UNCA CSCI 431
In Class Final Exam Fall 2019
10 December 2019 – 11:30 am to 2:00 pm

You may use your notes, printouts, scratch paper, and your textbook. You may not use any calculators, electronic devices, or help from any other source or person.

Anyone needing a break during the exam must leave their exam with the instructor.

This exam must be turned in before 2:00 PM.

Name:________________________________

There are four equally weighted problems.

There is also a written take-home part to this exam.

Problem 1:

Draw a DFA for the regular expression.

- `uv + (x + yz)*`

You can use the RE to NFA and NFA to DFA algorithms described in the textbook, or you can just try an *ad hoc* implementation. (My solution has five states, but you won’t get any extra credit for five or less.)

Problem 2: Specifying Java binary integer literals

Java 7 and Python 3 and C++14 support binary integer literals. Here is how the integer 1999 can be written in binary integer literals.

- `0b11111100011`
- `0B00000011111100011`
- `0b0000_0111_1110_0011`

Here are the rules for binary integer literals:

- Every binary number starts with either 0b or 0B
- This is followed by a string one are more binary digits, 0 or 1, or the underscore _ (which was added for readability)
- The underscores can only be placed between the digits.

Here are four examples of "good" binary integer literals:

- `0`, `0b0`, `0B0`, `0b11_011_0_010`

There five are not binary integer literals

- `0b`, `b0`, `0b_0`, `0b0_`, `0b1______1`

Problem 2A: Specifying Java binary integer literals with an RE

Write a regular expression that matches in binary integer literal. You can use either ∪ or | for specifying alternatives. I suggest you following the textbook author and use R^+ as a "shorthand" for $R\ R^*$. (I needed 21 characters for `grep`.)

Problem 2B: Specifying Java binary integer literals with a CFG

I know a compiler guru would never to this, but write a context-free grammar for the binary literals. You can use the two productions at the bottom of the page.

- `P → b | B`
- `D → 0 | 1`

That B is the terminal **not** a variable.

Problem 3: Yet more CFG

Noam Chomsky has his 91st birthday two days ago, so let's try a little CNF.

Below is a large adaption of an answer to the Exam 3 CFG to CNF question. **T** is the start variable. Circle the productions that do not conform to the requirements of CNF **and *explain*** what requirement they have broken.

- **T → 1M0 | M**
- **M → BT | B**
- **B → 01 | ε**

Next change the bad CNF to good CNF by modifying the productions. You can run the algorithm or you can just apply patches.

- **T → 1M0 | M**
- **M → BT | B**
- **B → 01 | ε**

Problem 4: Countable? Decidable?

Problem 4A: Counting

Decimal fractions (according to Wikipedia) are numbers of the form $i + f/10^n$ for integers i, f, and n. You can also think of them as numbers with stuff after the decimal point, such as 3.1415926535897932384626433832795028841971.

Example why the set of decimal fractions are countable. You don't need much space for this.

Problem 4A: Deciding

According to Jeffrey Shalit, it is in unknown if the following problem is decidable:

> Given a finite automaton A over the alphabet {0,1}, does A accept the base-2 representation of at least one prime number? This is currently not known to be either decidable or undecidable.

Describe in the space left below what you have have to do to solve this problem (and win a in a footnotes in theory of computation textbooks).

This is the three questions for a “take-home” section of the final. UNC Asheville requires all students to be assessed in their writing skills within their major and CSCI 434 is a course where this assessment is required. (I learned this last week.)

Here are **three** problems to solve with nicely written answers. Give a solution for the problem appropriate for your fellow classmates *and* your professor.

All problems are based on a variation of the middle thirds problem (Problem 4.48) of the textbook which involves two languages from the alphabet $\Sigma = \{0,1\}$

- D_1, the "language of all strings that contain a 1 in their middle third"
- D_2, the "language of all strings that contain two 1's in their middle third"

I am re-interpreting this as meaning exactly one 1 in D1 and two 1's in D2. I think this makes it a little easier to understand, solve, and illustrate. That is:

- $D_1 = \{xyz \mid x, y, z \in (0+1)^*$ and $|x| = |y| = |z|$

  where $x \in 0^*$, $y \in 0^*10^*$, and $z \in 0^*\}$

- $D_2 = \{xyz \mid x, y, z \in (0+1)^*$ and $|x| = |y| = |z|$

  where $x \in 0^*$, $y \in 0^*10^*10^*$, and $z \in 0^*\}$

Or, equivalently, D_1 and D_2 are of size $3n$ for some integer $n \geq 1$ and are composed only of 0’s except that the **middle** third of D_1 contains exactly one 1 and the **middle** third of D_2 contains exactly two 1’s.

For example, 010 and 000100 are in D_1, and 001100 and 000101000 are in D_1, but ϵ, 0, 1, 000, 0110, and 000111000 are in neither D_1 nor D_2.

Here are the three proofs you are to make. Try to keep each at a couple of paragraphs.

Mini Writing Program 1

Show that D_1 is a context-free language with a well-written argument that the following context-free grammar generates D_1.

$D \rightarrow 010 \mid 00D0 \mid 0D00$

I suggest using a proof by induction (pp 22-25 of the textbook). Start by thinking of why the 3 strings in D of size 6 can be safely extended to 5 strings of size 9.

Mini Writing Problem 2

Show that D_2 is **not** a context-free language.

Yes, you ***must*** use the Pumping Lemma (Theorem 2.34, p 125).

Remember that you do ***not*** choose the pumping length, the Pumping Lemma does. You **call** the Pumping Lemma and it **returns** *a* pumping length that you may use to cleverly choose a magic string *s* that can be divided into the *uvwxy*.

You can **not** start with something like:

Consider the string 000100. Let’s assume that *p* is 3.

However this is OK:

Suppose p is the pumping length. Consider the string 0^{p*434}oats1^p ...

Mini Writing Problem 3

Create a Turing machine to decide D_2.

Do ***not*** draw a state diagram, such as the one seen in Figure 3.10 (p 173).

Use Example 3.11 (p 174) as your model. Make a list of numbered actions. Use English to describe these actions. Use phrase “mark the X” (see the discussion of marking on page 175) or “cross off the Y” or “scan to the next Z”. “Move to the 2nd X after the 3rd Y” is also OK as is “If X is marked, goto step 7”.

What is allowed and not allowed?

It is OK to discuss the algorithms in a general way. For example, you could gather around a whiteboard and animate the actions of the Turing machine. You could also illustrate the kinds of strings that could generated by the grammar shown for Problem 1.

It’s a bit like a Literature assignment: You can’t copy the phrases of others. Also, you can’t write the solution jointly. Remember, I am required to assess **your** writing.

How to turn it in?

We’ve had three exams given in 100 minute periods. The final is given in a 150 minute period. The in-class part of the final will be targeted for completion in about 75 to 90 minutes. (You are allowed to stay for the entire 150.)

You could try writing these proofs during class, but I ***strongly*** recommend against that. I suggest you write up these proofs and submit them to the Moodle before Thursday, 10 December. I am OK with the usual formats: shared Google Doc, PDF, OpenOffice, MS Word, LaTeX, ...

Bringing a printed copy to class would also be appreciated.