Homework 0 – ECS 120, Winter 2020

Due date: Friday, April 3, 11:59pm

Instructions

Purpose. This homework is intended to ensure that you have mastered (some) background knowledge for this course covered in the prerequisite ECS 20: Discrete mathematics for computer science or MAT 108: Introduction to abstract mathematics.

Submission. Your solutions to these problems are to be submitted on the Gradescope page for ECS 120: https://gradescope.com/courses/92568. See the syllabus (https://canvas.ucdavis.edu/courses/457885/assignments/syllabus) for instructions on how to access the Gradescope page for this course. Each problem below appears as a separate "Assignment" in Gradescope (for technical reasons, Gradescope calls them "Programming" assignments, but they don't involve any programming), and they will each be auto-graded individually on Gradescope. If you submit after the deadline above, Gradescope will automatically apply a late penalty as described in the syllabus. Gradescope will no longer accept submissions 24 hours after the deadline.

Gradescope will not automatically use your maximum score on any submission as your grade; instead it uses the most recent by default. To override this, on the bottom of the assignment page, click on *Submission History* to select which submission you want to "activate".

Randomized problems. The problems below marked **(randomized)** require interacting with Gradescope in a strange way. The idea is to have Gradescope randomly generate data for a problem (so different students will receive different data although the fundamental problem remains the same), then you submit a solution, then Gradescope automatically grades your solution. Gradescope doesn't support this workflow directly; to implement it we will use the following hack:

For the randomized problems, you first *request* a problem by submitting a file named `req` (no filename extension, so *don't* name it `req.txt`; note that often a default on Windows and Mac is to hide the file extension, so you will have to disable this feature to ensure the file has no extension). Gradescope will autograde this as 0 points (don't take it personally), and then give text "feedback": this text describes the problem you are supposed to solve. On your *next* submission, submit your solution in a **plain text file** (not Word or RTF or some other format; it should be a plain ASCII text file, just like you would use for a C program), which can have any name ending in the extension `.txt`. Note that some programs (such as TextEdit on Mac), by default make `.rtf` files with extra non-ASCII characters in them; an editor such as vim, Atom, or Sublime is better. Whatever score you get will replace the 0 that you got when submitting the `req` file.

Carefully check your answer before submitting to make sure you don't have a typo. One tip is to compare against the example answers given below. **Check your formatting carefully!** The auto-grader can only parse submissions that are formatted as shown in the examples.

Auto-graded problems

power set: **(randomized)** Given set X, find its power set $\mathcal{P}(X)$, the set of all subsets of X. Use semicolons between sets in the power set, and commas between elements within each set. You can have any amount of whitespace.

For example, if you were given the set $X =$ `{a,b,c}`, you could write its power set $\mathcal{P}(X)$ as

```
{ {};
  {a};
  {b};
  {c};
  {a,b};
  {a,c};
  {b,c};
  {a,b,c}
}
```

evaluate Boolean formulas **(randomized)** You will be given several Boolean formulas with the same input variables, using operations `and`, `or`, `!` (negation), `=>` (implies), `<=>` (iff). The formulas are given as text on separate lines, followed by several assignments of values to the variables, for example

```
(((a or !b) and c) and (b or c)) or !a
(c <=> !a) or (b => (!a and c))
a=1  b=1  c=0
a=0  b=0  c=0
a=1  b=1  c=1
```

You must evaluate the formulas on these inputs, writing the result as a series of bits. Use one line per formula, and within a line, put a space between the answers. Since the example above has two formulas and three assignments, the correct submission has two lines, with three output bits on each line:

```
0 1 1
1 1 0
```

1-1 and onto functions **(randomized)** You will be given several functions and asked to indicate whether they are 1-1 or onto (or both, or neither). For example, if the functions are

```
f1: {a,b} x {0,1} --> {w,x,y,z} defined by
    f1(a,0) = z
    f1(a,1) = y
    f1(b,0) = w
    f1(b,1) = x

f2: {a,b} x {0,1} --> {w,x,y,z} defined by
    f2(a,0) = x
    f2(a,1) = y
```

```
    f2(b,0) = w
    f2(b,1) = x

f3: {u,w,x,y,z} --> {a,b} x {0,1} defined by
    f3(u) = (a,1)
    f3(w) = (a,0)
    f3(x) = (b,0)
    f3(y) = (b,1)
    f3(z) = (a,1)

f4: {x,y,z} --> {a,b} x {0,1} defined by
    f4(x) = (a,0)
    f4(y) = (b,0)
    f4(z) = (b,1)
```

Then `f1` is 1-1 and onto, `f2` is neither 1-1 nor onto, `f3` is not 1-1, but it is onto, and `f4` is 1-1, but not onto. Thus, the answers in order are "yes, yes", "no, no", "no, yes", and "yes, no". Indicate this by writing these answers (as Boolean bits) two per line:

```
1 1
0 0
0 1
1 0
```

combinatorics **(randomized)** You will be given several sets and asked to determine their cardinalities. For example, if the sets are

`A = { x ∈ {0,1}^6 | x[2..4] = 010 }`

`B = { f: {1,2,3,4} → {0,1} }`

`C = { f: {1,2,3} → {1,2,3,4} | f is 1-1 }`

`D = { x ∈ {0,1}^3 ∪ {0,1}^5 | #_0(x) = #_1(x)+1 }`

`E = { n ∈ {10,11,...,29} | n ≡ 1 mod 3 }`

In English, the above sets are: A = all binary strings of length 6 with substring 010 at indices 2,3,4. (Note that the length of a substring from i to j is $j - i + 1$.) B = all functions with domain $\{1, 2, 3, 4\}$ and range $\{0, 1\}$. C = all 1-1 functions with domain $\{1, 2, 3\}$ and range $\{1, 2, 3, 4\}$. D = all binary strings of length 3 or 5 that have one more 0 than they do a 1. E = all positive integers between 10 and 29 that are congruent to 1 mod 3 (i.e., the remainder is 1 after dividing by 3). Then $|A| = 8$, $|B| = 16$, $|C| = 24$, $|D| = 13$, $|E| = 7$. Indicate this by writing these numbers one per line:

```
8
16
24
13
7
```

Note: For the sake of understanding the problem statement, the examples given here are small, so small that you could solve the problem by brute force, simply writing down all the elements of the set. However, the sets you will be given on Gradescope will be much larger, so you will need to understand how to do this counting "algorithmically", without resorting to brute force. For example, the set A above can generalize to $A_{n,i,j} = \{x \in \{0,1\}^n \mid x[i..j] = \text{some string}\}$; how would you express $|A_{n,i,j}|$ as a function of n, i, j? To calculate the numbers for the answers, you will need to use an arbitrary-precision calculator, such as the Python interpreter (https://repl.it/languages/python3) or Wolfram Alpha (https://www.wolframalpha.com/). A normal calculator will convert large integers to floating-point (e.g., 1.9158123e+20) but these will not work as answers; the exact integer must be entered.

stringology **(randomized)** You will be given finite languages and asked to give various other finite languages obtained by common language operations. For example, suppose the sets are:

```
A = { "f", "n", "p" }
C = { "a", "bab" }
D = { "abb", "bb" }
X = { "10111", "00101", "01001", "11001", "10001" }
```

Suppose further that the problem asks you to compute

- C^2
- CD
- $L = \{ w \in C^* \mid |w| \leq 5 \}$
- $C \times D$
- $C \cup D$
- $C^2 \cup D^2$
- $(C \cup D)^2$
- A^2 as a tuple in length-lexicographical order
- X as a tuple in length-lexicographical order
- $S = \{ w \mid x \in X, w \text{ is a substring of } x, |w| = 4 \}$

Answer this by writing these sets one per line, using Python notation for string, set, and tuple literals (this differs from standard mathematical notation by the use of quotation marks).

```
{ "aa", "abab", "baba", "babbab" }
{ "aabb", "abb", "bababb", "babbb" }
{ "", "a", "aa", "aaa", "bab", "aaaa", "abab",
  "baba", "aaaaa", "aabab", "ababa", "babaa" }
{ ("a", "abb"), ("a", "bb"), ("bab", "abb"), ("bab", "bb") }
{ "a", "bab", "abb", "bb" }
{ "aa", "abab", "baba", "babbab", "abbabb", "abbbb", "bbabb", "bbbb" }
{ "aa", "abab", "aabb", "abb", "baba", "babbab", "bababb", "babbb", "abba",
```

```
  "abbbab", "abbabb", "abbbb", "bba", "bbbab", "bbabb", "bbbb" }
( "ff", "fn", "fp", "nf", "nn", "np", "pf", "pn", "pp" )
( "00101", "01001", "10001", "10111", "11001" )
{ "1011", "0111", "0010", "0101", "0100", "1001", "1100", "1000", "0001" }
```

Note: Some long lines are displayed in the above example answer crossing several lines, so that they will fit in the displayed PDF page. However, you must write the sets/tuples **one per line** in your text file. That means your text file submission should have 10 non-empty lines. (Empty lines will be ignored.) Note that the empty string ε is indicated by `""`, and tuple answers are surrounded by parentheses, and sets are surrounded with curly braces.