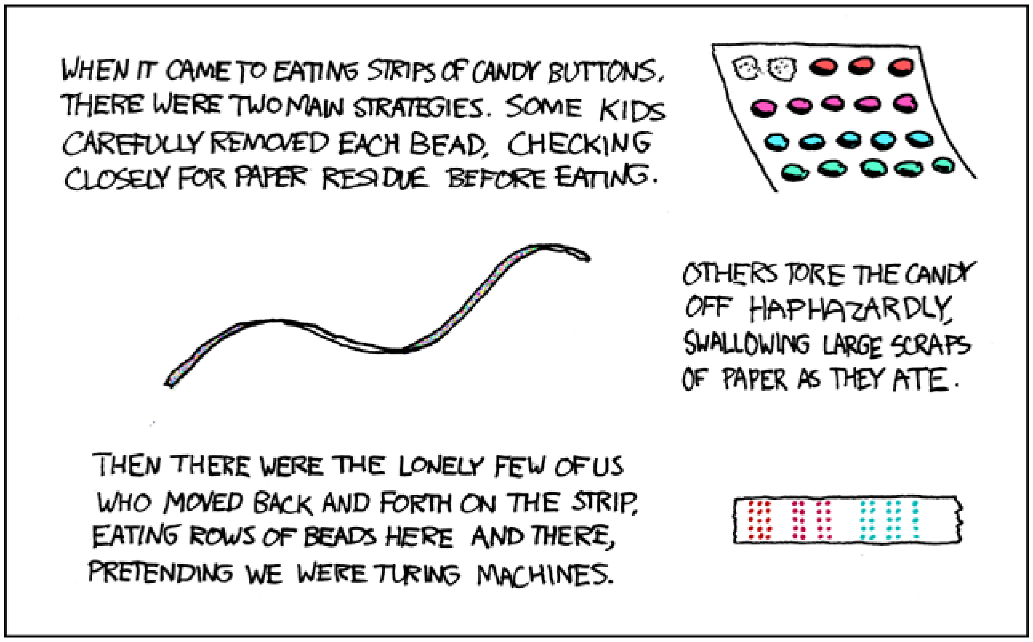
CS52 - Assignment 10

Due Wednesday 5/4 at 7:00pm


WHEN IT CAME TO EATING STRIPS OF CANDY BUTTONS, THERE WERE TWO MAIN STRATEGIES. SOME KIDS CAREFULLY REMOVED EACH BEAD, CHECKING CLOSELY FOR PAPER RESIDUE BEFORE EATING.
OTHERS TORE THE CANDY OFF HAPHAZARDLY, SWALLOWING LARGE SCRAPS OF PAPER AS THEY ATE.
THEN THERE WERE THE LONELY FEW OF US WHO MOVED BACK AND FORTH ON THE STRIP, EATING ROWS OF BEADS HERE AND THERE, PRETENDING WE WERE TURING MACHINES.

`https://xkcd.com/205/`

Important Notice Assignments 9 and 10 are due at the same time. This is to give you maximum flexibility in scheduling during the busy end of semester. It would be a serious mistake to attempt to begin either assignment in the few hours before it is due. Please plan ahead!

For this assignment you will be playing with the JFLAP automata tool to design DFAs, NFAs and turing machines. Each problem, except for number 5, requires you to construct a finite state automaton or a Turing machine. When you start JFLAP, select “Finite State Automaton” or “Turing Machine,” as appropriate. Construct your machine and use FJLAP to test it before saving it. Then save your JFLAP machine in a file named

    assign10-⟨one-digit-problem-number⟩.jff

For example, the file for Problem 3 would be named `assign10-3.jff`. Problem 5 is an exception; it asks for a text file named `assign10-5.txt`.

Obtaining JFLAP

We will use a Java-based program, called JFLAP, that we've been playing with in class. It was written by Professor Susan Rodger of Duke University.

- If you're working, use execute the following command in terminal:

  ```
  /common/cs/cs052/bin/jflap
  ```

- If you're working on your own computer, you can obtain it at:

  `http://www.jflap.org/`

  Click on the tab to the left "Get JFLAP", fill out the form and then download version 7.0 (NOT the beta version, 8.0). The "Thin" version should work, but if it doesn't, try downloading the full version. Once you download the .jar file you *should* only need to double-click on it for it to run.

  Even if you don't plan on doing the assignment early, make sure to download JFLAP ASAP and make sure that it works on your computer.s

If you need help using JFLAP, select the "Help" menu from the program or you can view a tutorial online at `http://www.jflap.org/`.

Finite Automota

1. [**2 poitns**] Abba (would not be accepted)

   Use JFLAP to construct a DFA that accepts all non-empty strings over the alphabet $\{a, b\}$ in which every other letter in the string is an a, starting with the first letter. The empty string should not be accepted, but ab and aaa should be.

2. [**2 points**] aaaaaah

   Use JFLAP to construct a six-state DFA which accepts a string over the one-letter alphabet $\{a\}$ if it has a number of a's that is a multiple of 2 or a multiple of 3 (or both).

3. [**2 points**] 7 is the magic number

   Use JFLAP to construct a DFA which accepts a string of 0's and 1's if, when interpreted as a binary number, is a multiple of 7. The binary representation is read from left to right; that is, the *most* significant bit comes first.

   Advice: This is not trivial; think about it before you start to build the DFA.

   Hint: The key is to think about this as a *math* problem and *not* a pattern matching problem. In assignment 8, problem 1, when you were processing bytes from left to right you were able to calculate a partial value for the byte so far. Think of trying to do something similar, since we'll be reading the bits off the string from left to right, one at a time.

4. [**3 points**] Abba (would be accepted)

Consider strings constructed over the alphabet $\{a, b\}$. The string *aabbbab* has two occurrences of the substring *ab* and one occurrence of the substring *ba*. The string *aba* has one occurrence of *ab* and one of *ba*; it does not matter that they share a character. Use JFLAP to construct a DFA that accepts strings for which the number of times *ab* appears is equal to the number of times that *ba* appears.

5. [**3 points**] If you squint, maybe you can distinguish them

   Two strings s and t are *distinguishable over a language* L if there is a third string u such that exactly one of su and tu are in L.

   (a) Find six strings which are pairwise distinguishable over the language of Problem 2 and show that they are in fact distinguishable. With six strings, there are fifteen pairs to check.

   (b) Any DFA that accepts the language (from Problem 2) must have at least six states. Justify this statement.

   For this problem, submit a separate file (plain text, not a Word document or a PDF file) named `assign10-5.txt`.

6. [**2 points**] aaaaah

   Use JFLAP to create a *nondeterministic* finite automaton that has eight states and accepts the strings over a one-letter alphabet $\{a\}$ whose length is a multiple of 2 or a multiple of 5 (or both). Notice that the technique you used in Problem 2 leads to a DFA with ten states.

Turing Machines

For the last two problems you will be creating Turing machines. Keep your Turing machines as simple as possible. Your Turing machines should have exactly one accepting state and one rejecting state, even though JFLAP does not enforce that requirement. In fact, JFLAP does not have explicit rejecting states; you can create a rejecting state by making a non-accepting state that has no transitions out of it.

7. [**3 points**] There are 11 kinds of people in the world, those that understand unary and those that don't

   Binary numbers consists of 0's and 1's. Unary numbers consist only of 1's. For example, we represent 1 as 1 in unary, 2 as 11, 3 as 111, etc. Write a Turing machine over the alphabet $\{1, -\}$ that accepts all unary number subtraction problems where the result is positive, i.e greater than 0. For example, "111-11" would be accepted while "11-111" would be rejected. Also, any string without a minus sign, or one with more than one minus sign, will be rejected.

8. [**3 points**] (Almost there!)

   Construct a turing machine over the input alphabet of parentheses, i.e. `)` and `(`, that recognizes the language of all strings of properly balanced parentheses (equal number of open and close parentheses *and* the open parentheses must come before the corresponding close parentheses). For example

```
()
(())
(()((()())))
```

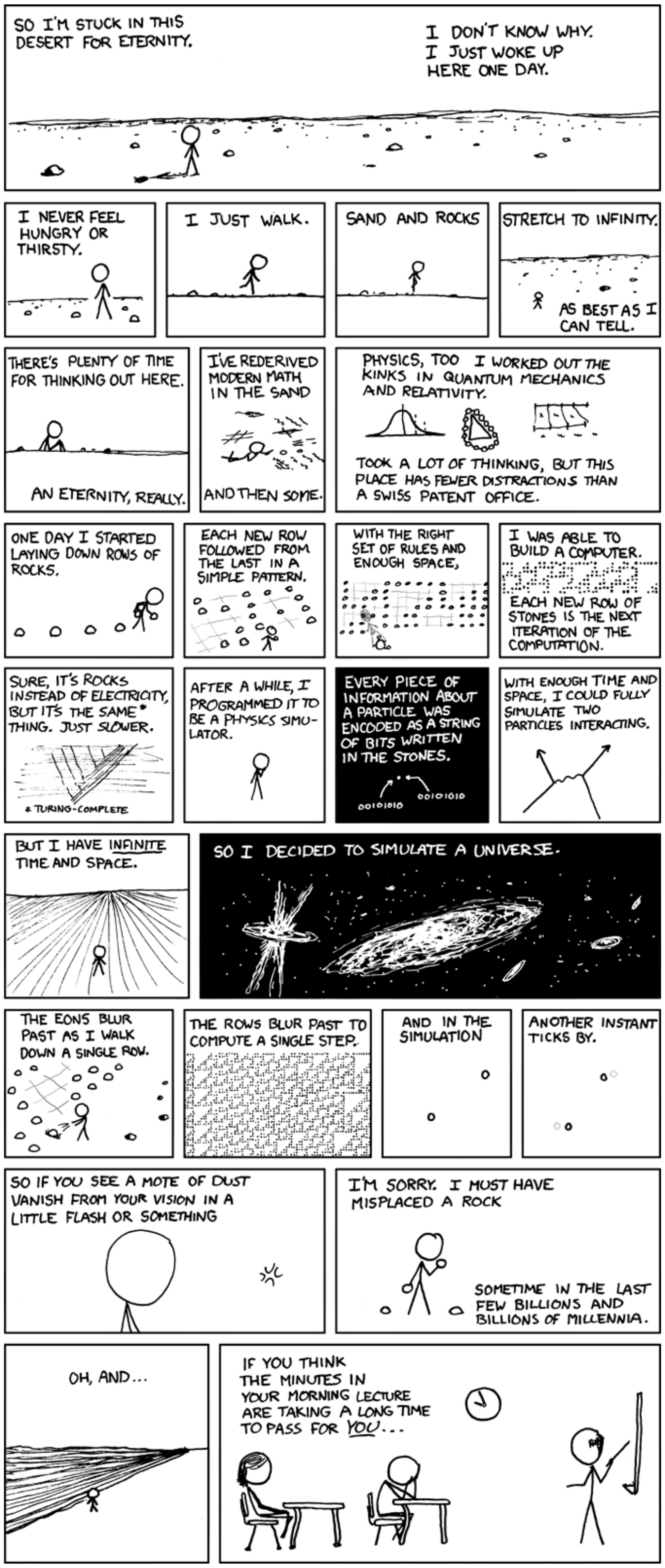
would all be members of the language, while

```
)(
(()
())(
```

would NOT be members of the language.

When you're done

You should have a separate file for each problem. Submit *each of these files separately* using the normal submission mechanism.


SO I'M STUCK IN THIS DESERT FOR ETERNITY.
I DON'T KNOW WHY. I JUST WOKE UP HERE ONE DAY.
I NEVER FEEL HUNGRY OR THIRSTY.
I JUST WALK.
SAND AND ROCKS
STRETCH TO INFINITY.
AS BEST AS I CAN TELL.
THERE'S PLENTY OF TIME FOR THINKING OUT HERE.
AN ETERNITY, REALLY.
I'VE REDERIVED MODERN MATH IN THE SAND
AND THEN SOME.
PHYSICS, TOO. I WORKED OUT THE KINKS IN QUANTUM MECHANICS AND RELATIVITY.
TOOK A LOT OF THINKING, BUT THIS PLACE HAS FEWER DISTRACTIONS THAN A SWISS PATENT OFFICE.
ONE DAY I STARTED LAYING DOWN ROWS OF ROCKS.
EACH NEW ROW FOLLOWED FROM THE LAST IN A SIMPLE PATTERN.
WITH THE RIGHT SET OF RULES AND ENOUGH SPACE,
I WAS ABLE TO BUILD A COMPUTER.
EACH NEW ROW OF STONES IS THE NEXT ITERATION OF THE COMPUTATION.
SURE, IT'S ROCKS INSTEAD OF ELECTRICITY, BUT IT'S THE SAME* THING. JUST SLOWER.
* TURING-COMPLETE
AFTER A WHILE, I PROGRAMMED IT TO BE A PHYSICS SIMULATOR.
EVERY PIECE OF INFORMATION ABOUT A PARTICLE WAS ENCODED AS A STRING OF BITS WRITTEN IN THE STONES.
00101010
00101010
WITH ENOUGH TIME AND SPACE, I COULD FULLY SIMULATE TWO PARTICLES INTERACTING.
BUT I HAVE INFINITE TIME AND SPACE.
SO I DECIDED TO SIMULATE A UNIVERSE.
THE EONS BLUR PAST AS I WALK DOWN A SINGLE ROW.
THE ROWS BLUR PAST TO COMPUTE A SINGLE STEP.
AND IN THE SIMULATION
ANOTHER INSTANT TICKS BY.
SO IF YOU SEE A MOTE OF DUST VANISH FROM YOUR VISION IN A LITTLE FLASH OR SOMETHING
I'M SORRY. I MUST HAVE MISPLACED A ROCK
SOMETIME IN THE LAST FEW BILLIONS AND BILLIONS OF MILLENNIA.
OH, AND...
IF YOU THINK THE MINUTES IN YOUR MORNING LECTURE ARE TAKING A LONG TIME TO PASS FOR YOU...

https://xkcd.com/505/