She shows up to work, configures an NFA for the pattern of the day, and then sits back and relaxes while the machine does her job for her. As long as she has access to Acme's NFA facilities, nothing can stop her.

Unfortunately, due to an economic downturn, Acme has to scale down the automata division. Acme will no longer product the advanced NFA machines; they will only produce a restricted version, called deterministic finite automata, or DFAs. A DFA can only have one state active at a time; every state has exactly one outgoing B channel and one outgoing W channel, and there are no ϵ-channels. The only advantage to a DFA is that it is permitted to have multiple accept states, and the DFA accepts a pattern if it finishes in any of these states.

This seems to be the end of Amanda's plan to automate her job. The construction of an NFA to recognize a regular language relies heavily on ϵ-channels and multiple states being active, something a DFA cannot have. Amanda's only hope is to convert the NFA which recognizes a regular language into an equivalent DFA. In fact, this is possible.

Problem 6: An NFA has no memory, so its configuration is fully specified by which of its states are active at any given time step.
(a) (2 points) Assume we have a machine with n nodes. Show that there are 2^n possible configurations of the machine.
(b) (8 points) Using the configurations defined in part (a), determine which configuration the machine goes into given the configuration it is in and the color of ball which appears on the conveyor belt.
(c) (5 points) Show that any NFA can be converted into an equivalent DFA. Perform such a conversion on the machine in Figure 3.

(a) The different configurations correspond to the different subsets of the n states. There are 2^n such subsets. We denote the set of all states of the NFA by Σ, and the set of configurations by 2^Σ.

(b) Assume that at time t the NFA is in configuration $A \in 2^\Sigma$, so $A \subset \Sigma$. If the symbol $\alpha \in \{B, W\}$ comes across the conveyor belt, then the subset

$$f(A;\alpha) = \bigcup_{i \in A} S_i^{(\alpha)}$$

will be activated at time $t+1$. Additionally, all states which can be reached from these by ϵ-channels will be activated. To express this formally, we define a function $f_\epsilon : 2^\Sigma \to 2^\Sigma$ by

$$g_\epsilon(A) = A \cup \bigcup_{i \in A} S_i^{(\epsilon)}.$$

Clearly $A \subset g_\epsilon(A)$ for any $A \in 2^\Sigma$. Therefore, we have a sequence of inclusions

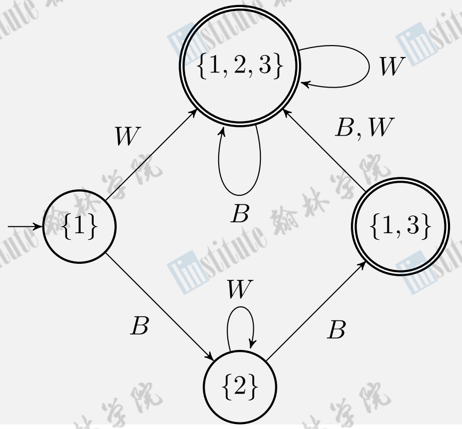
$$A \subset g_\epsilon(A) \subset g_\epsilon^2(A) \subset \cdots .$$

Since every term in this sequence is a subset of the finite set Σ, the sequence must eventually stabilize. Thus,

$$f_\epsilon(A) = \bigcup_{n=1}^{\infty} g_\epsilon^n(A)$$

is well-defined. Therefore, when the NFA is in configuration A and the conveyor belt has the symbol α, the NFA moves to the configuration $f_\epsilon(f(A,\alpha))$.

(c) From parts (a) and (b), it is clear that we should build a DFA with 8 states, labeled by the 8 subsets of $\{1,2,3\}$. We also need to determine how each configuration changes in response to each alphabet symbol. In this case some of the 8 states are not reachable, so we can discard them. The initial state is $\{1\}$, and the accept states are any subsets containing 3. The correct result is shown below.



We can now extend Theorem 2.1 to the following:

Theorem 3.1. *Let L be a language. The following are equivalent:*

(a) L is regular

(b) There exists an NFA which decides L

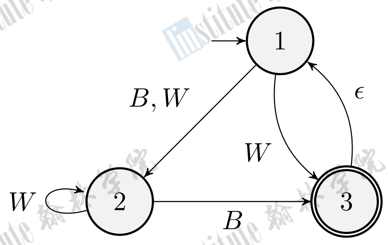


Figure 3: An example NFA.

(c) There exists a DFA which decides L

4 Pumping Lemma

Amanda's supervisors are impressed. All her pattern matching work is impeccable (as if done by machine!), and she manages to get it all done with only DFAs. Given this success, they decide to challenge her with a new kind of pattern.

```
AMANDA WILSON:

INTERPRET BLACK BALLS AS LEFT PARENTHESES ( AND WHITE BALLS AS RIGHT
PARENTHESES ).

ACCEPT ALL PATTERNS CORRESPONDING TO BALANCED PARENTHESES, E.G. (()()).
```

When Amanda receives this message, she is a little surprised – the patterns usually come pre-coded as regular expressions. This time, she'll just have to write the regular expression herself. But when she sits down to do this, she can't seem to find a regular expression that works.

Could it be that not all languages are regular? Amanda is beginning to suspect as much, but she'd like to know for sure. Looking at her DFAs, she has an idea. If the machine goes through a loop, coming back to the same state twice, then it can be pumped through that loop any number of times.

Problem 7:

(a) (2 points) A DFA has n nodes. After how many passages of balls is it guaranteed to have visited the same state at least twice?

(b) (12 points) Prove the *pumping lemma*: for any regular language L, there is some length n such that any pattern p in L with length at least n can be split as $p = xyz$ such that

- the length of the pattern xy is less than or equal to n,
- the length of the pattern y is at least 1,

- $xy^mz \in L$ for all $m > 0$, where y^m denotes the pattern y repeated m times.

(c) (6 points) Use the pumping lemma to show that the balanced parentheses language is not regular.

(a) After n balls pass by, the DFA has visited $n+1$ of its states. Therefore, by the pigeonhole principle, at least one is repeated.

(b) By Theorem 3.1, there is some DFA M which decides the regular language L. Let n be the number of states in M. By part (a), for any pattern of length n, M must visit a state more than once. Therefore, let $t_1 < t_2 \leq n$ be two time steps at which M is in the same state. Let x consist of the symbols which arrive at times $0, \dots, t_1 - 1$, and let y consist of the symbols which arrive at times $t_1, \dots, t_2 - 1$. Let z be the remainder of the pattern. Then, by construction, the length of xy is less than or equal to n, and the length of y is at least 1. Moreover, the pattern y took the machine M from some state back to that same state. It follows that we can repeat y as many times as we like, and the machine will still end up in this state; so the pattern z will still take it to an accepting state. Thus, $xy^mz \in L$ for all $m > 0$.

(c) Assume to the contrary that the balanced parentheses language were regular. Then, by the pumping lemma, there is some length n which satisfies the conditions laid out in part (b). But consider the following pattern in the balanced parentheses language:

$$p = \underbrace{(\cdots(}_{n \text{ times}}\underbrace{)\cdots)}_{n \text{ times}}.$$

By the pumping lemma, we should be able to split this pattern into $p = xyz$, with the length of xy being less than or equal to n, and the length of y being greater than or equal to one. For this particular p, these conditions imply that y consists of some number of left parentheses. If we form xy^mz for $m > 1$, then there are more left parentheses than right parentheses, so the pattern cannot be balanced. It follows that balanced parentheses do not form a regular language.