8-16-2018

Theory of Computation Lecture Notes (Student Version)

Kyle Burke
Plymouth State University, paithanq@gmail.com

CS 3780: Theory of Computation*

Lecture Notes - Student Version†

Kyle Burke

August 16, 2018

Abstract

Lecture notes for an undergraduate Theory of Computation course. These notes assume some background in discrete math or set theory. The notes deviate from the normal topic order by covering all the machines first, then properties of the language classes, and finally non-inclusion into those classes. Many sections of the notes have yet to be completed.

Contents

-1.0 Resources . . . 3
-1.1 Organization . . . 3
-1.2 Exercises . . . 3
-1.3 In Progress . . . 4
-1.4 Acknowledgements . . . 4
-1.5 LaTeX Commands and Packages . . . 4

0 Motivation and Preview of CS Theory **5**
0.0 Limits of Computation . . . 5
0.1 The Halting Problem is Undecideable . . . 6
0.2 Computing Models . . . 11
0.3 Languages . . . 11
0.4 Math Background . . . 12

1 Automata and Machines **15**
1.0 Deterministic Finite-state Automata (DFAs) . . . 15
1.1 Nondeterminism (NFAs) . . . 27
1.2 Pushdown Automata . . . 39
1.3 Turing Machines . . . 43

2 Language Classes **48**
2.0 Regular Languages . . . 48
2.1 Context-Free Languages . . . 50
2.2 Decideable Languages . . . 56
2.3 Turing-Recognizeable Languages . . . 62

*Kyle would always like to hear about how useful his notes are. If you have any comments about these, please email him at `paithanq@gmail.com` or leave a comment at `https://paithanq.blogspot.com/2018/08/theory-of-computation-lecture-notes.html`.

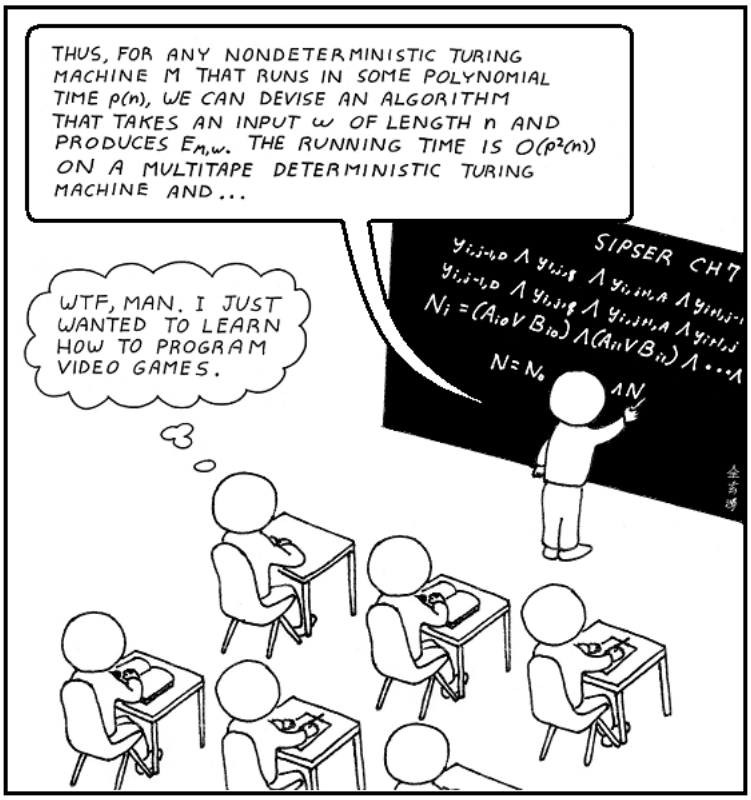
†Created with lectureNotes.sty, which is available at: `http://turing.plymouth.edu/~kgb1013/lectureNotesLatexStyle.php` (or, GitHub: `https://github.com/paithan/LaTeX-LectureNotes`).

3 Non-Inclusion **62**
3.0 Non-Regular Languages 62
3.1 Non-Context-Free Languages 65
3.2 Showing Undecideability (Reductions) 68
3.3 Turing Unrecognizeability 75

4 Computational Complexity **83**
4.1 NP-Completeness 83
4.2 Approximation Algorithms 83
4.3 Hardness of Approximation 85

A Answers to Exercises **85**

B Bibliography **114**



[1]

[1] Abtruse Goose comic, "Rite of Passage". From http://abstrusegoose.com/206

⟨ Go over syllabus/schedule/expectations. ⟩

-1.0 Resources

These notes were created while teaching from Introduction to the Theory of Computation, Third Edition, by Mike Sipser[2][1]. (The chapters and sections here do not correspond with the text chapters.) These notes deviate from the order of topics covered there, but this does not diminish just how useful Sipser's book is! Topics are covered in far more depth than these sparse notes.

When I reach this course, all assignments are to be done in LaTeX. You can use CoCalc[3] to edit and compile everything so you don't need to install LaTeX locally. Also, Stack Overflow has a good list of LaTeXresources here: `http://stackoverflow.com/documentation/latex/topics` (Thanks, Greg!) You'll want to learn to use the `automata` library from the tikz package to create diagrams; there's lots of good information about this online.

-1.1 Organization

In my first time teaching this course, I noticed that students really enjoyed learning about all the automata and machines, but didn't like all the theory served up too soon. For example, they weren't interested in learning too much about why some languages were not regular until they'd already seen in practice that Pushdown Automata are more powerful than DFAs and NFAs.

In order to better harness this motivation, I've reorganized the material so that all of the automata come first (Section 1), followed by more discussion about the different types of languages (Section 2). It is not until Section 3 that we use pumping lemmas to show that some languages are not in some classes. In Section 4, we look at intractibility and how to handle it.

I tested this new ordering for the first time in Fall of 2017 and it went really well. I will do it again for Fall 2018.

-1.2 Exercises

I have included exercises in many of the subsections. For each type of problem, I have tried to include at least three problems:

- At least one problem with the answer given in the end of the book so the student can see things worked through once.
- At least one problem with no answer given, which I assign for zero points. I can answer most questions from students without "giving points away" on the assignment.
- At least one problem assigned for credit which I expect students to do completely on their own.

Some advanced problems do not have this trio of parts.

[2]Book site: `http://www-math.mit.edu/~sipser/book.html`
[3]Cocalc: `https://cocalc.com/`

-1.3 In Progress

These notes are very much a work in progress. I do not expect that everything will be completed by the end of 2018, at which point I won't be teaching this subject again for a while.

Here's my TODO list:

- Indicate, at the beginning of each section, which part of Sipser is most relevant.
- Break apart multi-part homework questions to the point where my own homework assignments
- Fix homework assignments that refer to previous problems. Instead, restate that problem (or remove it altogether).
- Include a table of the best approximation algorithms/hardness-of-approximation results for some classical NP-hard problems.
- I need to add some diagrams. For example, the growing hierarchy of language classes that we build.
- There are entire sections where I haven't yet written up notes.

-1.4 Acknowledgements

Thanks to the Fall 2016 students of CS 3780 for putting up with me learning to teach this course while it was running. I changed the order of some things mid-lecture, and they rolled along with all of the modifications.

Thanks to the developers of the LaTeX automata library in the tikz package[4].

Thank you to Christin Wixson, who manages the Plymouth State's repository of Open Educational Resources. She helped me ensure that these notes fit the requirements for an OER and met the Creative Commons license standards.

Thanks to Lisa Walton, who first taught me this material in the fall of 2001. Sadly, Lisa passed away a few years ago, a definite loss to computer science education.

-1.5 LaTeX Commands and Packages

Make new macros like this:

```
\newcommand{\functionOfN}[1]{
    \ensuremath{f(n)=#1}
}

\newcommand{\homeworkProblem}[2]{
    \item \textbf{#1}:\\
    #2
}
```

These should be in the preamble of your document.

You can use these inside the document like this:

```
\functionOfN{4n + \sqrt{35}}

\begin{itemize}
    \homeworkProblem{3.5.2}{$R = \{2\}$}
\end{itemize}
```

[4] I don't know the homepage of the library, but here's a Stack Exchange answer with an example: `http://tex.stackexchange.com/questions/20784/which-package-can-be-used-to-draw-automata`

0 Motivation and Preview of CS Theory

Q: What does it mean for something to be *uncomputable*?

A:

0.0 Limits of Computation

Q: What if I limit the scope to computational problems? First, what is a computational *problem*?

A:

Here are some example problems, phrased as questions: Examples:

- "Is this list of numbers, X, sorted in increasing order?"
- "What is the result of this list of numbers, X, if I sort it in increasing order?"

Q: I have lots more to say about this, but first, let's return to the question of uncomputable. What could that mean if I have an uncomputable problem?

A:

There are things that we just can't write a program to do. Any program. In Python, Java, C/C++, etc. No programming language that we know can solve *undecideable* problems.

Q: What does that mean, more specifically?

A:

0.1 The Halting Problem is Undecideable

Q: Time for an example: The *Halting Problem*. What's that?

A:

Expect lots of confusion about this. Why is this undecideable?

Q: Why can't we just run the program to see if it halts? (Students will probably ask this one.)

A:

Expect them to ask more questions. Why don't we do this? Or this?

Q: What's a really good way to convince you that there's no program to do this?

A:

Q: What kind of proof techniques are you familiar with?

A:

We're going to use a proof by contradiction. I find that the proof is best explained by a Seussian poem, "Scooping the Loop Snooper" by Geoffrey K. Pullum at U. Edinburgh[5]. Here's the text (with my comments interspersed)

No general procedure for bug checks will do.
Now, I won't just assert that, I'll prove it to you.
I will prove that although you might work till you drop,
you cannot tell if computation will stop.

For imagine we have a procedure called P
that for specified input permits you to see
whether specified source code, with all of its faults,
defines a routine that eventually halts.

Q:	What kind of proof techinque am I using, assuming that program P does exist? Hint: in Latin, it's known as reductio ad absurdum.
A:	

Q:	What does that mean I'm going to show? What kind of situation am I going to arrive at?
A:	

You feed in your program, with suitable data,
and P gets to work, and a little while later
(in finite compute time) correctly infers
whether infinite looping behavior occurs.

If there will be no looping, then P prints out 'Good.'
That means work on this input will halt, as it should.
But if it detects an unstoppable loop,
then P reports 'Bad!' — which means you're in the soup.

Okay, let's summarize that on the board:

$$P(X,Y) \rightarrow \begin{cases} \text{Good} & \text{, if } X(Y) \text{ halts} \\ \text{Bad} & \text{, if } X(Y) \text{ loops infinitely} \end{cases}$$

[5] http://www.lel.ed.ac.uk/~gpullum/loopsnoop.html

Q: Does P solve the halting problem?

A:

Well, the truth is that P cannot possibly be,
because if you wrote it and gave it to me,
I could use it to set up a logical bind
that would shatter your reason and scramble your mind.

Here's the trick that I'll use — and it's simple to do.
I'll define a procedure, which I will call Q,
that will use P's predictions of halting success
to stir up a terrible logical mess.

For a specified program, say A, one supplies,
the first step of this program called Q I devise
is to find out from P what's the right thing to say
of the looping behavior of A run on A.

If P's answer is 'Bad!', Q will suddenly stop.
But otherwise, Q will go back to the top,
and start off again, looping endlessly back,
till the universe dies and turns frozen and black.

Q: Okay, let's write down the steps of Q. I wrote it in 3 steps. Step 0 is: Run $P(A, A) \to R$.
What are the other steps?

A:

And this program called Q wouldn't stay on the shelf;
I would ask it to forecast its run on itself.
When it reads its own source code, just what will it do?
What's the looping behavior of Q run on Q?

If P warns of infinite loops, Q will quit;
yet P is supposed to speak truly of it!
And if Q's going to quit, then P should say 'Good.'

Which makes Q start to loop! (P denied that it would.)

No matter how P might perform, Q will scoop it:
Q uses P's output to make P look stupid.
Whatever P says, it cannot predict Q:
P is right when it's wrong, and is false when it's true!

I've created a paradox, neat as can be —
and simply by using your putative P.
When you posited P you stepped into a snare;
Your assumption has led you right into my lair.

So where can this argument possibly go?
I don't have to tell you; I'm sure you must know.
A reductio: There cannot possibly be
a procedure that acts like the mythical P.

You can never find general mechanical means
for predicting the acts of computing machines;
it's something that cannot be done. So we users
must find our own bugs. Our computers are losers!

Q: What's the paradox?

A:

Q: Does this mean there's no way for us to solve the halting problem?

A:

For another version of the proof, you can watch a fun video: `https://www.youtube.com/watch?v=92WHN-pAFCs`.

Q: Why do I care about this? I'm going to be a web developer, not a theorist!

A:

Also, you don't want to fall for a hilarious joke like the one posted to `getacoder.com` in November, 2008. As covered by Will Benton's blog[6], user "AlanT" proposed the following project for programmers to bid on:

"The purpose of this project is to create a debugger program. This program will take as input the source code another program, and will analyze that other program and determine if it will run to completion, or have an error, or go into an infinite loop.

"To state that another way, given a function f and input x, determine if f(x) will halt."

The post and the responses are cached at `http://turing.plymouth.edu/~kgb1013/3780/images/getacoderHalting.pdf`[7]. Here are some highlights:

- User `kagtech` from Chennai, India is the last response, promising a solution in 18 days for $1,800, saying: "We have gone through your project specifications; give us the opportunity to arise up your needs to be done. We can do this task as sprint; let us do this work for you as per your timeline and schedule. We have a top-notch team. We ensure that your feedback and comments are timely reflected in the technical specification on all stages of the projects Development. We are ready to start the work. Thanks & Warm Regards, Team - KAG Tech."

  There are lots like this one.
- User `eliterammer` claims to be able to solve the halting problem in 10 days for $300!
- User `germanquality` can apparently do it for the same price, but in only 1 day! Also, there's some extreme biasing at work: "As the superior German programmer I am I've already solved the problem in my head as per your specification. I'm able to deliver a solution in source code in any language that can print a line of text. If necessary, I can also provide flowcharts and a solution on solid German-made paper".
- Some users got the joke. For example, `GeorgeCantor` says "Herr T, Despite the fact that I died in 1918 you may find the following advice important. I

[6] Post originally at `http://blog.willbenton.com/2008/11/rent-a-coder-hilarity/`, but it seems to be broken now. There's an old YCombinator article (`https://news.ycombinator.com/item?id=5660860`) about it with lots of good comments.

[7] This is my copy of Will Benton's PDF of the article that was at `http://blog.willbenton.com/wp-content/uploads//2008/11/bug-finder.pdf`. Thanks to Will Benton for saving this!

would be happy to work on your project but you will quickly find that it is impossible, nay undecidable. It is a simple consequence of my diagonalization argument for uncountable numbers that were such a program to exist it would be possible to create a program on which it could not work and hence you would obtain a contradiction and your original program would 'disappear in a puff of logic'. Gruß Gott, Georg"

0.2 Computing Models

Humans seem to be able to solve the halting problem. There may be some other "computing" device that someone invents that does solve the halting problem. It just means we can't do it in our current computing model.

Q: What's our computing model based on?

A:

Q: What does it mean when we say a programming language is complete?

A:

Are we going to learn about turing machines now? No, we'll get there, though. We're going to learn about simpler models of computation first. Plan:

- Finite Automata, then
- Pushdown Automata, then
- Turing Machines

0.3 Languages

Q: Recall the two sorting-based problems I mentioned at the beginning. What were those?

A:

We're going to focus on *boolean problems*, meaning the answer is either Yes or No. (True or False. Monkey or Gibbon.)

Actually, we're going say *Accept* and *Reject*. Each of the automata (machines) we create is going to return one of these two things on any input. The exact set of inputs that a machine answers *Accept* on is called a *language*. If automata M *recognizes* language A, then we say $L(M) = A$.

Q: Because we're calling it a language, what kinds of inputs are we going to focus on?

A:

Q: What are some common operations on strings? (I'm interested in two in particular)

A:

- Concatenation, and
- Repetition

Let's define some operations on sets of strings for each of these. Given languages (sets of strings) S, R, and T:

- $R \circ T = \{rt \mid r \in R \wedge t \in T\}$
- $S^* = \{s_1 s_2 s_3 \cdots s_n \mid n \in \mathbf{N} \wedge \forall i : s_i \in S\}$

Q: Do you think the different types of automata will accept different types of languages?

A:

As we get deeper in the material, the languages will be more complex:

- Finite-State Automata: Regular Languages
- Pushdown Automata: Context-Free Languages
- Turing Machines: Decideable Languages

0.4 Math Background

There's lots of math I'm expecting you to be familiar with. I'm not going to cover that all here. This material is covered in Sipser in section 0. [1]

Exercises for 0.4

Exercise 0

Write math-notation descriptions of each of the following sets. You do not need to show any work for these.

a) The set containing only the numbers -10, -3, 3, and 9.

b) The set of all even integers.

c) The set of all rational numbers ($\mathbb{Q}$) greater than or equal to 20.

d) The set of all rational numbers where the denominator is at most 2.

(Answer 0.4.0 in Appendix)

Exercise 1

Write math-notation descriptions of each of the following sets. (You should not use any ellipses (...).) You do not need to show any work for these.

a) The set containing only the strings `a`, `aa`, `aba`, `abc`.

b) The set of all real numbers between -1 and 1 (inclusive).

Exercise 2

Write math-notation descriptions of each of the following sets. (You should not use any ellipses (...).) You do not need to show any work for these.

a) The set of odd integers.

b) The set with no elements.

c) The set containing only the empty set.

d) The set containing only the empty string. (LaTeX hint: `\varepsilon` $\rightarrow \varepsilon$)

Exercise 3

Let $S = \{\texttt{a}, \texttt{b}\}$ and $T = \{\texttt{a}, \texttt{b}, \texttt{c}\}$. Replace the question marks in the following expressions:

a) $S \ ? \ T$ (Replace the ? with $=$, $\subset$, or $\supset$. Note that for some pairs of sets, none of these three is correct.)

b) $S \cup T =?$ (Include the set with all elements.)

c) $S \cap T =?$ (Ditto.)

d) $T \setminus S =?$ (Set-difference)

e) $S \times T =?$ (Cross-product)

f) $\mathcal{P}(\mathcal{S}) =?$ (Power set)

(Answer 0.4.3 in Appendix)

Exercise 4

Let $S = \{\texttt{a}, \texttt{b}\}$ and $T = \{\texttt{a}, \texttt{c}, \texttt{c}\}$. Replace the question marks in the following expressions with the set expression. (I.e., do not use S and T in your answers.)

a) $S \cup T =?$

b) $S \cap T =?$

c) $T \setminus S =?$
d) $S \times T =?$
e) $\mathcal{P}(\mathcal{S}) =?$

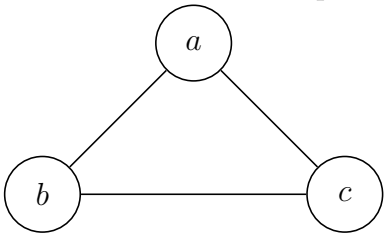
Exercise 5

Let $S = \{\mathtt{a}, \mathtt{d}\}$ and $T = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. Replace the question marks in the following expressions with the set expression. (I.e., do not use S and T in your answers.)

a) $S \cup T =?$
b) $S \cap T =?$
c) $T \setminus S =?$
d) $S \times T =?$
e) $\mathcal{P}(\mathcal{S}) =?$

Exercise 6

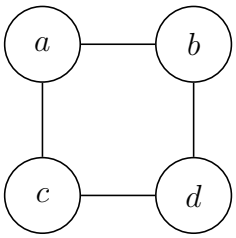
Write out the set description of this graph (the formal version, meaning $G = (V, E) = \ldots$):



(Answer 0.4.6 in Appendix)

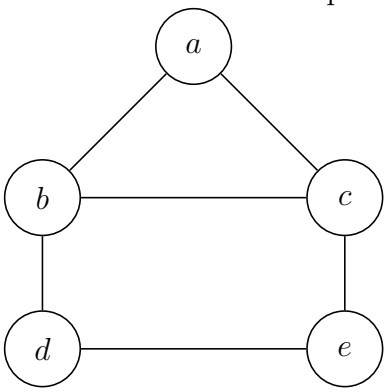
Exercise 7

Write out the set description of this graph (the formal version, meaning $G = (V, E) = \ldots$):



Exercise 8

Write out the set description of the graph below.



1 Automata and Machines

In this section, we'll learn about different types of computational machines, with different levels of computational power:

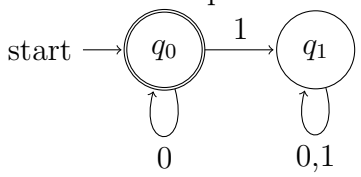
- Finite State Automata, which are either Deterministic (Section 1.0) or Non-deterministic (Section 1.1).
- Pushdown Automata (Section 1.2)
- Turing Machines (**??**)

1.0 Deterministic Finite-state Automata (DFAs)

This material is covered in Sipser in section 1.1. [1]

Although all of the machines we'll see have a finite number of "states", *Finite State Automata* are the most restricted because they can't store any extra data.

Here's an example of a drawing of a Deterministic Finite-state Automata:



Q:	There are two states: q_0 and q_1. Which one do you think is the starting one?
A:	

⟨ Keep describing the parts of the automata. ⟩

Q:	What language does this accept? What set of bit strings does this machine reach the accept state on?
A:	

If the above machine is M, then we write $L(M) = \{0^k | k \in \mathbb{N}\}$, or
$L(M) = \{0^*\}$

Q:	Design a DFA, M, such that $L(M) = \{s = 11\ldots1 \mid$ length of s is $k \in \mathbb{N} \setminus \{0\}\} = \{1^+\}$ Hint: you need a third state for this.
A:	

Notice that we have a "trash state" for strings that we find are illegal. That's often a thing.

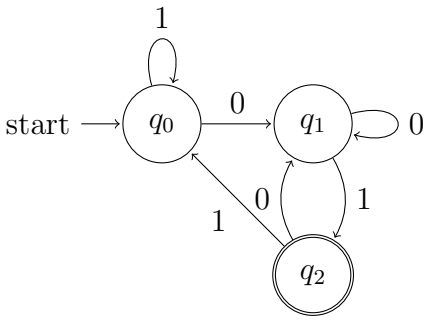
⟨ Show them how to create the first automata in TikZ from the code here. ⟩

Q: How can you draw the first automata in LaTeX?

A:

```
\begin{tikzpicture}[shorten >=1pt,node distance=2cm,on grid,auto]
   \node[state,initial,accepting] (q_0)   {$q_0$};
   \node[state] (q_1) [right=of q_0] {$q_1$};
    \path[->]
    (q_0) edge  [loop below] node {0} ()
          edge  node  {1} (q_1)
    (q_1) edge [loop below] node {0,1} ();
\end{tikzpicture}
```

Q: What is the language the following automata, M, accepts? Bonus: What is the LaTeX code for the automata?



A:

Q: Draw a DFA, M, such that $L(M) = \{(01)^*\}$ (Important: read the string from left to right.) Hint: I used three states.

A:

Q: New notation: $+$. Draw a DFA, M, such that $L(M) = \{(01)^+\} = \{01(01)^*\}$

A:

Q: Another one!
$L(M) = \{s | s$ starts with two zeroes and ends with two zeroes.$\}$

A:

Q: Another one! $L(M) = \{0^n 1^n | n \in \mathbb{N}\}$

A:

Q: What does that mean?

A:

It turns out that it is a context-free language, though! We'll see how to build a Push-Down Automata that recognizes it later.

Some of you might not like pictures. Let's learn to write out a DFA formally:

A DFA is a 5-tuple:

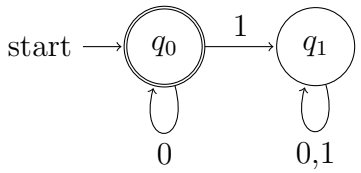
(states, alphabet, transitions, start, accepts)

Unfortunately, mathematicians don't like words as their variables, so the common notation is:

$(Q, \Sigma, \delta, q_0, F)$, where:

- Q is a finite set of states
- Σ is a finite set of characters in the strings, called the alphabet.
- δ is the transition function. $\delta : (Q \times \Sigma) \to Q$
- q_0 is the start state. $q_0 \in Q$
- F is the set of accepting states (yes, there can be more than one!) $F \subseteq Q$

Let's do the first machine I showed, as an example:



Build out the five parts first:

- $Q = \{q_0, q_1\}$
- $\Sigma = \{0, 1\}$
- $\delta = \{((q_0, 0), q_0), ((q_0, 1), q_1), ((q_1, 0), q_1), ((q_1, 1), q_1)\}$
- $q_0 = q_0$
- $F = \{q_0\}$

Q: So, what's the 5-tuple?

A:

In your homeworks, you can use either notation... or combine them. There's also an alternate way to describe the transition function:

	0	1
q_0	q_0	q_1
q_1	q_1	q_1

I'm happy with all of these methods, and any combination.

Let's do another one! Find the machine (and write it out formally) for the language, $L(M) = \{s | s$ is all zeroes or all ones.$\}$. (Yes, this includes the empty string.)

Q: What's the first part?

A:

⟨ TODO: draw it out. ⟩

Q: What should we do next?

A:

Q: Great! Do it!

A:

Q: What's are the formal definitions of the other two automata we've seen?

A:

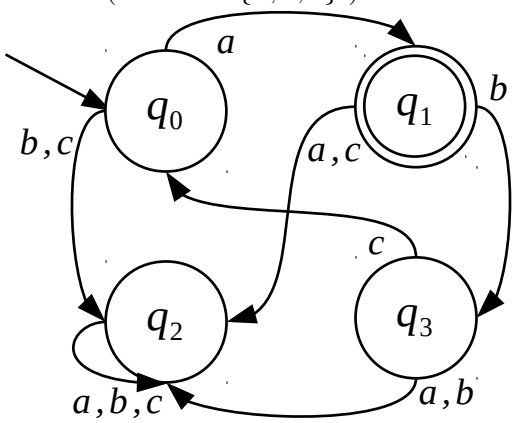
There's a whole part of book section 1.1 called **Designing Finite Automata**.

Q: What's the other way we could ask these questions?

A:

Let's try one: What's the language recognized by the following machine? (Let $\Sigma = \{a, b, c\}$.)

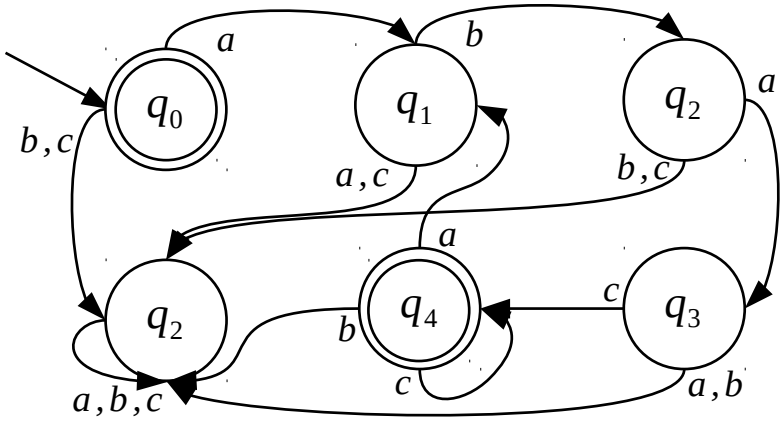
Q:



A:

Another: What's the language recognized by the following machine? (Let $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$.)

Q:



A:

⟨ Play the following game: Split the class up into two teams. Then:

1. Each draws a DFA on the board on separate panels.
2. Teams swap places.
3. Ask the teams to try to figure out the language of the *other* team's DFA.
4. Then modify the other team's DFA by moving one edge.

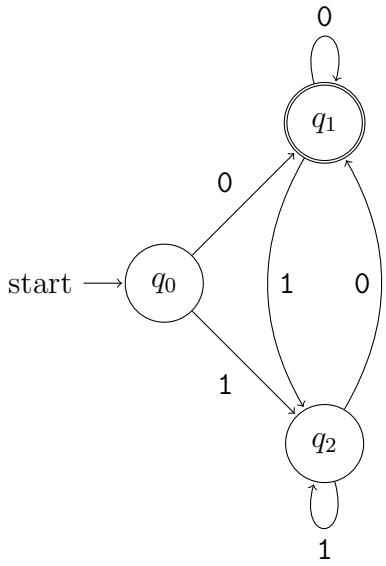
5. Swap places again.
6. Continue *ad infinitum*.

⟩

Exercises for 1.0

Exercise 0

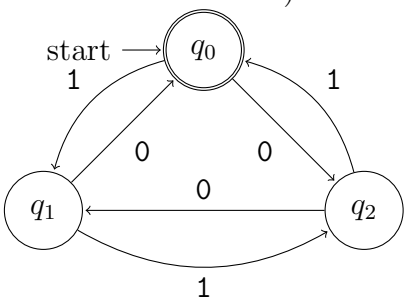
Write out the following DFA, D, using formal (set) notation. (You may use a table for the transition function.)



(Answer 1.0.0 in Appendix)

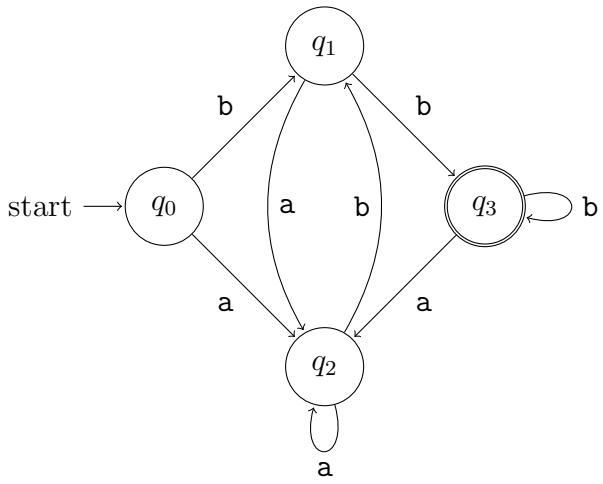
Exercise 1

Write out the following DFA, D, using formal (set) notation. (You may use a table for the transition function.)



Exercise 2

Write out the following DFA, D, using formal (set) notation. (You may use a table for the transition function.)



Exercise 3

Draw the automata for DFA $D = (Q, \Sigma, \delta, q_0, F)$, where:

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{\mathtt{0}, \mathtt{1}\}$
- $$\begin{aligned}\delta = \{&(q_0, \mathtt{0}, q_1),\\ &(q_0, \mathtt{1}, q_2),\\ &(q_1, \mathtt{0}, q_1),\\ &(q_1, \mathtt{1}, q_2),\\ &(q_2, \mathtt{0}, q_2),\\ &(q_2, \mathtt{1}, q_2)\}\end{aligned}$$
- $F = \{q_1\}$

(Answer 1.0.3 in Appendix)

Exercise 4

Draw the automata for DFA $D = (Q, \Sigma, \delta, q_0, F)$, where:

- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$

- $$\begin{aligned}\delta = \{&(q_0, \mathtt{a}, q_1),\\ &(q_0, \mathtt{b}, q_3),\\ &(q_0, \mathtt{c}, q_0),\\ &(q_1, \mathtt{a}, q_2),\\ &(q_1, \mathtt{b}, q_0),\\ &(q_1, \mathtt{c}, q_1),\\ &(q_2, \mathtt{a}, q_3),\\ &(q_2, \mathtt{b}, q_1),\\ &(q_2, \mathtt{c}, q_2),\\ &(q_3, \mathtt{a}, q_0),\\ &(q_3, \mathtt{b}, q_2),\\ &(q_3, \mathtt{c}, q_3)\}\end{aligned}$$
- $F = \{q_2\}$

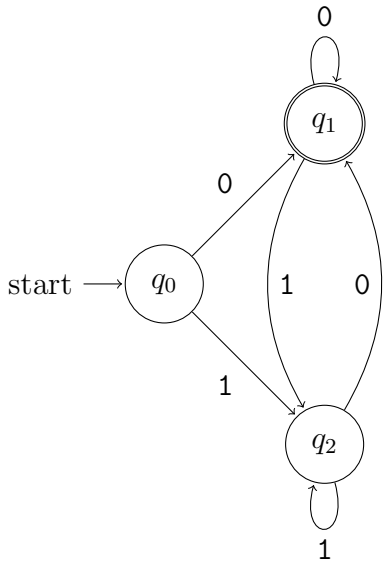
Exercise 5

Draw the automata for DFA $D = (Q, \Sigma, \delta, q_0, F)$, where:

- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{\mathtt{a}, \mathtt{b}\}$
- $$\begin{aligned}\delta = \{&(q_0, \mathtt{a}, q_1),\\ &(q_0, \mathtt{b}, q_3),\\ &(q_1, \mathtt{a}, q_3),\\ &(q_1, \mathtt{b}, q_2),\\ &(q_2, \mathtt{a}, q_0),\\ &(q_2, \mathtt{b}, q_3),\\ &(q_3, \mathtt{a}, q_3),\\ &(q_3, \mathtt{b}, q_3)\}\end{aligned}$$
- $F = \{q_2\}$

Exercise 6

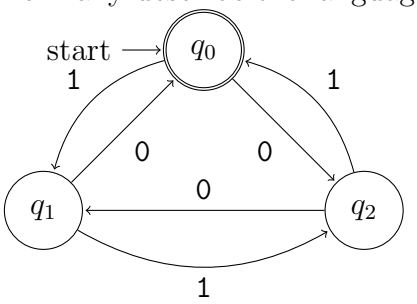
Formally describe the language of the machine, D, from Exercise 1.0.0:



(Answer 1.0.6 in Appendix)

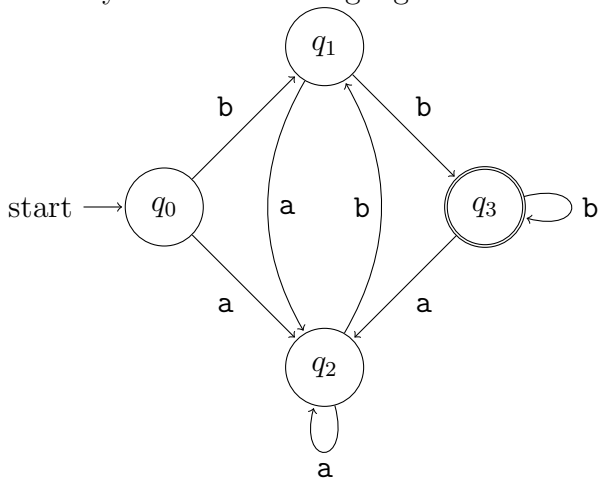
Exercise 7

Formally describe the language of the machine, D, from Exercise 1.0.1:



Exercise 8

Formally describe the language of the machine, D, from Exercise 1.0.2:



Exercise 9

Formally describe the language of DFA $D = (Q, \Sigma, \delta, q_0, F)$, where:

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{\mathtt{0}, \mathtt{1}\}$
- $$\begin{aligned}\delta = \{&(q_0, \mathtt{0}, q_1),\\ &(q_0, \mathtt{1}, q_2),\\ &(q_1, \mathtt{0}, q_1),\\ &(q_1, \mathtt{1}, q_2),\\ &(q_2, \mathtt{0}, q_2),\\ &(q_2, \mathtt{1}, q_2)\}\end{aligned}$$
- $F = \{q_1\}$

(This is the machine from Exercise 1.0.3.)
(Answer 1.0.9 in Appendix)

Exercise 10

Formally describe the language of DFA $D = (Q, \Sigma, \delta, q_0, F)$, where:

- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$
- $$\begin{aligned}\delta = \{&(q_0, \mathtt{a}, q_1),\\ &(q_0, \mathtt{b}, q_3),\\ &(q_0, \mathtt{c}, q_0),\\ &(q_1, \mathtt{a}, q_2),\\ &(q_1, \mathtt{b}, q_0),\\ &(q_1, \mathtt{c}, q_1),\\ &(q_2, \mathtt{a}, q_3),\\ &(q_2, \mathtt{b}, q_1),\\ &(q_2, \mathtt{c}, q_2),\\ &(q_3, \mathtt{a}, q_0),\\ &(q_3, \mathtt{b}, q_2),\\ &(q_3, \mathtt{c}, q_3)\}\end{aligned}$$
- $F = \{q_2\}$

(This is the machine from Exercise 1.0.4.)

Exercise 11

Formally describe the language of DFA $D = (Q, \Sigma, \delta, q_0, F)$, where:

- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{\mathtt{a}, \mathtt{b}\}$

- $$\begin{aligned}\delta = \{&(q_0, \texttt{a}, q_1),\\ &(q_0, \texttt{b}, q_3),\\ &(q_1, \texttt{a}, q_3),\\ &(q_1, \texttt{b}, q_2),\\ &(q_2, \texttt{a}, q_0),\\ &(q_2, \texttt{b}, q_3),\\ &(q_3, \texttt{a}, q_3),\\ &(q_3, \texttt{b}, q_3)\}\end{aligned}$$
- $F = \{q_2\}$

(This is the machine from Exercise 1.0.5.)

Exercise 12

Describe the DFA that recognizes the following language, $L \subset \{\texttt{0}, \texttt{1}\}^*$:
$\{w \mid w$ contains a `1` and a `0` somewhere before that `1`$\}$
Use either a figure or set notation to describe the automata.
(Answer 1.0.12 in Appendix)

Exercise 13

Describe the DFA that recognizes the following language $L \subset \{\texttt{a}, \texttt{b}\}^*$:
$\{w \mid w$ contains the substring `aaa`$\}$
Use either a figure or set notation to describe the automata.
(Answer 1.0.13 in Appendix)

Exercise 14

Describe the DFA that recognizes the following language, $L \subset \{\texttt{a}, \texttt{b}\}^*$:
$L = \{w \mid w$ does not contain the substring `bab`$\}$
Use either a figure or set notation to describe the automata.

Exercise 15

Describe the DFA that recognizes the following language, $L \subset \{\texttt{a}, \texttt{b}\}^*$:
$L = \{w \mid w$ starts with an `a` and ends with a `b`$\}$
Use either a figure or set notation to describe the automata.

Exercise 16

Describe the DFA that recognizes the following language, $L \subset \{\texttt{a}, \texttt{b}\}^*$:
$L = \{w \mid$ the second character of w is `a`$\}$
Use either a figure or set notation to describe the automata.

Exercise 17

Describe the DFA that recognizes the following language, $L \subset \{\texttt{a}, \texttt{b}\}^*$:
$L = \{\texttt{ba}, \texttt{aba}, \texttt{abab}\}$
Use either a figure or set notation to describe the automata.

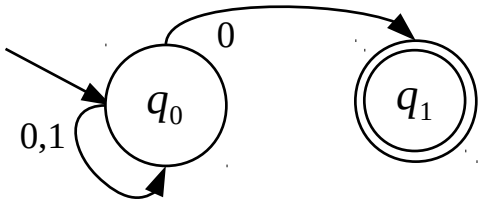
Exercise 18

Describe the DFA that recognizes the following language, $L \subset \{\mathtt{a}, \mathtt{b}\}^*$:
$L = \{w \mid$ every third character of w is $\mathtt{a}\}$ E.g. $\mathtt{bbaabab} \in L$
Use either a figure or set notation to describe the automata.

1.1 Nondeterminism (NFAs)

This material is covered in Sipser in section 1.2. [1]

Q: What's messed up about this automata?



A:

Q: What is it now? (Not a function)

A:

⟨ Talk about nondeterminism - use the parallel method. ⟩

Q: When does an NFA accept an input?

A:

Q: Design an NFA that recognizes $\mathcal{L} = \{w | w$ ends in 101 or 010$\}$

A:

Q: How big is the DFA for this language?

A:

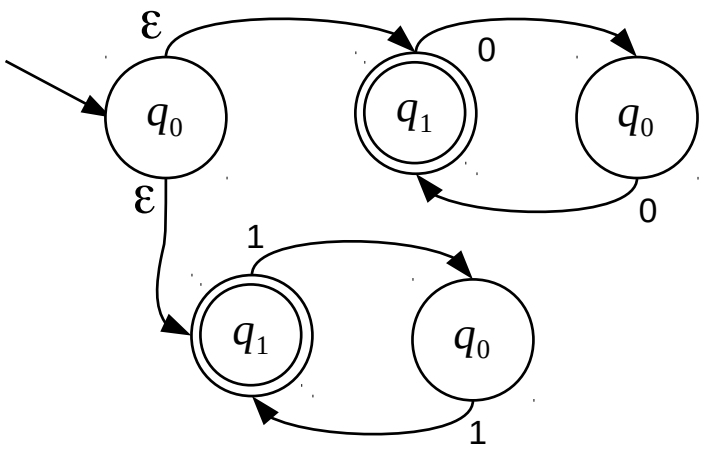
Q: Let's do another one: Design an NFA for $A = \{w \mid w \text{ contains } \texttt{010}\}$.

A:

Q: Let's do another one: Design an NFA that accepts $A = \{\texttt{01}\}^* \circ \{\texttt{0}\}$.

A:

Q: Let's go the other way, which language does this NFA accept:



A:

Q: What is the difference between a function from set X to Y and a relation from X to Y?

A:

Q: We want to write out an NFA in formal notation, a la a DFA. Can we still use a 5-tuple?

A:

Q: One of the five parts is going to be fundamentally different. Which is that?

A:

Q: Is it going to be a relation on $Q \times \Sigma$ and Q?

A:

Q: Okay, let's practice writing out machines in formal notation. First, write out the formal notation for the first machine.

A:

Q:

Determine $L(M)$ for $M = (Q, \Sigma, \delta, q_0, F)$, where

- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{\mathtt{0}, \mathtt{1}\}$
- $$\delta = \{((q_0, \mathtt{0}), q_0), ((q_0, \mathtt{1}), q_0) \\ ((q_0, \mathtt{0}), q_1), ((q_1, \mathtt{1}), q_2) \\ ((q_2, \mathtt{1}), q_3), ((q_2, \varepsilon), q_3) \\ ((q_3, \varepsilon), q_0)\}$$
- $F = \{q_0\}$

Hint: Draw the diagram first.

A:

Q: Let's change the last one by changing F to $\{q_3\}$. Now what's $L(M)$?

A:

Q: How can we write that language with only the three language operations: $*, \cup, \circ$?

A:

Q: Let's change the last one by removing the ε transition from q_3 to q_0. TODO: add the new machine here. What's the new language?

A:

Q: Okay, add that ε back. Let's make one more change, swapping the `1` transition looping at q_0 to go to q_1 instead. Now our transition relation should look like this:

$$\begin{aligned}\delta = \{&((q_0, \texttt{0}), q_0), ((q_0, \texttt{1}), q_1)\\ &((q_0, \texttt{0}), q_1), ((q_1, \texttt{1}), q_2)\\ &((q_2, \texttt{1}), q_3), ((q_2, \varepsilon), q_3)\\ &((q_3, \varepsilon), q_0)\}\end{aligned}$$

What's $L(M)$ now?

A:

TODO: add more of these!

Q: What does it mean for two machines to be equivalent?

A:

Every NFA has an equivalent DFA! Talk about subset construction.
Recall these operations on languages:

- Concatenation: $A \circ B$
- Union: $A \cup B$
- Star: A^*

Q: What do you think each of these mean? What is the language $A \circ B$? Example: $C = \{(01)*\}, D = \{(10)*\}$, what is $C \circ D$? TODO: make sure this is back in section 0.4, then remove it here.

A:

Q: What's the meaning of $A \circ B$, then?

A:

Q: What do you think is true of every $A \circ B$, when A and B are recognizeable by an NFA?

A:

Q: How can we prove it?

A:

⟨ Draw the machine-blobs M_A and M_B. Put a start state and some accepting states in each. ⟩

Q: How do I construct M_C where $C = A \circ B$?

A:

Q: Let $M_A = (Q_A, \Sigma, \delta_A, q_{0_A}, F_A)$ and $M_B = (Q_B, \Sigma, \delta_B, q_{0_B}, F_B)$. What is the formal definition of M_C?

A:

Q: What about union? What's the union of the above C and D?

Q: What's the definition of $A \cup B$?

A:

Q: What do you think is true of $A \cup B$ if A and B are recognizeable by an NFA?

A:

Q: What about star? What does A^* mean?

A:

Q: What do you think is true of A^* if there's an NFA that recognizes A?

A:

A language is called a *regular language* if it's recognized by some DFA (or NFA). We'll learn more about regular languages in Section 2.0.

TODO: Talk about subset construction here!

Exercises for 1.1

Exercise 0

Draw the figure corresponding to the NFA $N = (Q, \Sigma, \delta, q_0, F)$, where:

- $Q = \{q_0, q_1\}$
- $\Sigma = \{\mathtt{0}, \mathtt{1}\}$
- $$\delta = \{(q_0, \mathtt{1}, q_1), \\ (q_1, \mathtt{1}, q_1)\}$$
- $F = \{q_1\}$

(Answer 1.1.0 in Appendix)

Exercise 1

Using only *two transitions*, draw the diagram for an NFA equivalent to the following DFA: DFA $D = (Q, \Sigma, \delta, q_0, F)$, where:

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{\mathtt{0}, \mathtt{1}\}$

- $$\begin{aligned}\delta = \{&(q_0, \texttt{0}, q_1),\\ &(q_0, \texttt{1}, q_2),\\ &(q_1, \texttt{0}, q_1),\\ &(q_1, \texttt{1}, q_2),\\ &(q_2, \texttt{0}, q_2),\\ &(q_2, \texttt{1}, q_2)\}\end{aligned}$$
- $F = \{q_1\}$

(This is the DFA from Exercise 1.0.3.)
(Answer 1.1.1 in Appendix)

Exercise 2

Draw the diagram for an NFA equivalent to the following DFA:
DFA $D = (Q, \Sigma, \delta, q_0, F)$, where:

- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}\}$
- $$\begin{aligned}\delta = \{&(q_0, \texttt{a}, q_1),\\ &(q_0, \texttt{b}, q_3),\\ &(q_0, \texttt{c}, q_0),\\ &(q_1, \texttt{a}, q_2),\\ &(q_1, \texttt{b}, q_0),\\ &(q_1, \texttt{c}, q_1),\\ &(q_2, \texttt{a}, q_3),\\ &(q_2, \texttt{b}, q_1),\\ &(q_2, \texttt{c}, q_2),\\ &(q_3, \texttt{a}, q_0),\\ &(q_3, \texttt{b}, q_2),\\ &(q_3, \texttt{c}, q_3)\}\end{aligned}$$
- $F = \{q_2\}$

(This is the DFA from Exercise 1.0.4.)
TODO: replace this one with a different one... it's not a good one.

Exercise 3

Using only *three transitions*, draw the diagram for an NFA equivalent to the following DFA:
DFA $D = (Q, \Sigma, \delta, q_0, F)$, where:

- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{\texttt{a}, \texttt{b}\}$

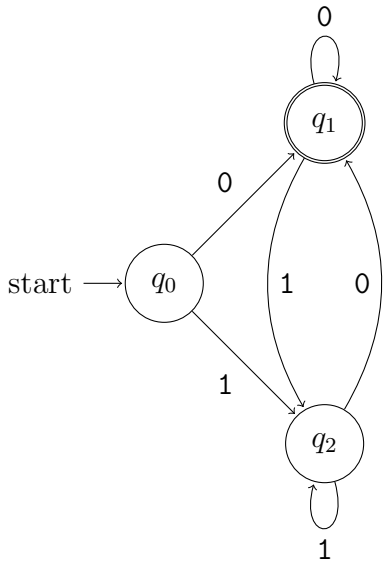
- $$\begin{aligned}\delta = \{&(q_0, \mathtt{a}, q_1),\\ &(q_0, \mathtt{b}, q_3),\\ &(q_1, \mathtt{a}, q_3),\\ &(q_1, \mathtt{b}, q_2),\\ &(q_2, \mathtt{a}, q_0),\\ &(q_2, \mathtt{b}, q_3),\\ &(q_3, \mathtt{a}, q_3),\\ &(q_3, \mathtt{b}, q_3)\}\end{aligned}$$

- $F = \{q_2\}$

(This is the DFA from Exercise 1.0.5.)

Exercise 4

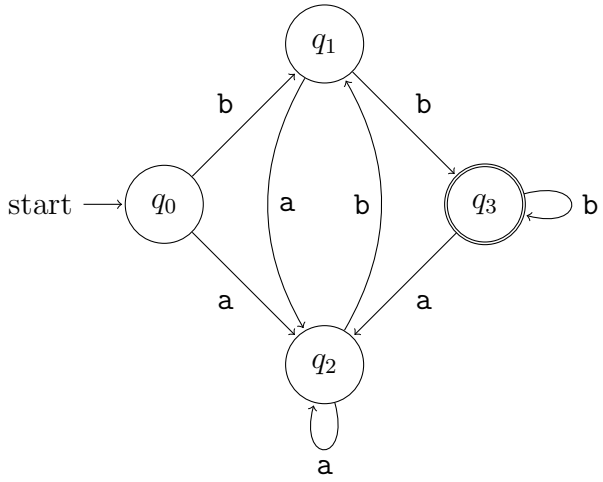
Draw the NFA with the smallest number of transitions equivalent to the following DFA:



(This is the DFA from Exercise 1.0.0.)
(Answer 1.1.4 in Appendix)

Exercise 5

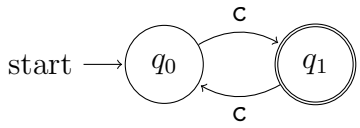
Using only *three arrows* (four actual transitions) draw the diagram for an NFA equivalent to the following DFA:



(This is the DFA from Exercise 1.0.2.)

Exercise 6

Determine the language of the following NFA, M, over alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$.



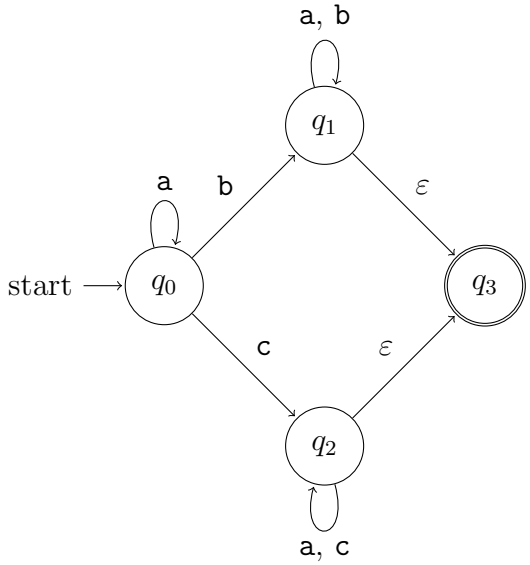
a) Describe the language using set-builder description using an English description, e.g. $L(M) = \{w \mid w \text{ is a string that likes to eat monkeys.}\}$.

b) Describe the language using only explicit sets and the language operations (*, $\cup$, and $\circ$). (E.g. $L(M) = \{\mathtt{x}\}^* \circ (\{\mathtt{y}\} \cup \{\mathtt{z}\})$

(Answer 1.1.6 in Appendix)

Exercise 7

Determine the language of the following NFA, M, over alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$.

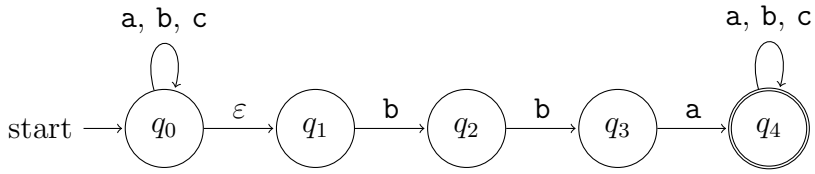


a) Describe the language using set-builder description using an English description.

b) Describe the language using only explicit sets and the language operations (*, $\cup$, and $\circ$).

Exercise 8

Determine the language of the following NFA, M, over alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$.

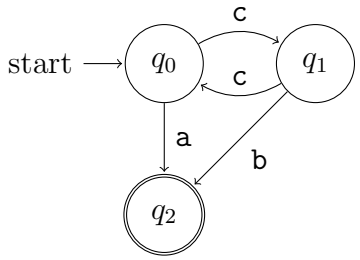


a) Describe the language using set-builder description using an English description.

b) Describe the language using only explicit sets and the language operations (*, $\cup$, and $\circ$).

Exercise 9

Determine the language of the following NFA, M, over alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$.

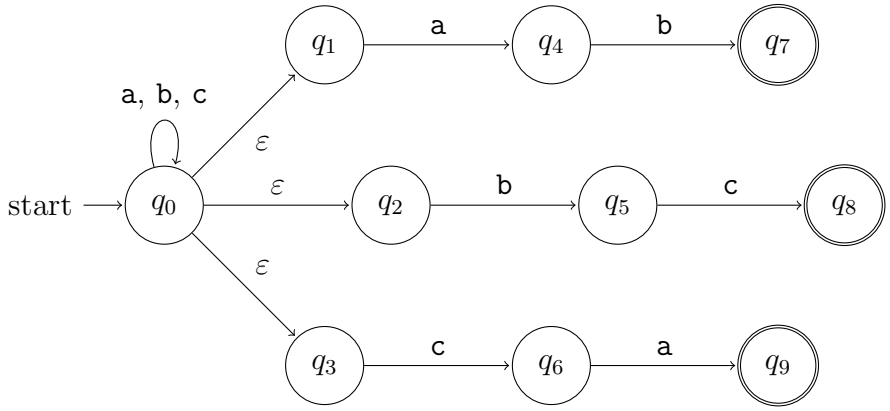


a) Describe the language using set-builder description using an English description.

b) Describe the language using only explicit sets and the language operations (*, $\cup$, and $\circ$).

Exercise 10

Determine the language of the following NFA, M, over alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$.

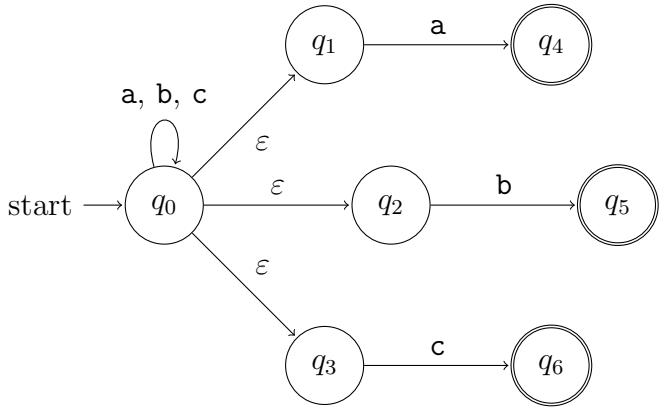


a) Describe the language using set-builder description using an English description.

b) Describe the language using only explicit sets and the language operations (*, $\cup$, and $\circ$).

Exercise 11

Determine the language of the following NFA, M, over alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$.



a) Describe the language using set-builder description using an English description.

b) Describe the language using only explicit sets and the language operations (*, $\cup$, and $\circ$).

Exercise 12

Use the NFA-building theorems from this section to build an NFA, M, so that $L(M) = \{\mathtt{c}\}^+$.

(Answer 1.1.12 in Appendix)

Exercise 13

Use the NFA-building theorems from this section to build an NFA, M, so that $L(M) = (\{\mathtt{b}\} \cup \{\mathtt{ab}\})^*$. Your answer should have four steps of automata. ($\{\mathtt{b}\}$, $\{\mathtt{ab}\}$, $\{\mathtt{b}\} \cup \{\mathtt{ab}\}$, and $(\{\mathtt{b}\} \cup \{\mathtt{ab}\})^*$)

Exercise 14

Use the NFA-building theorems from this section to build an NFA, M, so that $L(M) = \{\mathtt{a}\}^* \cup (\{\mathtt{c}\}^* \circ \{\mathtt{b}\})$.

Exercise 15

Use the NFA-building theorems from this section to build an NFA, M, so that $L(M) = \{w \mid w$ is either abc repeated any number of times, or an even-length string with no a's ending in bc$\}$. (Hint: first come up with the sublanguages, then put them together using the theorems.)

1.2 Pushdown Automata

This material is covered in Sipser in section 2.2. [1]

Remember, we couldn't recognize $L = \{\mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N}\}$. We'll now describe a new type of machine that can recognize languages like L.

Similar to FSA, but with a stack.

⟨ Draw the NPDA that recognizes $L = \{\mathtt{0}^n\mathtt{1}^n | n \in \mathbb{N}\}$ from above. Then talk about it. ⟩

⟨ Show the running of the automata on `aabb`. Then `aab` then `aabbb`. TODO: look up how to note the current "state" of a PDA. ⟩

Note: we're starting with Non-deterministic automata here.

Q: Draw the automata that recognizes $L = \{\mathtt{0}^n\mathtt{1}^{2n} | n \in \mathbb{N}\}$

A:

Q: Draw the automata that recognizes $L = \{0^m1^n | n < m\}$

A:

Q: Draw the automata that recognizes $L = \{0^m1^n | n > m\}$

A:

Q: What's the difference between Deterministic and Non-deterministic PDAs? Hint: we still want to be able to transition on ε:

- Instead of popping something off the stack,
- Instead of pushing something on the stack,
- and instead of reading in a new character.

A:

TODO: new subsubsection here?

Q: Can I write the formal definition of a Pushdown Automata as a 5-tuple?

A: No!

Q: Why not?

A: The language alphabet and the stack alphabet need to be separate.

Q: Why?

A:

Q: Here's the formal definition of a PDA: $(Q, \Sigma, \Gamma, \delta, q_0, F)$. What's each part?

A: TODO

TODO: add some examples in here!

Q: What's going to be different with a Deterministic PDA (DPDA)?

A:

Q: Let PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$. What's the condition that must be true in order for M to be a DPDA?

A:

Q: Can you have a DPDA that recognizes $\{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \mid i = j \text{ or } i = k\}$?

A: No.

Q: So what does that mean in terms of power?

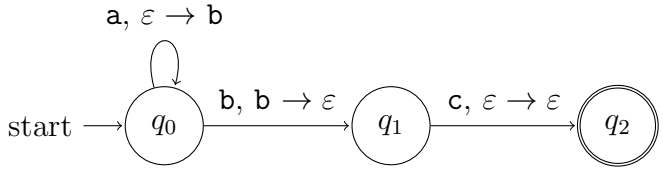
A:

TODO: add some examples or something. How much more do we really want to say about this?

Exercises for 1.2

Exercise 0

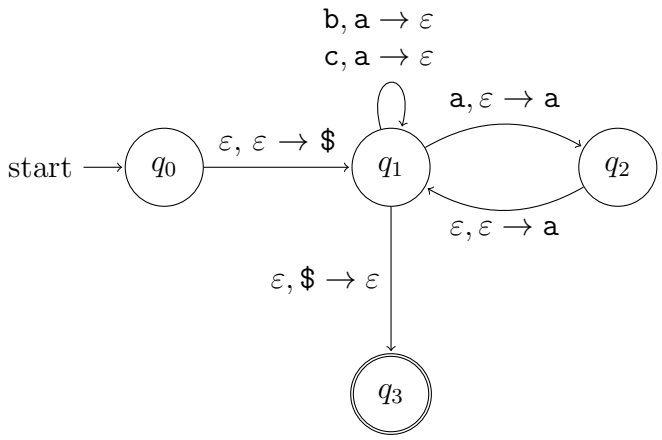
Give the formal (set notation) for the following PDA, P:



(Answer 1.2.0 in Appendix)

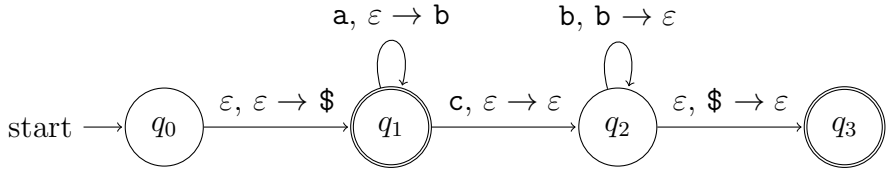
Exercise 1

Give the formal (set notation) for the following PDA, P:



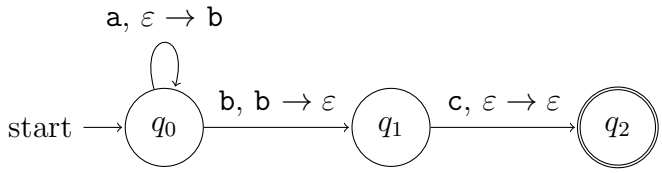
Exercise 2

Give the formal (set notation) for the following PDA, M:



Exercise 3

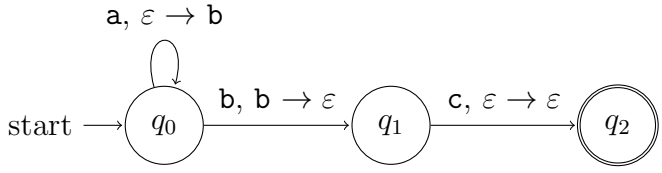
Describe the language, $L(P)$, for PDA P:



(Yes, this is the PDA from Exercise 1.2.0.)
(Answer 1.2.3 in Appendix)

Exercise 4

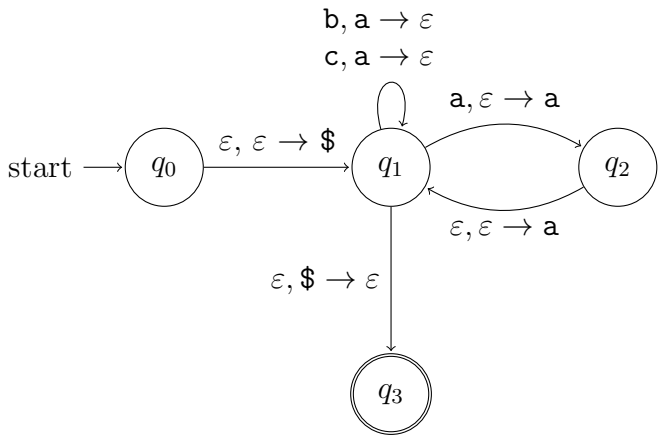
Create an NFA that recognizes the same language as PDA P:



(Yes, this again the same PDA as in Exercise 1.2.0.)

Exercise 5

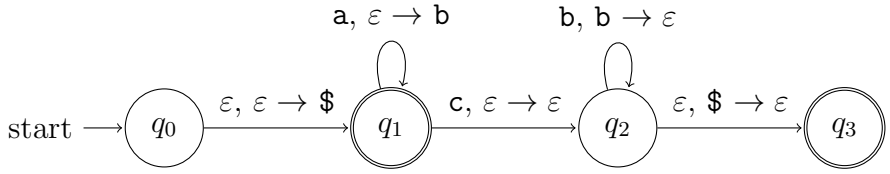
Describe the language, $L(P)$, for PDA M:



(Hint: Consider `((` instead of `a`. Also, yes, this is the PDA from Exercise 1.2.1.)

Exercise 6

Describe the language, $L(P)$, for PDA M:



(Hint: Yes, this is the PDA from Exercise 1.2.2.)

Exercise 7

Describe PDA $M(A)$ where $A = \{\texttt{ab}\}^* \circ \{\texttt{bc}\}^*$. Give either a figure or the set notation. (Answer 1.2.7 in Appendix)

Exercise 8

Describe PDA $M(A)$ where $A = \{(\texttt{ab})^i(\texttt{bc})^i \mid i \geq 0\}$. Give either a figure or the set notation.

Exercise 9

Describe PDA $M(A)$ where $A = \{\texttt{a}^i\texttt{bcbb}^i \mid i \geq 2\}$. Give either a figure or the set notation.

Exercise 10

Describe PDA $M(A)$ where $A = \{\texttt{a}^i\texttt{b}^j\texttt{c}^k\texttt{a}^k\texttt{b}^j\texttt{c}^i \mid i, j, k \in \mathbb{N}\}$. Give either a figure or the set notation.

1.3 Turing Machines

This material is covered in Sipser in section 3.1. [1]

Q:	So far all of our machines have done one of two things on any input. What are those two things?
A:	

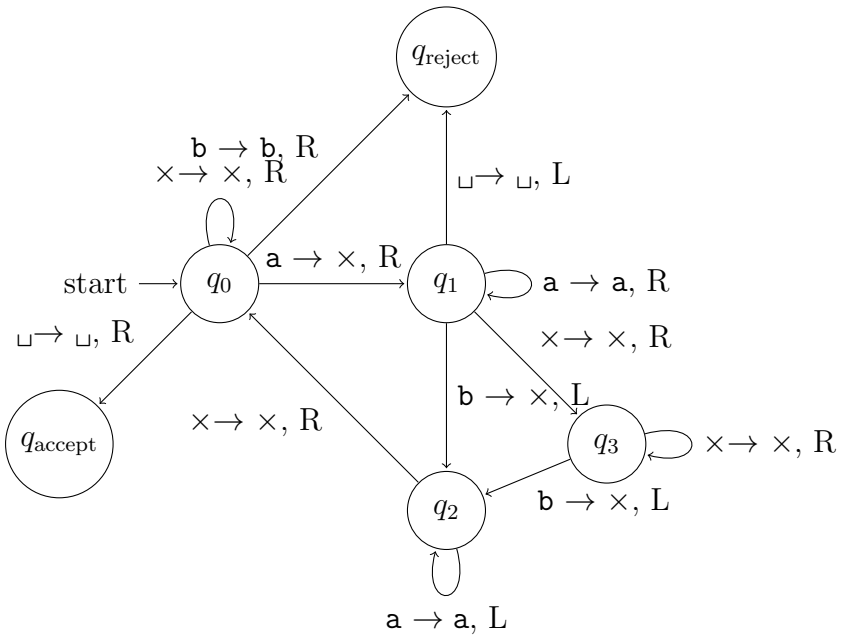
Q: What happens with actual computers?

A:

Q: What are some things that are going to be different about Turing Machines that will make this a possibility? (I don't expect them to get all of these.)

A:

- Can go back and forth on the input string.
- Can read *and write* the input. (So we call it a "tape" instead.)
- No stack, so we'll just use the tape. Thus, it must be unboundedly long on one side.
- Need another way to indicate when we're done, so the accept and reject states will be terminal (the machine stops when it reaches them).



Here's an example:

Q: Let's try running the above machine on an input string: `aaa`. What does that trial look like?

A: q_0`aaa` $\rightarrow \times q_0$`aa` ... TODO: finish this

Q: Let's try it out on `aabb`.

A:

Q: Let's try it out on `aabbb`.

A:

Q: What language does this TM recognize?

A:

Q: FSA could be described as a 5-tuple. PDAs as a 6-tuple. How big of a tuple do you need for a TM?

A:

Q: Okay, so what has to be in our formal definition? What are the seven things going to be?

A: Turing Machine is defined as a 7-tuple: $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where

- Q is the set of states
- $\Sigma \subsetneq \Gamma$ is the input alphabet and *cannot* include $\sqcup$
- Γ is the tape alphabet and $\sqcup \in \Gamma$
- $\delta : (Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}) \times \Gamma \to Q \times \Gamma \times \{\text{L}, \text{R}\}$ is the transition function.
- q_0 is the start state
- q_{accept} is the accept state, and
- q_{reject} is the reject state.

A turing machine that reaches the accept state on a language is said to *recognize* that language. A language is *turing-recognizable* if there's a TM that recognizes it.

Q: What does a TM do on the strings that it doesn't recognize?

A:

Q: What does looping mean?

A:

Q: Why didn't this happen in the previous machine types?

A:

Q: Are we happy with TMs that could loop indefinitely?

A:

A *decider* is a TM that doesn't loop. It always either accepts or rejects.

A *decideable* language is a language that is decided by a Turing Machine. We've already talked about undecideability a bit... it's the same concept.

Now do example 3.7. :-P ($\{\mathtt{0}^{2^k} \mid k \in \mathbf{N}\}$)

Exercises for 1.3

Exercise 0

Draw the diagram for a Turing Machine that decides the following language over alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$:

$\{w \mid$ The sum of the number of a's and b's in w is 8$\}$

(Answer 1.3.0 in Appendix)

Exercise 1

Draw the diagram for a Turing Machine that decides the following language, A, over alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$:

$A = \{w \mid w$ is in alphabetical order.$\}$

Hint: To clean up your figure, it's okay to leave edges out, then specify: "All missing transitions go to q_{reject}."

Exercise 2

Draw the diagram for a Turing Machine that decides the following language over alphabet $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$:

$\{w \mid$ The sum of the number of a's and b's in w is even.$\}$

Exercise 3

No PDA exists to recognize

$$L_0 = \{w \mid w \text{ contains an equal number of a's, b's, and c's}\}$$

. A Turing Machine does exist that decides L_0, but it is too complicated to include as a homework problem.

Instead, draw the diagram for a Turing Machine that decides

$$L_1 = \{w \mid w \text{ contains an equal number of a's and b's}\}$$

over the input alphabet $\{\mathtt{a}, \mathtt{b}\}$. There is a PDA that recognizes this, but it's still a worthwhile exercise.

(Answer 1.3.3 in Appendix)

Exercise 4

Draw the diagram for a turing machine that decides the following language over alphabet $\{\mathtt{a}, \mathtt{b}\}$:
$\{\mathtt{a}^n\mathtt{b}^{2n} \mid \forall n \geq 0\}$

Exercise 5

Draw the diagram for a turing machine that decides the following language over alphabet $\{\mathtt{a}, \mathtt{b}\}$:
$\{\mathtt{a}^n\mathtt{bab}^n \mid \forall n \geq 0\}$

2 Language Classes

2.0 Regular Languages

This material is covered in Sipser in section 1.3. [1]

A language is a *regular language* if there's a DFA that recognizes it.

A regular language is often described using a *regular expression*. Regular Expressions are used widely in computer science. Each corresponds to a regular language. Blahblahblah.

TODO: explain regular expressions here. Explain that they are generators for a language. Give a list of some of the special operations used by real-world regular expressions (e.g. Java).

TODO: new subsubsection: DFA and NFA Equivalence

TODO: move all the previous subset construction stuff here.

Q: If a language, A, is regular, is there an NFA that recognizes it?

A:

Q: Why?

A:

Q: Other direction: If there is an NFA, M and $L(M) = A$, does A have to be regular? (I don't expect them to know the answer to this.)

A: Yes

Q: Why?

A:

The details for subset construction are in the text book. All you need to know is that:

- It's deterministic (good)
- It can cause exponential blow-up in the number of states (bad)

TODO: Add subset construction stuff.

TODO: new subsubsection: Language Operations

TODO: move the NFA combination stuff here. Maybe move subset construction here too...

Union, concatenation, and the Kleene star are the three "normal" language operations.

Q: What other operations might we want to investigate?

A: Maybe complement? ... and intersection?

Q: If A is a regular language, what about A^C? Is it also regular?

A: Yes![8]

Q: Prove it! Hint: How do we know if a language is regular?

A:

Q: Okay, if A and B are regular, what about $A \cap B$?

A: Also regular.

[8] In Sipser, the notation $\overline{A}$ is used in place of A^C.

Q: Why? Hint: use the things you already know. Double hint: DeMorgan like a boss.

A:

Exercises for 2.0

Exercise 0

The following language is regular over the alphabet {a}:

$$L = \{w \mid \text{The number of a's in } w \text{ is congruent to 2 mod 5.}\}$$

Prove that L is regular by finding a regular expression equivalent to L.
(Answer 2.0.0 in Appendix)

Exercise 1

The following language is regular over the alphabet {a, b}:

$$L = \{w \mid w \text{ does not contain both a's and b's.}\}$$

Prove that L is regular by finding a regular expression equivalent to L.
(Answer 2.0.1 in Appendix)

Exercise 2

The following language is regular over the alphabet {a, b}:

$$L = \{w \mid w \text{ is not empty and if it contains any a's, it has no b's.}\}$$

Prove that L is regular by finding a regular expression equivalent to L.

Exercise 3

The following language is regular over the alphabet {a, b}:

$$L = \{w \mid w \text{ contains an odd number of a's and exactly one b.}\}$$

Prove that L is regular by finding a regular expression equivalent to L.

2.1 Context-Free Languages

This material is covered in Sipser in section 2.1. [1]

Q: Which kind of automata have the power to recognize context-free languages?

A: Pushdown Automata

Regular Languages are important because Regular Expressions are used in pattern recognition. Context-Free Languages are important in defining programming languages.

Just as we can use regular expressions to generate all strings in a regular language, *context-free grammars* can generate strings in context-free languages. Here's an example:

Context-Free Grammar for Python boolean expressions (where the only variables are X, Y, and Z).

$$
\begin{aligned}
< EXPRESSION > \rightarrow\ & (< EXPRESSION >) \\
| \ & \texttt{not} < EXPRESSION > \\
| \ & < EXPRESSION > \texttt{and} < EXPRESSION > \\
| \ & < EXPRESSION > \texttt{or} < EXPRESSION > \\
| \ & < VARIABLE > \\
| \ & < LITERAL > \\
< VARIABLE > \rightarrow\ & \texttt{X} \\
| \ & \texttt{Y} \\
| \ & \texttt{Z} \\
< LITERAL > \rightarrow\ & \texttt{True} \\
| \ & \texttt{False}
\end{aligned}
$$

Q: How does this work to generate expressions?

A:

Example, generate: `not (True and X)`

$$
\begin{aligned}
< EXPRESSION > &\rightarrow \texttt{not } < EXPRESSION > \\
&\rightarrow \texttt{not } (< EXPRESSION >) \\
&\rightarrow \texttt{not } (< EXPRESSION > \texttt{ and } < EXPRESSION >) \\
&\rightarrow \texttt{not } (< LITERAL > \texttt{ and } < EXPRESSION >) \\
&\rightarrow \texttt{not } (\texttt{True and } < EXPRESSION >) \\
&\rightarrow \texttt{not } (\texttt{True and } < VARIABLE >) \\
&\rightarrow \texttt{not } (\texttt{True and } < VARIABLE >) \\
&\rightarrow \texttt{not } (\texttt{True and X})
\end{aligned}
$$

We can use this to generate Parse Trees. What's the Parse Tree for this?
⟨ Draw the Parse Tree (Image TODO) ⟩

Each Context-Free language has a CF grammar that generates all the strings in it. Let's look at one of the first CF-languages we considered: $L = \{a^i b^i | i \in \mathbb{N}\}$.

Q: What is a CF grammar for L?

A:

TODO: add some parse tree examples for this.

Ambiguous grammars can have multiple parse trees for the same string. *Unambiguous grammars* cannot.

Q: Is the Python boolean expression grammar we came up with ambiguous or unambiguous?

A:

Q: Show an ambiguity. Hint: show a string that can be parsed in two different ways. Double hint: use both `and` and `or`.

A:

TODO: sme more stuff about ambiguity goes in here...

Q: Can we draw a (non-deterministic) PDA for our language L?

A:

Q: What about for the Python boolean expression language?

A:

Let's do it! TODO: include the figure.
Use book page 118 to explain how to build a PDA from a grammar.
Consider the following C-F Grammar:

$$A \to wAx \mid yB$$
$$B \to vBy \mid AC \mid C$$
$$C \to Cz \mid zz$$

Let's design a PDA to recognize the language generated by this grammar.

Q: What's the plan?

A:

Q: What if there are multiple options for that variable?

A:

Q: So, there will be one state where all the transitions leave based on the top element of the stack. What happens after we finish pushing those symbols backwards onto the stack?

A:

Q: How do we get to that main state from the start?

A:

Q: How do we finish?

A:

⟨ Do out the example above. ⟩

Exercises for 2.1

Exercise 0

Consider the following CFG G:

$$A \to \mathtt{z}A\mathtt{z} \mid \mathtt{y}B\mathtt{y}$$
$$B \to \mathtt{aaa} \mid \mathtt{z}C\mathtt{b}$$
$$C \to \mathtt{a} \mid A \mid \mathtt{bz}$$

(a) What are the variables of G?
(b) What is the start variable of G?
(c) What are the terminals of G?
(d) Give the derivation that shows $\mathtt{zyaaayz} \in L(G)$.
(e) Give the derivation that shows $\mathtt{zyaaayz} \in L(G)$.

(Answer 2.1.0 in Appendix)

Exercise 1

Consider the following CFG G:

$$A \to B\mathtt{x} \mid \mathtt{x}$$
$$B \to C\mathtt{xx}C \mid \mathtt{x}AB\mathtt{x} \mid \mathtt{xx}$$
$$C \to \mathtt{xxx} \mid \mathtt{x}A\mathtt{x}A\mathtt{x} \mid \mathtt{xx}C\mathtt{x}$$

(a) What are the variables of G?
(b) What is the start variable of G?
(c) What are the terminals of G?
(d) Give the derivation that shows $\mathtt{xxxxxxxxx} = \mathtt{x}^9 \in L(G)$.
(e) Give a derivation that shows $\mathtt{xxxxxxxxxxxxxx} = \mathtt{x}^{14} \in L(G)$. (There are multiple.)

Exercise 2

Consider the following CFG G:

$$A \to \mathtt{xyz} \mid \mathtt{x}B$$
$$B \to \mathtt{yzx} \mid \mathtt{y}C$$
$$C \to \mathtt{zxy} \mid \mathtt{z}A \mid \mathtt{zz}$$

(a) What are the variables of G?

(b) What is the start variable of G?

(c) What are the terminals of G?

(d) Give the derivation that shows $\texttt{xyzxyz} \in L(G)$.

(e) Give the derivation that shows $\texttt{xyzxyzz} \in L(G)$.

Exercise 3

Let G be the following context-free grammar:

$$C \to \texttt{a}C \mid \texttt{a}D\texttt{a} \mid \texttt{b}$$
$$D \to \texttt{b} \mid \varepsilon \mid C\texttt{aaa}$$

1. Show that $\texttt{aaaba} \in L(G)$.
2. Show that $\texttt{abba} \notin L(G)$.

(Answer 2.1.3 in Appendix)

Exercise 4

Consider the following context-free grammar, G:

$$X \to \texttt{a}X\texttt{b}Y \mid \texttt{c}$$
$$Y \to \texttt{c} \mid X\texttt{b}Z$$
$$Z \to \texttt{a} \mid \texttt{b}Y\texttt{b}$$

For each of the following strings, determine whether the string is in $L(G)$. If it is, show a derivation. If it isn't, show that all possible derivations fail to match.

(a) $\texttt{a}$

(b) $\texttt{c}$

(c) $\texttt{abc}$

Exercise 5

Consider the following context-free grammar, G:

$$X \to \texttt{a}X\texttt{b}Y \mid \texttt{c}$$
$$Y \to \texttt{c} \mid X\texttt{b}Z$$
$$Z \to \texttt{a} \mid \texttt{b}Y\texttt{b}$$

For each of the following strings, determine whether the string is in $L(G)$. If it is, show a derivation. If it isn't, show that all possible derivations fail to match. (Yes, this is the same grammar as in Exercise 2.1.4.)

(a) $\texttt{aabbcc}$

(b) $\texttt{acbc}$

(c) acbcbbcb

Exercise 6

Give a context-free grammar, G, that generates $L = \{w \mid w \text{ starts and ends with } \mathtt{0}\}$ over alphabet $\Sigma = \{\mathtt{0}, \mathtt{1}\}$.
(Answer 2.1.6 in Appendix)

Exercise 7

Give a context-free grammar, G, that generates $L = \{\mathtt{a}^{3n+1}\mathtt{b}^n \mid n \in \mathbb{N}\}$.

Exercise 8

Give a context-free grammar, G, that generates $L = \{\mathtt{a}^i\mathtt{b}^i\mathtt{cbac}^j\mathtt{b}^k\mathtt{c}^k\mathtt{abc}^j \mid i, j, k \in \mathbb{N}\}$.

Exercise 9

Create a PDA for the following grammar on $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. (Hint: Book Theorem 2.20 has the steps to follow on page 118.)

$$X \to X\mathtt{a} \mid Y\mathtt{a}$$
$$Y \to Y\mathtt{b} \mid Z$$
$$Z \to Z\mathtt{c} \mid \mathtt{c}$$

(Answer 2.1.9 in Appendix)

Exercise 10

Create a PDA for the following grammar on $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. (Hint: Book Theorem 2.20 has the steps to follow on page 118.)

$$A \to A\mathtt{ab} \mid \mathtt{c}B$$
$$B \to B\mathtt{ba} \mid \varepsilon$$

Exercise 11

Create a PDA for the following grammar on $\Sigma = \{a, b, c\}$. (Hint: Book Theorem 2.20 has the steps to follow on page 118.)

$$E \to aaEb \mid SS$$
$$S \to Sa \mid c$$

2.2 Decideable Languages

This material is covered in Sipser in section 4.1. [1]

TODO: draw superset figure: (((Unambiguous CFLs (Regular Languages)) CFLs) Decideable Languages)

A languages is *decideable* if there is a Turing Machine that decides it.

Q: What if we consider languages that are sets of machines? Consider the following language: $L = \{M \mid M \text{ is a DFA}\}$. How would we represent a single machine, M, as a string?

A:

Q: How would we represent δ? Picture?

A:

Q: Is this language decideable?

A:

Q: How do we know?

A:

Q: How does A work? Hint: It may help to think about it as a normal program, not a Turing Machine.

A:

Q: Cool. What if I want a language that contains DFAs and their input strings? $L = \{(M, w) \mid M \text{ is a DFA that accepts string } w\}$. Is L decideable?

A:

Q: There are three parts to this machine. (Let's call it B.) What's the first thing B needs to confirm?

A:

Q: How do we do that?

A:

Q: What's the next thing?

A:

Q: How do we do that?

A:

Q: What's the last part?

A:

Q: How do we do that?

A:

Q: Let's think about this part as a Turing Machine. How do we do the simulation?

A:

Q: Okay, what about NFAs? Is this decideable? $\{(M, w) \mid M \text{ is an NFA that accepts string } w\}$

A:

Q: Let's call the decider C. How does it work? Hint: remember subset construction!

A:

Q: Recall that each Regular Language can be generated by a Regular Expression. What if instead of specifying the machine, I specify the expression? Is $\{(E, w) \mid E \text{ is a regular expression and } w \text{ is a matching string}\}$ decideable?

A:

Q: Let's call this machine D. How does it work?

A:

Q: Let's try some questions where we don't include any input and just ask about a machine. What's the most basic question we could ask about a DFA?

A:

Q: Is this language decideable? $\{M \mid M \text{ is a DFA and it recognizes } \emptyset\}$

A:

Q: How does the decider work?

A:

Q: How does it do that?

A:

TODO: add questions here about whether two machines recognize the same language.

Okay, we've shown that answering questions about Regular-Language-Machines is decideable. We know from the beginning of the semester (Section 0.1) that we can't answer all questions about Turing Machines. Next, let's take a closer look at analyzing Pushdown Automata using Turing Machines...

Q: What do you think? Are basic questions about PDAs going to be decideable?

⟨ Talk about the complement of a language, $\overline{L} = L^C$. If L is decideable, what about $\overline{L}$? TODO: write this out. ⟩

TODO: add more here?

Exercises for 2.2

Exercise 0

Let $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. Let Σ-STAR be this language: $\{A \mid A \text{ is a DFA and } L(A) = \Sigma^*\}$. Show that Σ-STAR is decideable.

(Answer 2.2.0 in Appendix)

Exercise 1

Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ and ONLYA $= \{A \mid A \text{ is a DFA and } L(A) = \{\mathtt{a}\}^*\}$. Show that ONLYA is decideable.

(Answer 2.2.1 in Appendix)

Exercise 2

Let REJECTED = $\{(A, w) \mid A$ is an NFA and $w \notin L(A)\}$. Show that REJECTED is decideable.

Exercise 3

Let INTERSECT = $\{(A, B, w) \mid A$ and B are DFAs and $w \in L(A) \cap L(B)\}$. Show that INTERSECT is decideable.

Exercise 4

Let NFA-ACCEPT = $\{(A, w) \mid A$ is an NFA and $w \in L(A)\}$. Show that NFA-ACCEPT is decideable. (This is totally an example in the text book. TODO: add the citation here.)
(Answer 2.2.4 in Appendix)

2.3 Turing-Recognizeable Languages

This material is covered in Sipser in section 4.2. [1]

There are some languages that are recognized by Turing Machines, even if those machines aren't decided. A language is *Turing-recognizeable* if there is a Turing Machine that recognizes it, meaning a machine that always accepts when given a string in that language, and does not accept when given a string not in that language.

3 Non-Inclusion

When is a language not regular? When is it not context-free? When is it undecideable? There are some tools we can use to help sort languages out. If we can't find an automata, but we think one might exist, these parts can help us be more certain.

3.0 Non-Regular Languages

This material is covered in Sipser in section 1.4. [1]

There is a nice lemma[9] to help determine whether a language is regular: The *Pumping Lemma*:

If L is a regular language, then there exists an integer, $p \geq 1$ such that $\forall w \in L$: if $|p| \geq p$, then $\exists x, y, z : w = xyz$ where:

- $|y| \geq 1$,
- $|xy| \leq p$, and
- $\forall i \geq 0 : xy^i z \in L$

For any language, the smallest possible p where this holds is called the *pumping length* of that language.

Example, consider the language $L = \{\varepsilon, a, aa, aaa, aaaa, aaaaa\}$.

[9] A lemma is a baby theorem.

Q: Is there a DFA that accepts this language?

A:

Q: So is L regular?

A:

Q: Okay, so what should we be able to find out about L?

A:

Q: Great! What is it?

A:

Q: That seems like cheating. You just picked a length longer than any string in the language.

A:

This language is also certainly regular: $L = \{ab\}^*$. Let's break down verifying the regularity by using the pumping lemma!

The trick is to find an appropriate pumping length and decompositions for any string longer than that length. For L, we can use a pumping length of 2. The strings, $w \in L$, at least 2 characters long are all of L aside from ε, so $\{ab, abab, ababab, \ldots\}$.

Now, in the decomposition of w into xyz, only y needs to be non-empty. Notice that we can cover all of the acceptable strings by decomposing the following way:

- $w = \varepsilon$
- $y =$ ab
- $z =$ whatever comes after y

Now let's check whether we fit all of the pumping lemma requirements:

- $\forall i \in \mathbb{N} : xy^iz \in L$? ✓(These are just the other strings in L.)
- $|y| \geq 1$? ✓
- $|xy| \leq p = 2$? ✓

Sometimes you need to break things down into cases. Let's do another example. $L = \{ab\}^* \circ \{cd\}^*$.

TODO: do out this example.

TODO: another example. I think I took a picture.

TODO: Now do a negative example: $L = \{a^k b^k \mid k \in \mathbb{N}\}$

Exercises for 3.0

Exercise 0

What is the pumping length of $L = \{\mathtt{a}, \mathtt{aa}, \mathtt{ab}, \mathtt{bb}\}$?
(Answer 3.0.0 in Appendix)

Exercise 1

What is the pumping length of $L = \{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \mid i, j, k \leq 5\}$?

Exercise 2

Consider the following regular language:

$$L = \{\mathtt{a}\}^* \circ \{\mathtt{b}\}^*$$

(a) Choose a working p to satisfy the pumping lemma.

(b) For any string, w, longer than your chosen p, show how to divide it into x, y, and z. (You might need multiple cases.)

(c) Show that the three conditions of the decomposition hold.

(Answer 3.0.2 in Appendix)

Exercise 3

Consider the following regular language:

$$L = \{\mathtt{cb}\}^* \circ \{\mathtt{c}\} \circ \{\mathtt{a}\}^*$$

(a) Choose a working p to satisfy the pumping lemma for L.

(b) For any string, $w \in L$, longer than your chosen p, show how to divide it into x, y, and z. (You might need multiple cases.)

(c) Show that the three conditions of the decomposition hold.

Exercise 4

Consider the following regular language:

$$L = \{\mathtt{a}^i\mathtt{b}^j \mid i + j \text{ is even } \}$$

(a) Choose a working p to satisfy the pumping lemma.

(b) For any string, w, longer than p, show how to divide it into x, y, and z. (You might need multiple cases.)

(c) Show that the three conditions of the decomposition hold.

Exercise 5

Use the pumping lemma to show that the following language, L, is not regular.

$$L = \{a^{2^k} \mid \forall k \in \mathbb{N}\}$$

(Answer 3.0.5 in Appendix)

Exercise 6

Use the pumping lemma to prove that the following language is not regular.

$$L = \{\mathtt{a}^n\mathtt{b}^{2n} \mid n \in \mathbb{N}\}$$

Exercise 7

Use the pumping lemma to prove that the following language is not regular.

$$L = \{\mathtt{a}^n\mathtt{b}\mathtt{c}^n \mid n \in \mathbb{N}\}$$

3.1 Non-Context-Free Languages

This material is covered in Sipser in section 2.3. [1]

Q:	Okay, so what do we do if we want to show that a language is too hard for a PDA to recognize?
A:	

3.1.1 Pumping Lemma

Pumping Lemma for CFLs: If A is a context-free language, then $\exists p > 0 : \forall s \in A$, if $|s| \geq p$, then $\exists uvxyz = s$ where:

- $\forall i \geq 0 : uv^ixy^iz \in A$
- $|vy| > 0$, and

- $|vxy| \leq p$

As with Regular Languages, the smallest p is known as A's *pumping length*.

Let's apply the Pumping Lemma to $L = \{a^k b^k \mid k \in \mathbb{N}\}$.

Q: How should we divvy up any string $0^i 1^i$ into u, v, x, y, and z?

A:

TODO: ... more about this example. E.g. what is the pumping length? (2)
TODO: next example: $L = \{a^k bbbc^k \mid k \in \mathbb{N}\}$
TODO: next example: $L = \{a^m b^n c^m \mid m, n \in \mathbb{N}\}$
TODO: next example: $L = \{a^m b^n c^n d^m \mid m, n \in \mathbb{N}\}$
TODO: next example: $L = \{a^k b^m c^n \mid k = m \text{ or } k = n\}$

3.1.2 Showing Non-Context Free

This material is covered in Sipser in section TODO. [1]

TODO: show example: $L = \{a^{2^k} \mid k \in \mathbb{N}\}$

Use: Order of the elements doesn't matter, so...

Consider the case where: $s = a^8$, $uxz = a^4$, $v = aa$, and $y = aa$.

Notice, when $i = 0$, $|s| = 4$, which is okay.

When $i = 1$, $|s| = 8$, also okay.

But, when $i = 2$, $|s| = 12$, which doesn't work.

Let's generalize this to show that it never works. Proof by contradiction:

Assume L is context-free. Then the pumping lemma holds. Let p be the pumping length of L. Then let s be a string in L where $|s| > 2p$.

Let $m = |uxz|$ and $n = |vy|$. Then $m + i \times n$ is always a power of 2, $\forall i \in \mathbb{N}$.

Consider the first 4: $m = 2^b$, $m + n = 2^b 2^c = 2^{b+c}$, $m + 2n = 2^{b+c} 2^d = 2^{b+c+d}$, $m + 3n = 2^{b+c+d} 2^f = 2^{b+c+d+f}$, where $b, c, d, f > 0$.

Note: $c \geq 1 \rightarrow m + n = 2^{b+c} \geq 2^{b+1} = 2 \cdot 2^b = 2m$. Thus, $m + n \geq 2m$ and $n \geq m$.

At the same time, $d \geq 1 \rightarrow m + 2n = 2^{b+c+d} \geq 2^{b+c+1} = 2 \cdot 2^{b+c} = 2(m + n)$. Thus, $m + 2n \geq 2m + 2n$, meaning $m \geq 2m$, and $0 \geq m$. Thus, $m = 0$.

But, $|vxy| \leq p$, so $m = |uxz| \geq |uz| = |s| - |vxy| > 2p - |vxy| \geq 2p - p = p > 0$. Thus, $m > 0$.

Next example: $L = \{a^i b^i c^i \mid i \in \mathbb{N}\}$ Show that this is not context-free.

Exercises for 3.1

Exercise 0

Consider the following context-free language:

$$L = \{\mathtt{a}^k\mathtt{b}\mathtt{c}^k \mid k \in \mathbb{N}\}$$

(a) Choose a working p to satisfy the pumping lemma. (It does not need to be the shortest.)

(b) For any string, w, longer than your chosen p, show how to divide it into u, v, x, y, and z. (You might need multiple cases.)

(c) Show that the three conditions of the pumping lemma decomposition hold for your divisions.

(Answer 3.1.0 in Appendix)

Exercise 1

Consider the following context-free language:

$$L = \{\mathtt{a}^k\mathtt{b}^m\mathtt{c}^k \mid k, m \in \mathbb{N}\}$$

(a) Choose a working p to satisfy the pumping lemma. (It does not need to be the shortest.)

(b) For any string, w, longer than your chosen p, show how to divide it into u, v, x, y, and z. (You might need multiple cases.)

(c) Show that the three conditions of the pumping lemma decomposition hold for your divisions.

Exercise 2

Consider the following context-free language:

$$L = \{\mathtt{a}^k\mathtt{b}^m\mathtt{c}^m\mathtt{d}^k \mid k, m \in \mathbb{N}\}$$

(a) Choose a working p to satisfy the pumping lemma. (It does not need to be the shortest.)

(b) For any string, w, longer than your chosen p, show how to divide it into u, v, x, y, and z. (You might need multiple cases.)

(c) Show that the three conditions of the pumping lemma decomposition hold for your divisions.

Exercise 3

(Should I even include this one? I do it in class. I guess I'm keeping this right now because the answer in the back is more detailed than in the notes.) Use the Pumping Lemma for Context-Free Languages to show that this language is not context free:

$$L = \{\mathtt{a}^{2^n} \mid n \in \mathbb{N}\}$$

(Answer 3.1.3 in Appendix)

Exercise 4

Use the Pumping Lemma for Context-Free Languages to show that L is not context free, where

$$L = \{\mathtt{a}^n\mathtt{b}^m\mathtt{c}^n\mathtt{d}^m | \ n, m \in \mathbb{N}\}$$

(Answer 3.1.4 in Appendix)

Exercise 5

Use the Pumping Lemma for Context-Free Languages to show that this language is not context free:

$$L = \{\mathtt{a}^n\mathtt{b}^m\mathtt{c}^n \mid m > n\}$$

Exercise 6

Use the Pumping Lemma for Context-Free Languages to show that L is not context free, where

$$L = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^{2n} | \ n \in \mathbb{N}\}$$

3.2 Showing Undecideability (Reductions)

This material is covered in Sipser in section 5.1. [1]

Let $L = \{(p, x) \mid p$ is a basic python program that, on input x, prints out the number $5\}$.[10]

Q:	L is undecideable. How can we prove this?
A:	

Q:	How do we do that?
A:	

[10] By "basic", I mean, it doesn't import any packages. Here, I specifically don't want it to import `sys` then call `sys.exit()`. Thanks to Dale Skrien for pointing out this issue!

Q: What if we showed that if there was a TM, M, that decided it, then it could also decide the Halting Problem?

A:

Q: What kind of a proof is this?

A:

For simplicity, we'll just use a python program for M instead of a Turing Machine. I'm going to be a bit more restrictive and assume that the following language is undecideable as a result of the halting problem:

$$\textsc{PythonHalt} = \{(p, x) \mid p \text{ is a Python program that, on input } x, \text{ halts.}\}$$

From there, we'll build a new program, H, that decides PythonHalt.

Q: What's our first step?

A:

Okay, create a new TM, $H(k, y)$ (k is the input python program, y is the input to k):

1. Create a new Python program, n, which is just k but with the following line added to the end:
   `print(5) #Apparently we're using Python 3.`
2. Run $M(n, y)$
3. Then $\begin{cases} \text{accept}, & M(n, y) \text{ accepts} \\ \text{reject}, & M(n, y) \text{ rejects} \end{cases}$

Q: What does $H(k, y)$ do if $k(y)$ halts?

A:

Q: What does $H(k, y)$ do if $k(y)$ doesn't halt?

A:

Q: Why might it accept?

A:

Q: Okay, we're halfway there. Luckily, in Python, there's a way to stifle the output. How do we do that?

A:

```
import sys
oldStdout = sys.stdout #save the old stdout
fakeOut = io.StringIO() #will store instead of
printing
sys.stdout = fakeout #redirect to storage
```

Q: Okay, when do we do this?

A:

Q: What do we have to do afterwards?

A:

Q: What's the entire program we generate from k?

A:

Q: There's still a problem here. What is it? Hint: worst possible case for division.

A:

Q: Can we fix this?

A:

Q: What's the code so far we've generated from k?

A:

```
import sys
oldStdout = sys.stdout #save the old stdout
fakeOut = io.StringIO() #will store instead of
printing
sys.stdout = fakeout #redirect to storage
try:
    # k goes here!
except:
    pass #this does nothing
sys.stdout = oldStdout
print(5)
```


Q: There's still a problem here. What's that problem?

A:

Q: How do we solve this problem?

A:

That function we added to transform the problem instances is called a *reduction*. If we can *reduce* from language A to language B, that means that there's a function, f, where:

- $f : \Sigma^*(A) \rightarrow \Sigma^*(B)$, and
- $w \in A \Leftrightarrow f(w) \in B$

The reduction is exactly the function f. We say, "f is a reduction from A to B".

Q: If we show a reduction from an undecideable language, U, to some other language, V, what must be true of V?

A:

Q: Why?

A:

Q: There's one important point about this. We have to know something specific about f to make this work.

A: f has to be *computable*. You can't base it off of an undecideable property or something like that. Usually this is not an issue.

Q: Okay, now what are our three properties of a reduction, f, from language A to language B?

A:
- $f : \Sigma^*(A) \to \Sigma^*(B)$, and
- $w \in A \Leftrightarrow f(w) \in B$, and
- f is computable.

In order for f to be a reduction, all three of those must be true.

Q: If f is a reduction from A to B, what do we know in the following four cases?
- A is undecideable. (What does this tell us about B?)
- A is decideable. (What about B?)
- B is undecideable. (What about A?)
- B is decideable. (What about A?)

What if f goes the other way, from V to U? What can we say about V now?

A:

Q: Why does this last one work that way?

A: We just use f and the decider for B to decide A.

I have names for the first and fourth properties:
- A undecideable $\to$ B undecideable: "Hardness follows a reduction"
- B decideable $\to$ A decideable: "Easiness salmons a reduction"[11]

Exercises for 3.2

Exercise 0

[11] Salmons swim upstream. If this doesn't work for you, you can use "Easiness flows against a reduction" or "Easiness goes against a reduction" instead.

Find a reduction, f from A to B, where:

$$A = \{\mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N}\}$$

, and

$$B = \{\mathtt{a}^n\mathtt{b}^m\mathtt{c}^m\mathtt{d}^n \mid n, m \in \mathbb{N}\}$$

Assume that both languages are over the same alphabet: $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$.
(Answer 3.2.0 in Appendix)

Exercise 1

Consider these two languages:

$$A = \{w \mid w \text{ is an even number represented in binary}\}$$

, and

$$B = \{w \mid w \text{ is in alphabetical order}\}$$

The alphabets for these are $\Sigma(A) = \{\mathtt{0}, \mathtt{1}\}$ and $\Sigma(B) = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. Find a reduction, f, from A to B.

Exercise 2

Consider these two languages over the same alphabet $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$:

$$A = (\{\mathtt{ab}\} \cup \{\mathtt{cb}\})^*$$

, and

$$B = \{\mathtt{ac}\}^* \circ \{\mathtt{bbb}\}$$

Find a reduction, f, from A to B.

Exercise 3

Consider the following language:
YesBeluga = $\{M \mid M$ is a Java program that prints the string `Beluga` to the console$\}$
Show that YesBeluga is undecideable.
(Answer 3.2.3 in Appendix)

Exercise 4

You can run code in many languages that reformats (meaning, deletes) a hard drive. For example, in Java, run this code to reformat a Windows drive:[13] (I haven't actually tried this out, but I think it formats the C drive. For the purposes of this, let's assume that's the case.)

```
Process p = Runtime.getRuntime().exec("CMD /C format f:
/FS:NTFS /Q /X /Y");
```

[13]Source: `http://stackoverflow.com/questions/20491849/formatting-fat32-usb-drive-on-windows` Don't actually run this code unless you really want to delete all of your files!

Consider the following language:
ReformatsYou = $\{(X, w) \mid X$ is a Java program that, on input w, reformats the `C` drive.$\}$.
Show that ReformatsYou is undecideable.
Hint: for my solution, I assumed that I could create a Virtual Machine and execute Java code inside of that.

Exercise 5

Show that the following language, VisitsState, is undecideable.

$$\text{VisitsState} = \{(M, q, w) \mid \text{TM } M \text{ run on input } w \text{ visits state } q\}$$

(Answer 3.2.5 in Appendix)

Exercise 6

Show that the following language, VisitsThreeStates, is undecideable.

$$\text{VisitsThreeStates} = \{(M, w, q_a, q_b, q_c) \mid \text{TM } M \text{ run on input } w \text{ visits all states } q_a, q_b, \text{ and } q_c\}$$

If you can't complete a formal proof, write a "proof sketch" or try writing it out thoroughly in words. If you can explain how this proof will differ from the previous one, that will get you most of the way there.

Exercise 7

Show that the following language, VisitsAllStates, is undecideable.

$$\text{VisitsAllStates} = \{(M, w) \mid \text{TM } M \text{ run on input } w \text{ visits all states aside from } q_{reject}\}$$

If you can't complete a formal proof, write a "proof sketch" or try writing it out thoroughly in words.

Exercise 8

Consider the following language: NothingUseless $= \{M \mid M$ is a TM with no unreachable states$\}$. Show that NothingUseless is undecideable. (An unreachable state is one that is never visited on any input string.)

3.3 Turing Unrecognizeability

This material is covered in Sipser in section 4.2. [1]

3.3.1 Different Infinities

Q:	How many Turing Machines are there?
A:	Infinitely-many

Q: How many languages are there over the alphabet $\Sigma = \{\texttt{a}, \texttt{b}\}$?

A: Also infinitely-many

Q: Are there any languages that aren't Turing-Recognizeable?

A: Maybe

Let's take a closer look at these infinities...

Q: How big is the set $\mathbb{N}$?

A: Infinitely big... but considered the "smallest infinity".
We say this is *countably* infinite.

Q: If $A \subseteq B$, what does that say about the sizes of A and B?

A:

Q: What about...

- $H = \{\frac{a}{b} \mid a, b \in \mathbb{N}\}$? ($H$ for Halves)
- $E = \{s \mid s \text{ is an even number}\}$? ($E$ for evens)

A:

- $\{\frac{a}{b} \mid a, b \in \mathbb{N}\}$ seems to be bigger than $\mathbb{N}$ (twice as big)
- $\{s \mid s \text{ is an even number}\}$ seems to be smaller than $\mathbb{N}$ (half as big).

So, it seems like $|E| < |\mathbb{N}| < |H|$

Q: If I can show an *onto* function $f : A \to B$, what does that say about the sizes of A and B?

A:

Q: So what does that mean if both are true? ($A \subseteq B$ and $\exists f : A \to B$ that's onto)

A:

Q: Can someone find an onto function $f : \mathbb{N} \to \mathbb{N} \cup \{-1\}$

A:

Q: What does that mean?

A:

So they're both countably infinite! Even though one is a strict subset of the other, they have the same "size". That word is a little weird... when we're talking about infinities, we use the term *cardinality* instead of size.

Let's consider some other sets:

- $\mathbb{Z}$
- H
- E
- $\mathbb{R}$
- $\mathbb{Q}$
- $\overline{\mathbb{Q}}$

Q: What's the cardinality of $\mathbb{Z}$? Hint: find a function

A:

Q: What about H?

A:

Q: What about E? Hint: onto $f : \mathbb{N} \to E$ is the wrong direction. $E \subseteq \mathbb{N}$, not the other way around.

A:

Q: What about $\mathbb{R}$?

⟨ Let them suggest some ideas for a bit before revealing... ⟩
$\mathbb{R}$ is not countable: *uncountable*.
⟨ Show Cantor's diagonalization argument. ⟩[14]

Q: What do you think about $\mathbb{Q}$?

A: It's countable. You can explain here.

Q: What about $\overline{\mathbb{Q}}$

A: Well, this has to be uncountable then!

[14]Good video: `https://www.youtube.com/watch?v=dEOBDIyz0BU`

3.3.2 Countability of Computing

Q: What do we do if $A \not\subseteq B$ and $B \not\subseteq A$? We still want a relationship between $|A|$ and $absB$... how does an onto $f : A \to B$ help?

A: It still shows that $|A| \geq |B|$

Q: So how can we show that they're equal?

A:

Q: What's another option with just one function?

A:

Q: Let's find the cardinality of $\Sigma = \{0, 1\}^*$. First, find an onto $f : \Sigma \to \mathbb{N}$

A:

Q: Now find an onto $g : \mathbb{N} \to \Sigma$. Note: can't just convert to binary; you need to hit the strings that start with lots of zeroes!

A:

Q: So what's true about the cardinality of $\Sigma^* = \{0, 1\}^*$?

A:

Q: Think about a language L over alphabet $\Sigma = \{0, 1\}$. What are the possible cardinalities of L?

A:

Q: What about $\Sigma = \{a, b\}$?

A:

Q: What if we add elements to Σ? E.g. $\{a, b, c\}$

A:

Q: So, over any finite alphabet, what is the cardinality of L?

A: Finite or countably infinite

Q: Now let's try to figure out the cardinality of the set of Turing Machines over some finite Σ. What is true about the size of the number of states in a Turing Machine?

A: Finite

Q: What about the size of the tape alphabet Γ?

A:

Q: So what is the size of the set of *all* Turing Machines? (Let's call this set $\mathcal{M}$)

A:

Q: What about $\mathcal{S}$, the set of all languages over some $\Sigma = \{\mathtt{a}, \mathtt{b}\}$?

A: There is no onto $f : \mathbb{N} \to \mathcal{S}$

Q: How is this possible?

A: Consider the following diagonalization argument. Assume onto f. Now consider the following language $L = \{\mathtt{a}^k \mid f(k) \text{ does not contain a string of length } k\}$

Q: What's the result of all of this?

A: There are more languages than TMs! That means there are languages that aren't Turing-Recognizeable!

Let's try to find one of these languages that is not Turing-Recognizeable! First we have to take a look back at language complements, i.e. L^C. For some language, L, if L^C is recognizeable, then we say that L is *co-Turing recognizeable*.

Q: What if L is both Turing-recognizeable and co-Turing recognizeable?

A: Then it must be decideable

Q: How do we know that?

A:

Q: What then must be true of L^C?

A:

⟨ Add co-Turing recognizeable to the subsets drawing so that the intersection of that and Turing-recognizeable is exactly the decideable languages. ⟩

Q: Consider a language, L, that is not decideable, but is Turing-recognizeable. What must be true about L^C?

A:

Q: Why?

A:

Q: Okay, so what is a language that is *not* Turing-recognizeable?

A:

4 Computational Complexity

4.1 NP-Completeness

This material is covered in Sipser in section 7.4. [1]

TODO: questionify this

⟨ Show the P v NP video (same one as 2381): `https://youtu.be/YX40hbAHx3s` ⟩

In the beginning there was chaos: lots of problems that we couldn't solve efficiently, but we could check efficiently!

Talk about each of these:

- Hamiltonian Cycle (and TSP?)
- Max Cut
- Independent Set
- VERTEX-COVER: smallest set of vertices that is incident to all edges.
- Subset Sum (and Knapsack?)
- Partition Problem
- ...?

Note: we can verify solutions in polynomial time.

I.e., they can be solved in polynomial time on a Non-deterministic Turing Machine.

Then, out of the chaos, 3SAT emerged and NP was defined.

Cook-Levin Theorem:

⟨ Talk more about this stuff. You know what you're doing. ⟩

4.2 Approximation Algorithms

This material is covered in Sipser in section 10.1. [1]

Q:	Your boss comes up to you and wants you to write a perfect scheduling algorithm for meetings. Should your response be that this is impossible?
A:	Maybe. Maybe not. What if you couldn't get it perfectly right, but you could get pretty close. What if you couldn't schedule everything, but you could schedule 98% of the meetings?

Q: For a maximization problem, if the optimal (max) value is OPT, then a k-approximation algorithm is one that always returns a value between $k \times OPT$ and OPT. In the example above, your algorithm would be a .98-approximation algorithm.
How are approximations for minimization problems described?

A:

Q: For minimization, is k greater or less than 1?

A:

Example: MAX-CUT, which is NP-hard. There is an α-approximation algorithm (where $\alpha \approx .868$) by Goemanns and Williamson[15][**?**]
This algorithm is really cool! ... add in the details...

Q: Consider all of the following problems. Are they minimization or maximization problems? What property is being approximated? If necessary, give them a new name that makes sense.

- Hamiltonian Cycle
- Independent Set
- Vertex Cover
- Subset Sum

A:

[15] A good summary of the details is here: `http://www.sfu.ca/~mdevos/notes/semidef/GW.pdf`

Q: There is a weird 2-approximation algorithm for Vertex Cover. See if you can figure it out. Hint: order doesn't matter. Double hint: just take both.

A:

Q: Why is that a 2-approximation?

A:

TODO: mention Polynomial Time Approximation Schemes

4.3 Hardness of Approximation

TODO

More information here: `http://www.cs.berkeley.edu/~luca/pubs/inapprox.ps%0a`

A Answers to Exercises

Answer of exercise 0.4.0

Question:

Write math-notation descriptions of each of the following sets. You do not need to show any work for these.

a) The set containing only the numbers -10, -3, 3, and 9.

b) The set of all even integers.

c) The set of all rational numbers ($\mathbb{Q}$) greater than or equal to 20.

d) The set of all rational numbers where the denominator is at most 2.

Answer:

a) $\{-10, -3, 3, 9\}$

b) $\{2k \mid k \in \mathbb{Z}\}$ LaTeX: `$\{2k\ |\ k \in \mathbb{Z}\}$`

c) $\{n \in \mathbb{Q} \mid n \geq 20\}$

d)

$$\left\{\frac{a}{b} \mid a \in \mathbb{Z} \wedge b \in \{1, 2\}\right\}$$

Answer of exercise 0.4.1

Answer Not Provided

Answer of exercise 0.4.2

Answer Not Provided

Answer of exercise 0.4.3

Question:
Let $S = \{\mathtt{a}, \mathtt{b}\}$ and $T = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. Replace the question marks in the following expressions:

a) $S \ ? \ T$ (Replace the ? with $=$, $\subset$, or $\supset$. Note that for some pairs of sets, none of these three is correct.)

b) $S \cup T =$? (Include the set with all elements.)

c) $S \cap T =$? (Ditto.)

d) $T \setminus S =$? (Set-difference)

e) $S \times T =$? (Cross-product)

f) $\mathcal{P}(\mathcal{S}) =$? (Power set)

Answer:

a) $S \subset T$

b) $S \cup T = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\} = T$

c) $S \cap T = \{\mathtt{a}, \mathtt{b}\} = S$

d) $T \setminus S = \{\mathtt{c}\}$

e) $S \times T = \{(\mathtt{a}, \mathtt{a}), (\mathtt{a}, \mathtt{b}), (\mathtt{a}, \mathtt{c}), (\mathtt{b}, \mathtt{a}), (\mathtt{b}, \mathtt{b}), (\mathtt{b}, \mathtt{c})\}$

f) $\mathcal{P}(\mathcal{S}) = \{\emptyset, \{\mathtt{a}\}, \{\mathtt{b}\}, \{\mathtt{a}, \mathtt{b}\}\}$

Answer of exercise 0.4.4

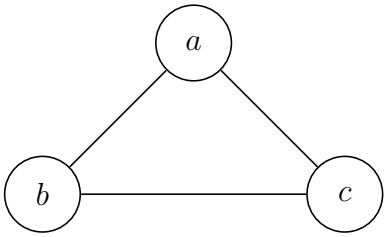
Answer Not Provided

Answer of exercise 0.4.5

Answer Not Provided

Answer of exercise 0.4.6

Question:
Write out the set description of this graph (the formal version, meaning $G = (V, E) = \ldots$):



Answer:
$G = (V, E) = (\{a, b, c\}, \{\{a, b\}, \{a, c\}, \{b, c\}\})$
or, preferably
$G = (V, E)$, where

- $V = \{a, b, c\}$, and
- $E = \{\{a, b\}, \{a, c\}, \{b, c\}\}$

Answer of exercise 0.4.7

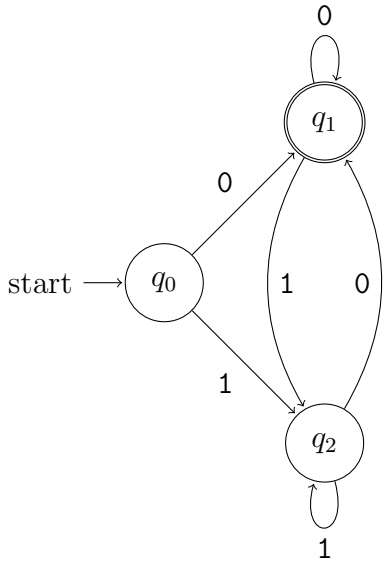
Answer Not Provided

Answer of exercise 0.4.8

Answer Not Provided

Answer of exercise 1.0.0

Question:
Write out the following DFA, D, using formal (set) notation. (You may use a table for the transition function.)



Answer:
$D = (Q, \Sigma, \delta, q_0, F)$, where:

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{0, 1\}$

- $$\begin{aligned}\delta = \{&(q_0, \mathtt{0}, q_1),\\ &(q_0, \mathtt{1}, q_2),\\ &(q_1, \mathtt{0}, q_1),\\ &(q_1, \mathtt{1}, q_2),\\ &(q_2, \mathtt{0}, q_1),\\ &(q_2, \mathtt{1}, q_2)\}\end{aligned}$$

  or

δ	0	1
q_0	q_1	q_2
q_1	q_1	q_2
q_2	q_1	q_2

- $F = \{q_1\}$

Answer of exercise 1.0.1

Answer Not Provided

Answer of exercise 1.0.2

Answer Not Provided

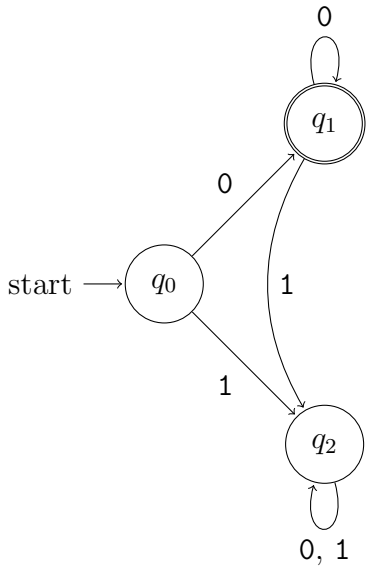
Answer of exercise 1.0.3

Question:
Draw the automata for DFA $D = (Q, \Sigma, \delta, q_0, F)$, where:

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{\mathtt{0}, \mathtt{1}\}$
- $$\begin{aligned}\delta = \{&(q_0, \mathtt{0}, q_1),\\ &(q_0, \mathtt{1}, q_2),\\ &(q_1, \mathtt{0}, q_1),\\ &(q_1, \mathtt{1}, q_2),\\ &(q_2, \mathtt{0}, q_2),\\ &(q_2, \mathtt{1}, q_2)\}\end{aligned}$$

- $F = \{q_1\}$

Answer:



Answer of exercise 1.0.4

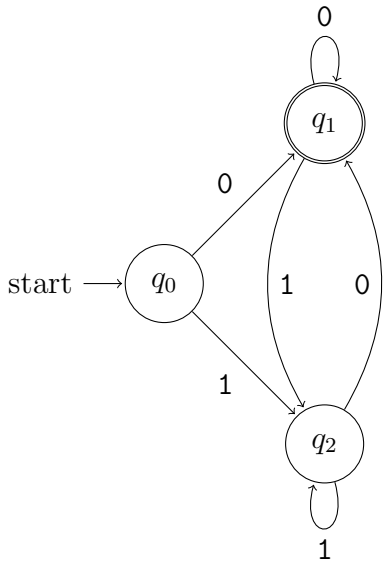
Answer Not Provided

Answer of exercise 1.0.5

Answer Not Provided

Answer of exercise 1.0.6

Question:
Formally describe the language of the machine, D, from Exercise 1.0.0:



Answer:
$L(D) = \{w \mid w \text{ ends with a } \mathtt{0}\}$

Answer of exercise 1.0.7

Answer Not Provided

Answer of exercise 1.0.8

Answer Not Provided

Answer of exercise 1.0.9

Question:
Formally describe the language of DFA $D = (Q, \Sigma, \delta, q_0, F)$, where:

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{\mathtt{0}, \mathtt{1}\}$
- $$\begin{aligned}\delta = \{&(q_0, \mathtt{0}, q_1),\\ &(q_0, \mathtt{1}, q_2),\\ &(q_1, \mathtt{0}, q_1),\\ &(q_1, \mathtt{1}, q_2),\\ &(q_2, \mathtt{0}, q_2),\\ &(q_2, \mathtt{1}, q_2)\}\end{aligned}$$
- $F = \{q_1\}$

(This is the machine from Exercise 1.0.3.)
Answer:
$L(D) = \{0\}^+$

Answer of exercise 1.0.10

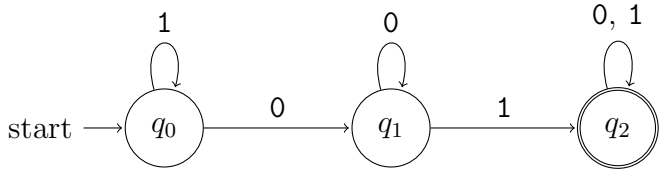
Answer Not Provided

Answer of exercise 1.0.11

Answer Not Provided

Answer of exercise 1.0.12

Question:
Describe the DFA that recognizes the following language, $L \subset \{\mathtt{0}, \mathtt{1}\}^*$:
$\{w \mid w$ contains a $\mathtt{1}$ and a $\mathtt{0}$ somewhere before that $\mathtt{1}\}$
Use either a figure or set notation to describe the automata.
Answer:
Figure solution:



or, set solution:
$D = (Q, \Sigma, \delta, q_0, F)$, where:

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{\mathtt{0}, \mathtt{1}\}$

- $$\begin{aligned}\delta = \{&(q_0, \texttt{0}, q_1),\\ &(q_0, \texttt{1}, q_0),\\ &(q_1, \texttt{0}, q_1),\\ &(q_1, \texttt{1}, q_2),\\ &(q_2, \texttt{0}, q_2),\\ &(q_2, \texttt{1}, q_2)\}\end{aligned}$$

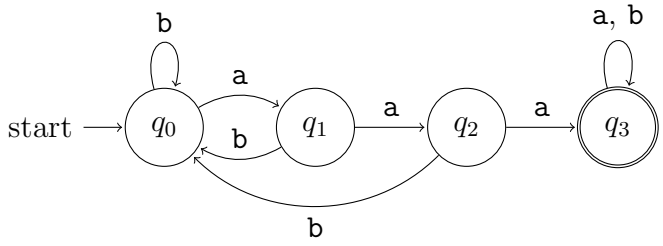
  or

δ	0	1
q_0	q_1	q_0
q_1	q_2	q_1
q_2	q_2	q_2

- $F = \{q_2\}$
  Naturally, there are lots of other possible machines that accept this language.

Answer of exercise 1.0.13

Question:
Describe the DFA that recognizes the following language $L \subset \{\texttt{a}, \texttt{b}\}^*$:
$\{w \mid w$ contains the substring aaa$\}$
Use either a figure or set notation to describe the automata.
Answer:
Figure solution:



or, set solution:
$D = (Q, \Sigma, \delta, q_0, F)$, where:

- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{\texttt{a}, \texttt{b}\}$

- $$\begin{aligned}\delta = \{&(q_0, \mathtt{a}, q_1),\\ &(q_0, \mathtt{b}, q_0),\\ &(q_1, \mathtt{a}, q_2),\\ &(q_1, \mathtt{b}, q_0),\\ &(q_2, \mathtt{a}, q_3),\\ &(q_2, \mathtt{b}, q_0),\\ &(q_3, \mathtt{a}, q_3),\\ &(q_3, \mathtt{b}, q_3)\}\end{aligned}$$

  or

δ	a	b
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_3	q_0
q_3	q_3	q_3

- $F = \{q_3\}$
  Naturally, there are other equivalent machines.

Answer of exercise 1.0.14

Answer Not Provided

Answer of exercise 1.0.15

Answer Not Provided

Answer of exercise 1.0.16

Answer Not Provided

Answer of exercise 1.0.17

Answer Not Provided

Answer of exercise 1.0.18

Answer Not Provided

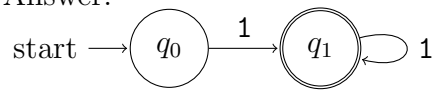
Answer of exercise 1.1.0

Question:
Draw the figure corresponding to the NFA $N = (Q, \Sigma, \delta, q_0, F)$, where:

- $Q = \{q_0, q_1\}$
- $\Sigma = \{\mathtt{0}, \mathtt{1}\}$
- $$\begin{aligned}\delta = \{&(q_0, \mathtt{1}, q_1),\\ &(q_1, \mathtt{1}, q_1)\}\end{aligned}$$

- $F = \{q_1\}$

Answer:



Answer of exercise 1.1.1

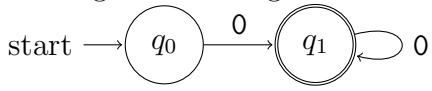
Question:
Using only *two transitions*, draw the diagram for an NFA equivalent to the following DFA:
DFA $D = (Q, \Sigma, \delta, q_0, F)$, where:

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{\mathtt{0}, \mathtt{1}\}$
-

$$\begin{aligned}\delta = \{&(q_0, \mathtt{0}, q_1),\\ &(q_0, \mathtt{1}, q_2),\\ &(q_1, \mathtt{0}, q_1),\\ &(q_1, \mathtt{1}, q_2),\\ &(q_2, \mathtt{0}, q_2),\\ &(q_2, \mathtt{1}, q_2)\}\end{aligned}$$

- $F = \{q_1\}$

(This is the DFA from Exercise 1.0.3.)
Answer:
Meeting the challenge:



Answer of exercise 1.1.2

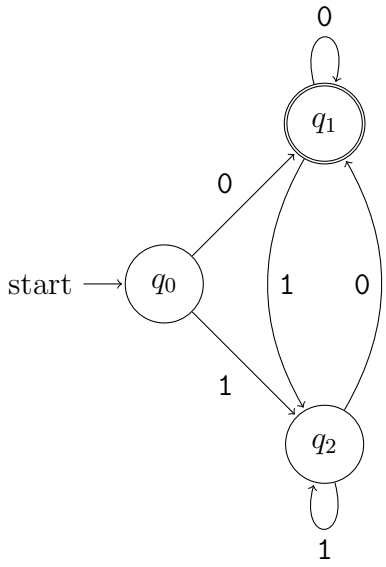
Answer Not Provided

Answer of exercise 1.1.3

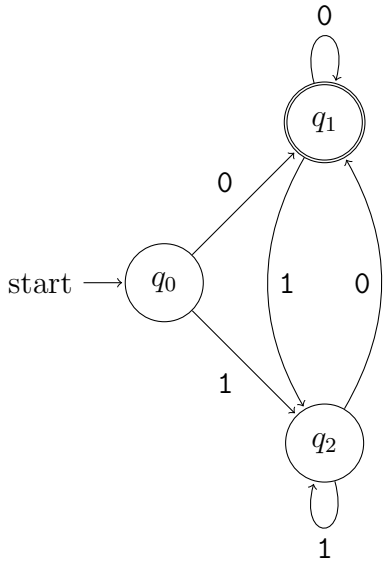
Answer Not Provided

Answer of exercise 1.1.4

Question:
Draw the NFA with the smallest number of transitions equivalent to the following DFA:



(This is the DFA from Exercise 1.0.0.)
Answer:
There is no smaller equivalent machine, so the smallest NFA is the exact same DFA:

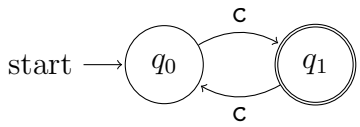


Answer of exercise 1.1.5

Answer Not Provided

Answer of exercise 1.1.6

Question:
Determine the language of the following NFA, M, over alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$.



a) Describe the language using set-builder description using an English description, e.g. $L(M) = \{w \mid w \text{ is a string that likes to eat monkeys.}\}$.

b) Describe the language using only explicit sets and the language operations (*, $\cup$, and $\circ$). (E.g. $L(M) = \{\mathtt{x}\}^* \circ (\{\mathtt{y}\} \cup \{\mathtt{z}\})$

Answer:

a) $L(M) = \{w \mid w \text{ contains only the character } \mathtt{c} \text{ and has an odd length}\}$

b) $L(M) = \{\mathtt{c}\}^+$

Answer of exercise 1.1.7

Answer Not Provided

Answer of exercise 1.1.8

Answer Not Provided

Answer of exercise 1.1.9

Answer Not Provided

Answer of exercise 1.1.10

Answer Not Provided

Answer of exercise 1.1.11

Answer Not Provided

Answer of exercise 1.1.12

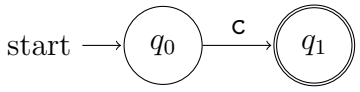
Question:

Use the NFA-building theorems from this section to build an NFA, M, so that $L(M) = \{\mathtt{c}\}^+$.

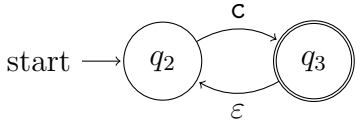
Answer:

$\{\mathtt{c}\}^+ = \{\mathtt{c}\} \circ \{\mathtt{c}\}^*$

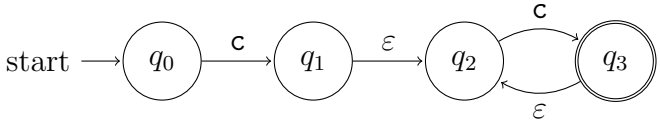
- $\{\mathtt{c}\}$:



- $\{\mathtt{c}\}^*$:



- $\{\mathtt{c}\} \circ \{\mathtt{c}\}^*$:



Answer of exercise 1.1.13

Answer Not Provided

Answer of exercise 1.1.14

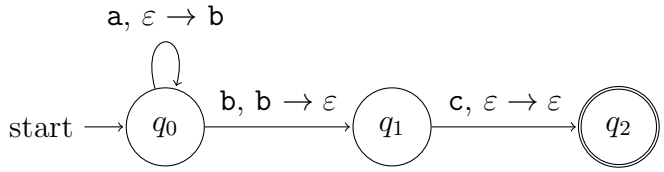
Answer Not Provided

Answer of exercise 1.1.15

Answer Not Provided

Answer of exercise 1.2.0

Question:
Give the formal (set notation) for the following PDA, P:



Answer:
$P = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where:

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \Gamma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$
- $$\delta = \{((q_0, \mathtt{a}, \varepsilon), (q_0, \mathtt{b})) \\ ((q_0, \mathtt{b}, \mathtt{b}), (q_1, \varepsilon)) \\ ((q_1, \mathtt{c}, \varepsilon), (q_2, \varepsilon))\}$$
- $F = \{q_2\}$

Answer of exercise 1.2.1

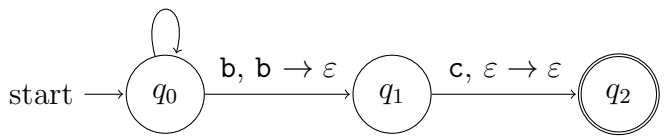
Answer Not Provided

Answer of exercise 1.2.2

Answer Not Provided

Answer of exercise 1.2.3

Question:
Describe the language, $L(P)$, for PDA P:



(Yes, this is the PDA from Exercise 1.2.0.)
Answer:
$L(P) = \{\mathtt{a}\}^+ \circ \mathtt{bc}$

Answer of exercise 1.2.4

Answer Not Provided

Answer of exercise 1.2.5

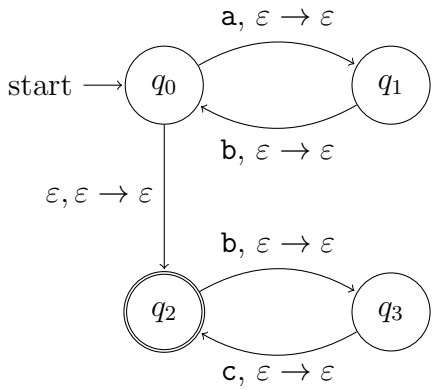
Answer Not Provided

Answer of exercise 1.2.6

Answer Not Provided

Answer of exercise 1.2.7

Question:
Describe PDA $M(A)$ where $A = \{\mathtt{ab}\}^* \circ \{\mathtt{bc}\}^*$. Give either a figure or the set notation.
Answer:
Figure version:



Set notation:
$M(A) = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where:

- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$
- $\Gamma = \emptyset$
- $$\begin{aligned}\delta = \{&((q_0, \mathtt{a}, \varepsilon), (q_1, \varepsilon))\\ &((q_0, \varepsilon, \varepsilon), (q_2, \varepsilon))\\ &((q_1, \mathtt{b}, \varepsilon), (q_0, \varepsilon))\\ &((q_2, \mathtt{b}, \varepsilon), (q_3, \varepsilon))\\ &((q_3, \mathtt{c}, \varepsilon), (q_2, \varepsilon))\}\end{aligned}$$

  , and
- $F = \{q_2\}$

Answer of exercise 1.2.8

Answer Not Provided

Answer of exercise 1.2.9

Answer Not Provided

Answer of exercise 1.2.10

Answer Not Provided

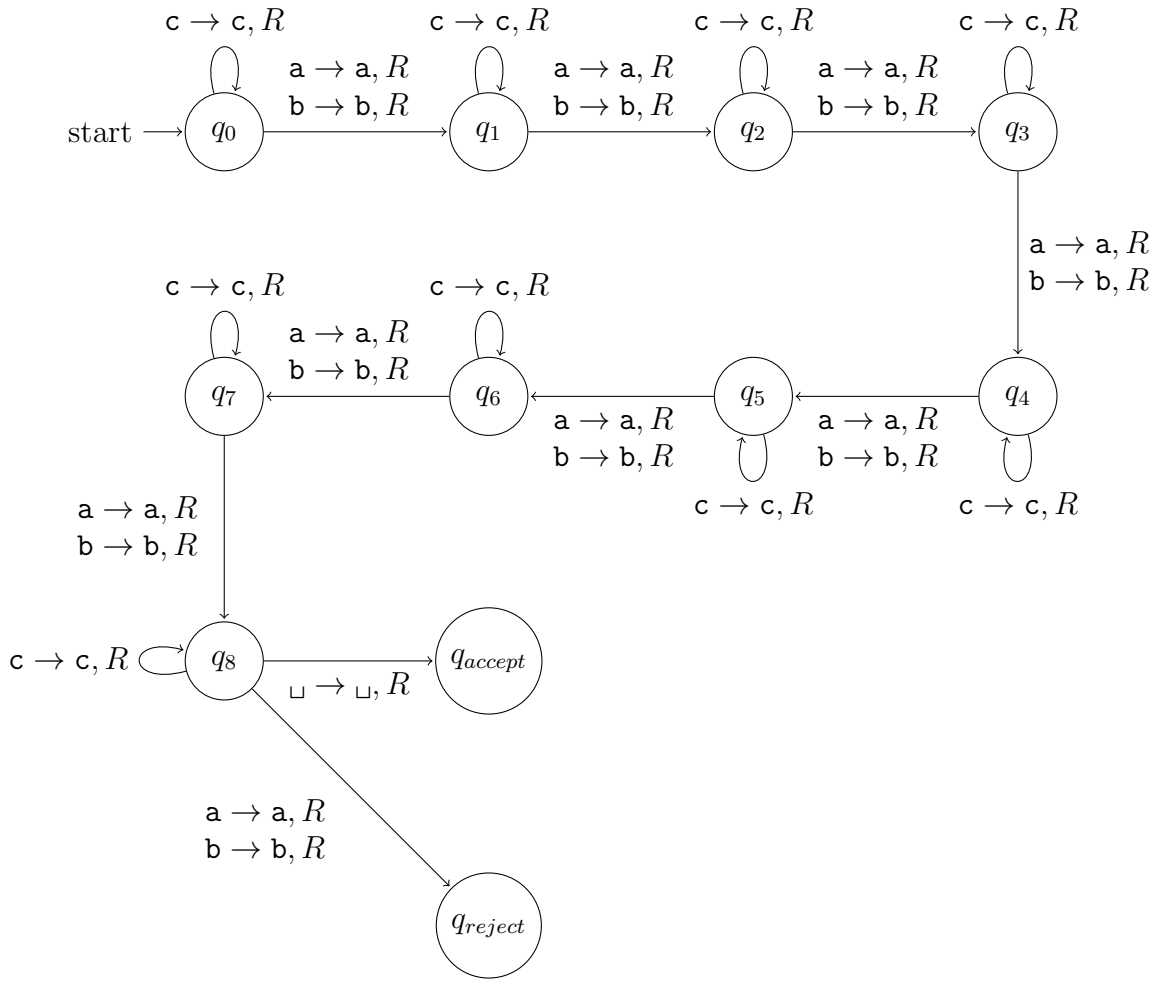
Answer of exercise 1.3.0

Question:

Draw the diagram for a Turing Machine that decides the following language over alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$:

$\{w \mid$ The sum of the number of $\mathtt{a}$'s and $\mathtt{b}$'s in w is 8$\}$

Answer:



Answer of exercise 1.3.1

Answer Not Provided

Answer of exercise 1.3.2

Answer Not Provided

Answer of exercise 1.3.3

Question:

No PDA exists to recognize

$$L_0 = \{w \mid w \text{ contains an equal number of } \texttt{a}\text{'s, } \texttt{b}\text{'s, and } \texttt{c}\text{'s}\}$$

. A Turing Machine does exist that decides L_0, but it is too complicated to include as a homework problem.

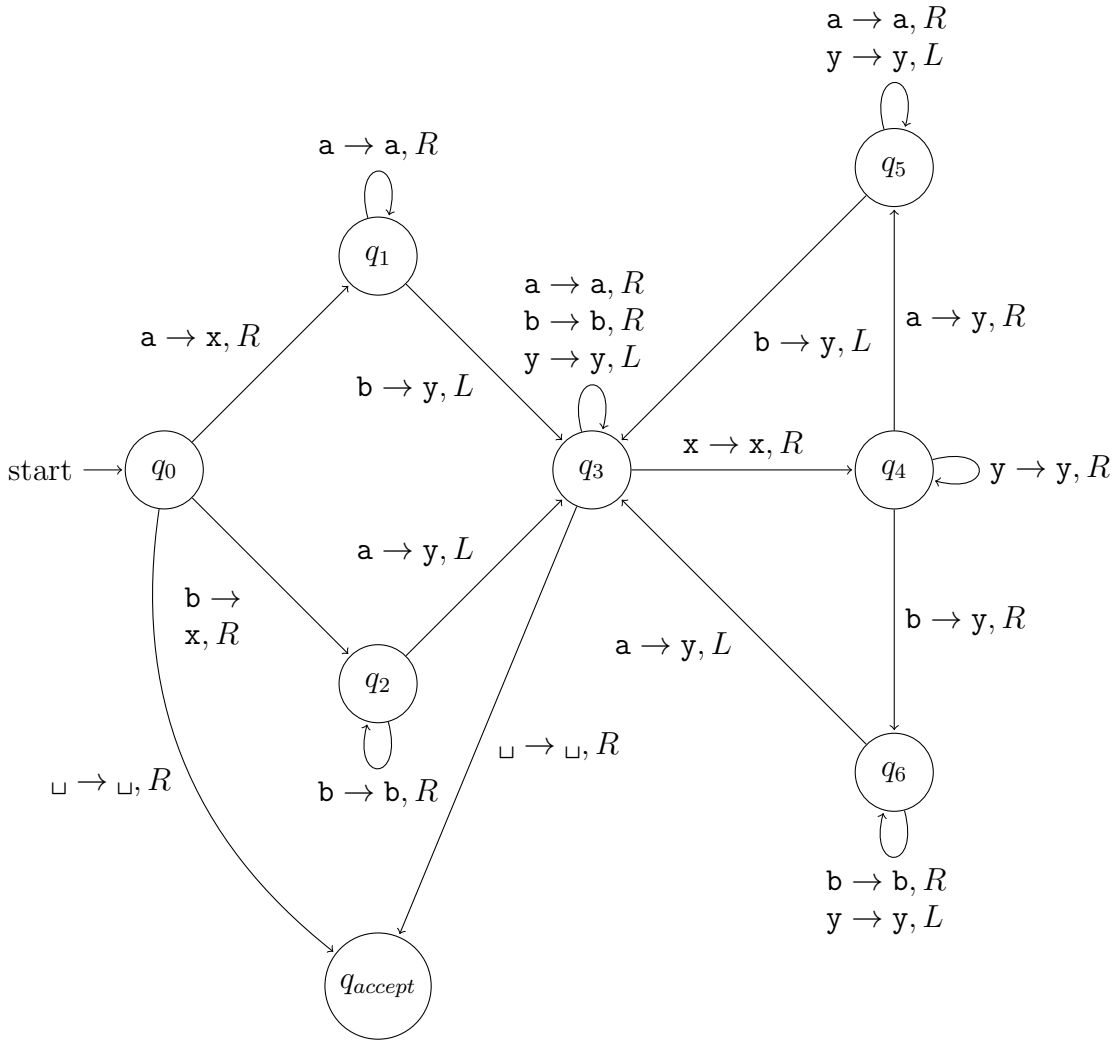
Instead, draw the diagram for a Turing Machine that decides

$$L_1 = \{w \mid w \text{ contains an equal number of } \texttt{a}\text{'s and } \texttt{b}\text{'s}\}$$

over the input alphabet $\{\texttt{a}, \texttt{b}\}$. There is a PDA that recognizes this, but it's still a worthwhile exercise.

Answer:

All missing edges lead to q_{reject}.



Answer of exercise 1.3.5

Answer Not Provided

Answer of exercise 1.3.5

Answer Not Provided

Answer of exercise 2.0.0

Question:
The following language is regular over the alphabet $\{\texttt{a}\}$:

$$L = \{w \mid \text{The number of } \texttt{a}\text{'s in } w \text{ is congruent to 2 mod 5.}\}$$

Prove that L is regular by finding a regular expression equivalent to L.
Answer:

$$\begin{aligned} L &= \{w \mid \text{The number of } \texttt{a}\text{'s in } w \text{ is congruent to 2 mod 5}\} \\ &= \{w \mid w \text{ is } \texttt{aa} \text{ followed by a multiple of } \texttt{aaaaa} \} \\ &= \{\texttt{aa}w \mid \text{The number of } \texttt{a}\text{'s in } w \text{ is a multiple of 5.}\} \\ &= \{\texttt{aa}\} \circ \{w \mid \text{The number of } \texttt{a}\text{'s in } w \text{ is a multiple of 5.}\} \\ &= \{\texttt{aa}\} \circ \{\texttt{aaaaa}\}^* \end{aligned}$$

Thus, $L = \{\texttt{aa}\} \circ \{\texttt{aaaaa}\}^*$

Answer of exercise 2.0.1

Question:
The following language is regular over the alphabet $\{\texttt{a}, \texttt{b}\}$:

$$L = \{w \mid w \text{ does not contain both } \texttt{a}\text{'s and } \texttt{b}\text{'s.}\}$$

Prove that L is regular by finding a regular expression equivalent to L.
Answer:

$$\begin{aligned} L &= \{w \mid w \text{ does not contain both } \texttt{a}\text{'s and } \texttt{b}\text{'s.}\} \\ &= \{w \mid w \text{ contains only } \texttt{a}\text{'s or only } \texttt{b}\text{'s}\} \\ &= \{w \mid w \text{ contains only } \texttt{a}\text{'s } \} \\ &\quad \cup \{w \mid w \text{ contains only } \texttt{b}\text{'s}\} \\ &= \{\texttt{a}\}^* \cup \{\texttt{b}\}^* \end{aligned}$$

Thus, $L = \{\texttt{a}\}^* \cup \{\texttt{b}\}^*$.

Answer of exercise 2.0.2

Answer Not Provided

Answer of exercise 2.0.3

Answer Not Provided

Answer of exercise 2.1.0

Question:
Consider the following CFG G:

$$A \to \mathtt{z}A\mathtt{z} \mid \mathtt{y}B\mathtt{y}$$
$$B \to \mathtt{aaa} \mid \mathtt{z}C\mathtt{b}$$
$$C \to \mathtt{a} \mid A \mid \mathtt{bz}$$

(a) What are the variables of G?
(b) What is the start variable of G?
(c) What are the terminals of G?
(d) Give the derivation that shows $\mathtt{zyaaayz} \in L(G)$.
(e) Give the derivation that shows $\mathtt{zyaaayz} \in L(G)$.

Answer:

(a) A, B, and C are the variables.
(b) A is the start variable.
(c) $\mathtt{a}$, $\mathtt{b}$, $\mathtt{y}$, and $\mathtt{z}$ are the terminals.
(d) $A \Rightarrow \mathtt{z}A\mathtt{z} \Rightarrow \mathtt{zy}B\mathtt{yz} \Rightarrow \mathtt{zyaaayz}$
(e) $A \Rightarrow \mathtt{z}A\mathtt{z} \Rightarrow \mathtt{zy}B\mathtt{yz} \Rightarrow \mathtt{zyz}C\mathtt{byz} \Rightarrow \mathtt{zyzy}B\mathtt{ybyz} \Rightarrow \mathtt{zyzyz}C\mathtt{bybyz} \Rightarrow \mathtt{zyzyzabybyz}$

Answer of exercise 2.1.1

Answer Not Provided

Answer of exercise 2.1.2

Answer Not Provided

Answer of exercise 2.1.3

Question:
Let G be the following context-free grammar:

$$C \to \mathtt{a}C \mid \mathtt{a}D\mathtt{a} \mid \mathtt{b}$$
$$D \to \mathtt{b} \mid \varepsilon \mid C\mathtt{aaa}$$

1. Show that $\mathtt{aaaba} \in L(G)$.
2. Show that $\mathtt{abba} \notin L(G)$.

Answer:

1. $\mathtt{aaaba} \in L(G)$. To prove this, we show a derivation $C \Rightarrow^* \mathtt{aaaba}$.

$$\begin{aligned} C &\Rightarrow \mathtt{a}C \\ &\Rightarrow \mathtt{aa}C \\ &\Rightarrow \mathtt{aaa}D\mathtt{a} \\ &\Rightarrow \mathtt{aaaba} \end{aligned}$$

2. $\texttt{abba} \notin L(G)$. To prove this, we show that no possible derivations work. When there's no chance for a path to match `abba` any longer, we indicate this with the $\oslash$ symbol.

   Start with C:

   - $C \Rightarrow \texttt{a}C \Rightarrow \texttt{aa}C \not\Rightarrow \texttt{abba}\ \oslash$
   - $\Rightarrow \texttt{aa}D\texttt{a} \not\Rightarrow \texttt{abba}\ \oslash$
   - $\Rightarrow \texttt{ab} \neq \texttt{abba}\ \oslash$
   - $\Rightarrow \texttt{a}C \not\Rightarrow \texttt{abba}\ \oslash$
   - $\Rightarrow \texttt{a}D\texttt{a} \Rightarrow \texttt{aba} \neq \texttt{abba}\ \oslash$
   - $\Rightarrow \texttt{aa} \neq \texttt{abba}\ \oslash$
   - $\Rightarrow \texttt{a}C\texttt{aaaa} \not\Rightarrow \texttt{abba}\ \oslash$
   - $\Rightarrow \texttt{a}D\texttt{a} \not\Rightarrow \texttt{abba}\ \oslash$
   - $\Rightarrow \texttt{b} \neq \texttt{abba}\ \oslash$
   - $C \not\Rightarrow \texttt{abba}$

   All of the paths fail, so `abba` cannot be in $L(G)$.

Answer of exercise 2.1.4

Answer Not Provided

Answer of exercise 2.1.5

Answer Not Provided

Answer of exercise 2.1.6

Question:
Give a context-free grammar, G, that generates $L = \{w \mid w \text{ starts and ends with } \texttt{0}\}$ over alphabet $\Sigma = \{\texttt{0}, \texttt{1}\}$.
Answer:
$G =$

$$A \to \texttt{0} \mid \texttt{0}B\texttt{0}$$
$$B \to \texttt{0}B \mid \texttt{1}B \mid \varepsilon$$

G generates L because:

- B generates a binary string of any length, then
- A is either just `0` or B with zeroes stuck on the front and end.

Answer of exercise 2.1.7

Answer Not Provided

Answer of exercise 2.1.8

Answer Not Provided

Answer of exercise 2.1.9

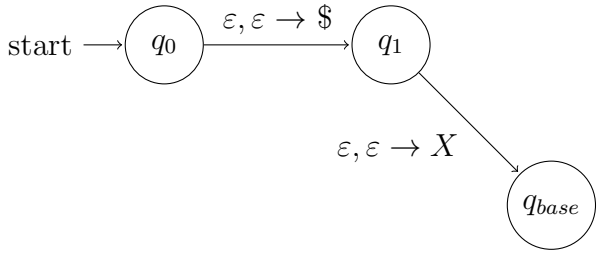
Question:

Create a PDA for the following grammar on $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. (Hint: Book Theorem 2.20 has the steps to follow on page 118.)

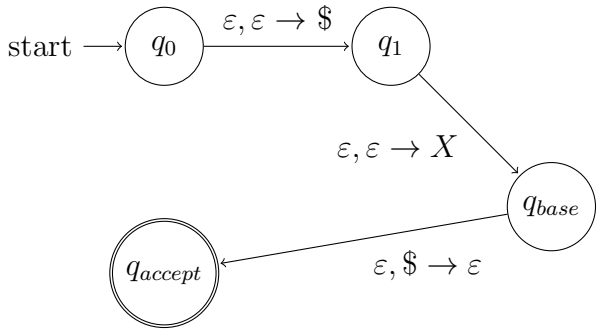
$$X \to X\mathtt{a} \mid Y\mathtt{a}$$
$$Y \to Y\mathtt{b} \mid Z$$
$$Z \to Z\mathtt{c} \mid \mathtt{c}$$

Answer:

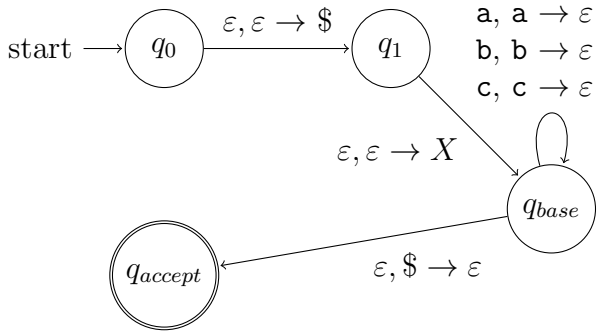
Begin with a PDA that just adds the \$ to the bottom of the stack, followed by the start variable from the grammar, then moves to the "base state" (this is part 1):



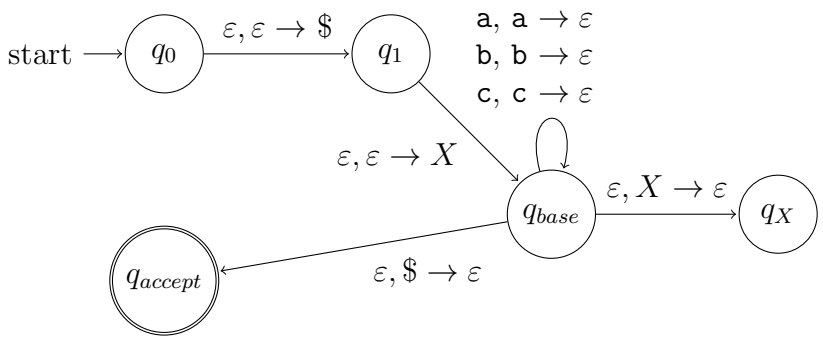
Now add the final accepting state, which will succeed once all the input has been read (this is part 2c):



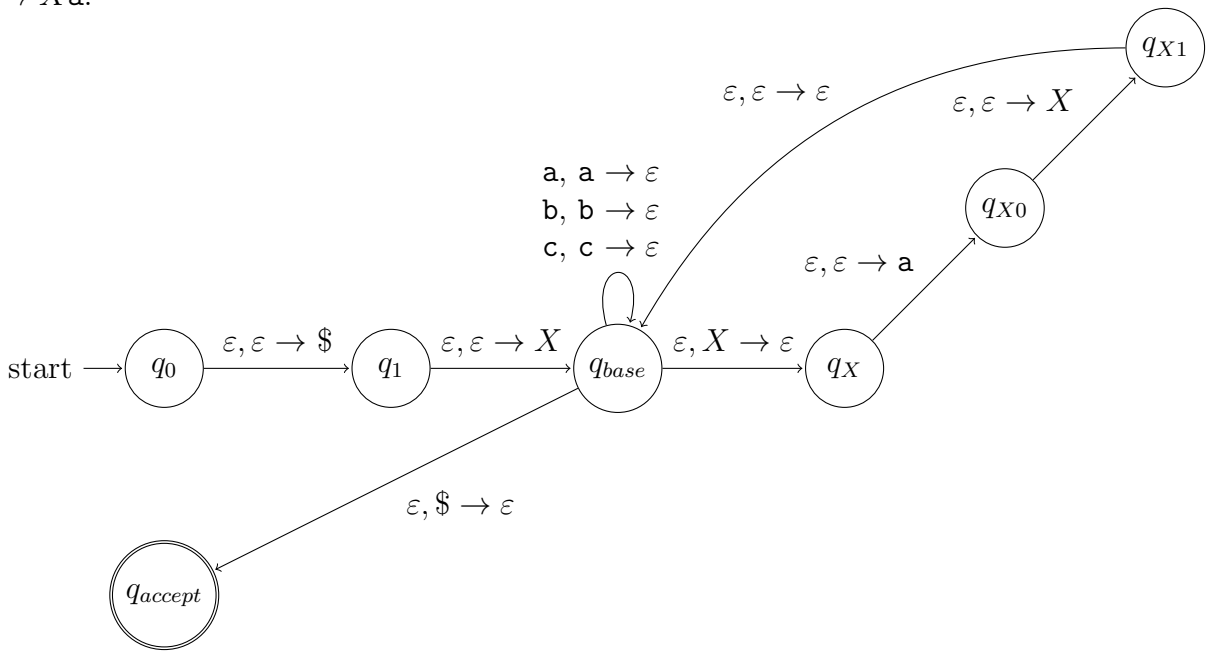
In this grammar, the following symbols are used for terminal characters: $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. Create a cycle to and from the base state that uses these (this is part 2b):



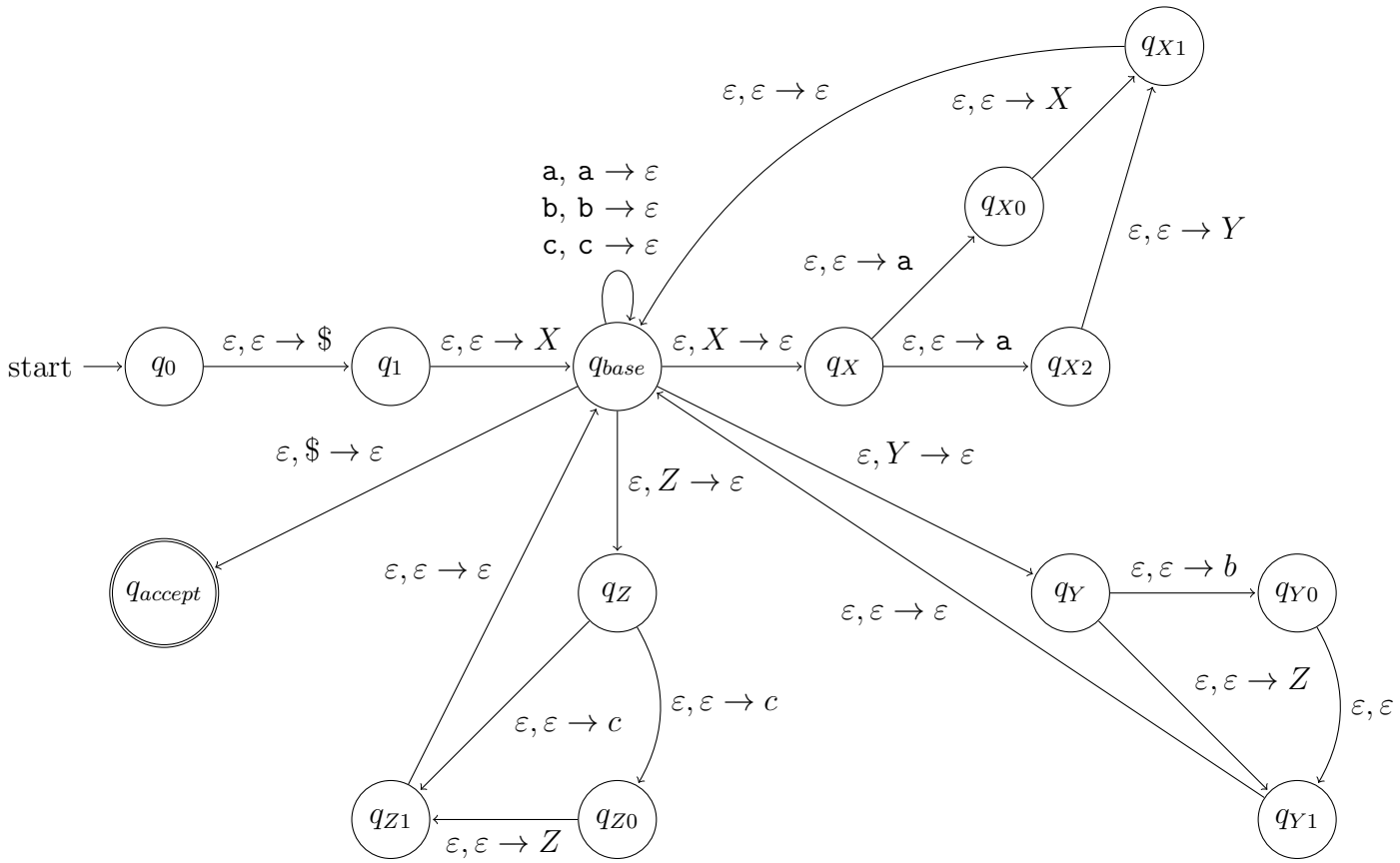
Now add a leaving path for the first grammar variable (this is part 2a):



Then include transitions for each option from that variable-rule, putting the symbols on the stack in reverse order, then returning to the base state. Here's the loop for the rule $X \to X\mathtt{a}$:



Add the remaining loops for X and all the loops for the other variables to get the finished automata:



Answer of exercise 2.1.10

Answer Not Provided

Answer of exercise 2.1.11

Answer Not Provided

Answer of exercise 2.2.0

Question:

Let $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. Let Σ-STAR be this language: $\{A \mid A \text{ is a DFA and } L(A) = \Sigma^*\}$. Show that Σ-STAR is decideable.

Answer:

We show that Σ-STAR is decideable by giving a high-level description of a Turing Machine to decide it. (This is similar to Theorem 4.4 from our book.)

On input $\langle A \rangle$, where A is a DFA:

1. Mark the start state of A.
2. Repeat until no new states get marked: Mark any state that has a transition to it from any already-marked state.
3. If any non-accepting state is marked, *reject*. Otherwise, *accept*.

This decides Σ-STAR because it should accept all inputs. All marked states are reachable by some input string, so A accepts all inputs exactly when all marked states are accepting states.

Answer of exercise 2.2.1

Question:

Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ and ONLYA $= \{A \mid A \text{ is a DFA and } L(A) = \{\mathtt{a}\}^*\}$. Show that ONLYA is decideable.

Answer:

We show that ONLYA is decideable by describing a program that decides ONLYA:

On input (A), where A is a DFA:

1. Mark the start state of A with one symbol (say, `x`).
2. Repeat until no new states get marked with `x`: Mark any state `x` that has a transition on `a` to it from any already-marked `x` state.
3. For each state in A marked with an `x`, follow the `b`-transition and mark the resulting state with symbol `y`.
4. Repeat until no new states get marked with `y`: Mark any state `y` that has a transition to it from any already-`y`-marked state.
5. Check that all `x`-marked states are accepting states, and all `y`-marked and non-marked states are not accepting states. If this is the case, *accept*, otherwise *reject*.

This decides ONLYA because all `a`-transitions lead back to accepting states, but all `b`-transitions should lead to non-accepting states. Any states marked `x` are reachable by only `a` strings, while `y`-marked states are reachable by any string that contains at least one `b`. (States marked with both should not are reachable by both, and such a machine will be rejected.)

Answer of exercise 2.2.2

Answer Not Provided

Answer of exercise 2.2.3

Answer Not Provided

Answer of exercise 2.2.4

Question:

Let NFA-ACCEPT $= \{(A, w) \mid A \text{ is an NFA and } w \in L(A)\}$. Show that NFA-ACCEPT is decideable. (This is totally an example in the text book. TODO: add the citation here.)

Answer:

Here is the decider for NFA-ACCEPT:

Step 0 : Verify that A is an NFA. If not, *reject*.

Step 1 : Verify that $w \in \Sigma(A)^*$. If not, *reject*.

Step 2 : Create DFA B from A using subset construction.

Step 3 : Simulate B on input w.

Step 4 : $\begin{cases} \textit{accept} & \text{, if B accepts w} \\ \textit{reject} & \text{, if B rejects w} \end{cases}$

Answer of exercise 3.0.0

Question:
What is the pumping length of $L = \{\mathtt{a}, \mathtt{aa}, \mathtt{ab}, \mathtt{bb}\}$?
Answer:
This is a finite set of strings, so the pumping length is one more than the longest string in L. $p = 3$.

Answer of exercise 3.0.1

Answer Not Provided

Answer of exercise 3.0.2

Question:
Consider the following regular language:

$$L = \{\mathtt{a}\}^* \circ \{\mathtt{b}\}^*$$

(a) Choose a working p to satisfy the pumping lemma.

(b) For any string, w, longer than your chosen p, show how to divide it into x, y, and z. (You might need multiple cases.)

(c) Show that the three conditions of the decomposition hold.

Answer:

(a) $p = 1$

(b) 2 cases: $\mathtt{a} \in w$ and $\mathtt{a} \notin w$

Case A: $\mathtt{a} \in w$, so $w = \mathtt{a}^{1+j}\mathtt{b}^k$ where $j, k \geq 0$

- $x = \varepsilon$
- $y = \mathtt{a}$
- $z =$ remainder of $w = \mathtt{a}^j\mathtt{b}^k$

Case B: $\mathtt{a} \notin w$. Since $|b| \geq 1$, $w = \mathtt{b}^{1+j}$, where $j \geq 1$

- $x = \varepsilon$
- $y = \mathtt{b}$
- $z =$ remainder of $w = \mathtt{b}^j$

(c) In Case A:

- $|y| = |\mathtt{a}| \geq 1$ ✓
- $|xy| = |\mathtt{a}| \leq 1 = p$ ✓
- $\forall i : xy^iz = \mathtt{a}^i\mathtt{a}^j\mathtt{b}^k = \mathtt{a}^{i+j}\mathtt{b}^k \in L$ ✓

In Case B:

- $|y| = |\mathtt{b}| \geq 1$ ✓
- $|xy| = |\mathtt{b}| \leq 1 = p$ ✓
- $\forall i : xy^iz = \mathtt{b}^i\mathtt{b} = \mathtt{b}^{i+j} \in L$ ✓

Answer of exercise 3.0.3

Answer Not Provided

Answer of exercise 3.0.4

Answer Not Provided

Answer of exercise 3.0.5

Question:
Use the pumping lemma to show that the following language, L, is not regular.

$$L = \{a^{2^k} \mid \forall k \in \mathbb{N}\}$$

Answer:
Proof-by-contradiction. Assume L is regular. Then the pumping lemma applies.
Let w be a string of 2^k a's, with a working decomposition $w = xyz$. Note that $|y| \geq 1$ (otherwise, the first decomposition condition is broken) so y includes at least one a.
Now, by the third condition, $xy^2z = xyyz$ is also in the language, and, since $|y|\, leq xyz$, $|xyyz| \leq 2\,|xyz|$.
However, $|xyyz| \geq 2\,|xyz|$, otherwise, $|xyyz|$ is not a power of 2.
Thus, $|xyyz| = 2\,|xyz|$, meaning $x = z = \varepsilon$. Then $xyz = y$. Also, by the third condition, $y^3 = yyy \in L$.
However, y and y^3 cannot both have lengths that are powers of 2. Thus, y and yyy cannot both be in L. $\rightarrow\leftarrow$
By contradiction, our assumpion that L is regular must be false.
Thus, L is not regular.

Answer of exercise 3.0.6

Answer Not Provided

Answer of exercise 3.0.7

Answer Not Provided

Answer of exercise 3.1.0

Question:
Consider the following context-free language:

$$L = \{\mathtt{a}^k\mathtt{bc}^k \mid k \in \mathbb{N}\}$$

(a) Choose a working p to satisfy the pumping lemma. (It does not need to be the shortest.)

(b) For any string, w, longer than your chosen p, show how to divide it into u, v, x, y, and z. (You might need multiple cases.)

(c) Show that the three conditions of the pumping lemma decomposition hold for your divisions.

Answer:

(a) $p = 3$

(b) Let $w = \mathtt{a}^k\mathtt{bc}^k$. Then $k \geq 1$ in order for $|w| \geq p$. We can divide in the following way:

- $u = \mathtt{a}^{k-1}$
- $v = \mathtt{a}$
- $x = \mathtt{b}$
- $y = \mathtt{c}$
- $z = \mathtt{c}^{k-1}$

(c) For our division, check that we satisfy the three properties:

- $|vxy| = |\mathtt{abc}| = 3 \leq p$ ✓
- $|vy| = |\mathtt{ac}| = 2 \geq 1$ ✓
-

$$\begin{aligned} \forall i : \mathtt{uv}^i\mathtt{xy}^i\mathtt{z} &= \mathtt{a}^{k-1}\mathtt{a}^i\mathtt{bc}^i\mathtt{c}^{k-1} \\ &= \mathtt{a}^{k+i-1}\mathtt{bc}^{k+i-1} \\ &\in L \checkmark \end{aligned}$$

Answer of exercise 3.1.1

Answer Not Provided

Answer of exercise 3.1.2

Answer Not Provided

Answer of exercise 3.1.3

Question:

(Should I even include this one? I do it in class. I guess I'm keeping this right now because the answer in the back is more detailed than in the notes.) Use the Pumping Lemma for Context-Free Languages to show that this language is not context free:

$$L = \{\mathtt{a}^{2^n} \mid n \in \mathbb{N}\}$$

Answer:

Proof-by-contradiction: Assume L is context-free.

Then the pumping lemma for context-free languages must hold. Thus, there exists a p such that for any string, $w \in L$, if $|w| \geq p$, there must be a working decomposition $w = uvxyz$ such that:

- $|vxy| \leq p$
- $|vy| \geq 1$
- $\forall i : uv^ixy^iz \in L$

Consider the first power of 2 strictly greater than p, $2^k > p$. $\mathtt{a}^{2^k} \in L$. Let $uvxyz$ be the decomposition for $w = \mathtt{a}^{2^k}$.

Since $|vy| \geq 1, \mathtt{a} \in vy$. Thus, $|uvvxyyz| > 2^k$

Also, since $|vxy| \leq p$:

$$\begin{aligned} |uvvxyyz| &= |uvxyz| + |vy| \\ &\leq |uvxyz| + |vxy| \\ &= 2^k + |vxy| \\ &\leq 2^k + p \\ &< 2^k + 2^k \\ &= 2^{k+1} \end{aligned}$$

Thus, $2^k < |uvvxyyz| < 2^{k+1}$, so $uv^2xy^2z \notin L$. Thus L cannot be context-free! $\rightarrow\leftarrow$

By contradiction, the assumption must be false. Thus, L must not be context-free.

Answer of exercise 3.1.4

Question:

Use the Pumping Lemma for Context-Free Languages to show that L is not context free, where

$$L = \{\mathtt{a}^n\mathtt{b}^m\mathtt{c}^n\mathtt{d}^m |\ n, m \in \mathbb{N}\}$$

Answer:

Proof-by-contradiction. Assume that L is context free. Then there must be a p that satisfies the pumping lemma (fCFL). Thus, for any string $w \in L$ with at least p characters, there's a decomposition that satisfies the three conditions.

Consider $w = \mathtt{a}^{p+1}\mathtt{b}^{p+1}\mathtt{c}^{p+1}\mathtt{d}^{p+1}$. Let's find a working decomposition to satisfy the pumping lemma (for Context-Free Languages).

In order for the third condition to be met ($\forall i : uv^ixy^iz \in L$), v cannot consist of a mix of different characters and the same is true for y. Since the number of a's and c's has to be the same (and the same is true for b's and d's) there are only two possible cases. Either $v = \mathtt{a}^k$ and $y = \mathtt{c}^k$ or $v = \mathtt{b}^k$ and $y = \mathtt{d}^k$.

Case A: $v = \mathtt{a}^k$ and $y = \mathtt{c}^k$ where $k \leq p + 1$

Then x contains all of the b's, so $|vxy| \geq p + 1 > p$, breaking that condition.

Case B: $v = \mathtt{b}^k$ and $y = \mathtt{d}^k$ where $k \leq p + 1$

Then x contains all of the c's, so $|vxy| \geq p + 1 > p$, also breaking that condition.

Neither of the cases satisfy the pumping lemma! $\rightarrow\leftarrow$ Thus, L must not be context free.

Answer of exercise 3.1.5

Answer Not Provided

Answer of exercise 3.1.6

Answer Not Provided

Answer of exercise 3.2.0

Question:

Find a reduction, f from A to B, where:

$$A = \{\mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N}\}$$

, and

$$B = \{\mathtt{a}^n\mathtt{b}^m\mathtt{c}^m\mathtt{d}^n \mid n, m \in \mathbb{N}\}$$

Assume that both languages are over the same alphabet: $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$.

Answer:

f needs to be a function from Σ^* to itself, so: $f : \Sigma^* \to \Sigma^*$.

Let $w = x_0x_1x_2\cdots x_k$ where each x_i is a single character of w. Let g be this function on a single character:

$$g(x) = \begin{cases} \mathtt{a} & , x = \mathtt{a} \\ \mathtt{d} & , x = \mathtt{b} \\ \mathtt{c} & , x \in \{\mathtt{c}, \mathtt{d}\} \end{cases}$$

Now, let's define f:

$$f(w) = f(x_0x_1x_2\cdots x_k) = \begin{cases} w & , |w| = k+1 \text{ is odd} \\ g(x_0)g(x_1)g(x_2)\cdots g(x_k) & , \text{ otherwise} \end{cases}$$

Let's check that f is a reduction:

- f is a function from Σ^* to Σ^*. ✓
- f is computable. ✓
- We still have to show that $f(w) \in B \Leftrightarrow w \in A$. Two cases: $w \in A \Rightarrow f(w) \in B$ and $f(w) \in B \Rightarrow w \in A$. [12]

  Case A: $w \in A$

  Then $\exists k$ such that $w = \mathtt{a}^k\mathtt{b}^k$. Then,

$$\begin{aligned} f(w) &= f(\mathtt{a}^k\mathtt{b}^k) \\ &= g(\mathtt{a})^k g(\mathtt{b})^k \\ &= \mathtt{a}^k\mathtt{d}^k \\ &\in B \checkmark \end{aligned}$$

  Case B: $f(w) \in B$

  $f(w)$ can't contain any $\mathtt{b}$'s (since $\mathtt{b}$ is not in the range of g) so $f(w) = \mathtt{a}^k\mathtt{d}^k$ for some k. Since the only way to get those characters with g is from $\mathtt{a}$'s and $\mathtt{b}$'s, respectively, w must equal $\mathtt{a}^k\mathtt{b}^k$. ✓

So, f is a reduction from A to B.

Answer of exercise 3.2.1

Answer Not Provided

Answer of exercise 3.2.2

Answer Not Provided

[12] Alternatively, for the second case, we could just show $w \notin A \Rightarrow f(w) \notin B$. I chose to stick with the direct route because there were lots of little cases I didn't want to list, but it can certainly be done!

Answer of exercise 3.2.3

Question:

Consider the following language:

YesBeluga = $\{M \mid M$ is a Java program that prints the string `Beluga` to the console$\}$

Show that YesBeluga is undecideable.

Answer:

Reduce from Halting Problem. So we'll transform any Halting-problem input into an equivalent YesBeluga-program in the following way:

Let X be the program input for the halting problem. We then transform X into $Y = T(X)$.

On input string w, $Y(w)$:

1. Store the original print stream with:

```
PrintStream originalSysOut = System.out;
```

2. Redirect standard out to an anonymous class that will do nothing:

```
System.setOut(new PrintStream() {
    public void print(String s) {
        //do nothing
    }
});
```

3. Run program $X(w)$.
4. Restore the old `System.out`:

```
System.setOut(originalSysOut);
```

5. Now print the beluga:

```
System.out.println("Beluga");
```

Proof-by-contradiction:

Assume that program R solves YesBeluga. Thus, $R(X)$ *accepts* exactly if X is a program that prints the string `Beluga`.

We now create a program that solves the Halting Problem:

$M(X, w)$:

1. Create program $Y = T(X)$.
2. Run R on (Y, w)
3. If R accepts, *accept*. Otherwise, *reject*.

Now $M(X, w)$ accepts exactly when $X(w)$ would halt, thus M solves the halting problem! It can't exist. Thus, YesBeluga is undecideable.

Answer of exercise 3.2.4

Answer Not Provided

Answer of exercise 3.2.5

Question:
Show that the following language, VISITSSTATE, is undecideable.

$$\text{VISITSSTATE} = \{(M, q, w) \mid \text{TM } M \text{ run on input } w \text{ visits state } q\}$$

Answer:
Reduce from Halting Problem.

Let (M, w) be the input for the halting problem. We will build a new machine M', choose a state q' such that $(M', q', w) \in$ VISITSSTATE exactly when $(M, w) \in$ HALTING.

Let $M = (Q, \Gamma, \Sigma, \delta, q_0)$. We need to talk specifically about the transitions into q_{accept} and q_{reject}, so we specify those parts of δ:

- $\delta_{to-acc} = \{((q_i, s_i), (q_j, s_j, d)) \in \delta \mid q_j = q_{accept}\}$
- $\delta_{to-rej} = \{((q_i, s_i), (q_j, s_j, d)) \mid q_j = q_{reject}\}$

We create $M' = (Q', \Gamma', \Sigma', \delta', q_0')$ from M this way:

- $Q' = Q \cup \{q_{old-acc}, q_{old-rej}\}$
- $\Gamma' = \Gamma$
- $\Sigma' = \Sigma$
- $\delta' = \delta \setminus (\delta_{to-acc} \cup \delta_{to-rej}) \cup \delta_{to-old-a} \cup \delta_{to-old-r} \cup \delta_{to-new-acc}$

Where we define these new δ-sets as:

- $\delta_{to-old-a} = \{((q_i, s_i), (q_{old-acc}, s_j, d)) \mid ((q_i, s_i), (q_{accept}, s_j, d)) \in \delta_{to-acc}\}$
- $\delta_{to-old-r} = \{((q_i, s_i), (q_{old-rej}, s_j, d)) \mid ((q_i, s_i), (q_{reject}, s_j, d)) \in \delta_{to-rej}\}$
- $\delta_{to-new-acc} = \{(q_i, s_i), (q_{accept}, \sqcup, R)) \mid \forall q_i \in \{q_{old-acc}, q_{old-rej}\}, s_i \in \Gamma\}$

Now the input for VISITSSTATE is: (M', q_{accept}, w).

Claim: $(M', q_{accept}, w) \in$ VISITSSTATE exactly when $(M, w) \in$ HALTING.

Proof of claim:

M' is exactly the same as M, except that we moved the accept and reject states (q_{accept} and q_{reject}) and replaced them with non-halting states ($q_{old-acc}$ and $q_{old-rej}$, respectively). Then we added transitions between these two states and q_{accept} so that the machine will always advance there.

Now, if $M(w)$ halts, then $M'(w)$ will reach either $q_{old-acc}$ or $q_{old-rej}$ and then move on to q_{accept}, which is the vertex we chose as our second input for VISITSSTATE. Thus, whenever $M(w)$ halts, $M'(w)$ will visit q_{accept}. ✓

We still have to show that if $M(w)$ doesn't halt, $M'(w)$ will not reach q_{accept}. Note that only way to reach q_{accept} in M' is by going through $q_{old-acc}$ or $q_{old-rej}$. Those two states are only reached on $M'(w)$ when $M(w)$ halts by reaching q_{accept} or q_{reject}. Thus, if $M(w)$ doesn't halt, $M'(w)$ cannot reach q_{accept}. ✓This completes the proof of the claim.

The reduction is complete, thus VISITSSTATE is undecideable.

Answer of exercise 3.2.6

Answer Not Provided

Answer of exercise 3.2.7

Answer Not Provided

Answer of exercise 3.2.8

Answer Not Provided

B Bibliography

References

[1] Michael Sipser. *Introduction to the theory of computation.* PWS Publishing Company, 1997.

