CSE 311: Foundations of Computing I Spring 2025

Problem Set 7

Due: Wednesday, May 28th by 11:00pm

Instructions

Submit your solutions in Gradescope. Your Gradescope submission should follow these rules:

- Each numbered task should be solved on its own page (or pages). Do not write your name on the individual pages. (Gradescope will handle that.)
- When you upload your pages, make sure each one is **properly rotated**. If not, you can use the Gradescope controls to turn them to the proper orientation.
- Follow the Gradescope prompt to **link tasks to pages**. You do not need to link tasks that you did not include, e.g., Task 1f (ungraded) and Task 7 (extra credit).
- You are not required to typeset your solution, but your submission must be **legible**. It is your responsibility to make sure solutions are readable — we will *not* grade unreadable write-ups.

Task 1 – i l$*$vε rεg$\cup 10r$ εxpr$\varepsilon 2210$n2 [10 pts]

For each of the following, construct regular expressions that match the given set of strings. Write a brief explanation of your regular expression's design and why you believe it recognizes exactly the language (and no strings that are not in the language).

a) *Syllables*, which are defined for the sake of this problem as [1] optionally starting with only one consonant letter, [2] followed by one or more vowels, [3] optionally ending with another consonant letter. For example, "oar", "car", and "queue" are syllables, whereas "regular" is not (but "reg", "u", and "lar" all are). You may use "$b \cup c \cup \ldots \cup z$" as shorthand for "all consonants". For the sake of this problem, y is exclusively a consonant. The following letter are vowels: a, e, i, o, u.

b) *Positive integer solutions to the equation* $x \equiv_5 1$ (written in base-10). You may write $1 \cup 2 \cup \ldots \cup 9$ as shorthand for $1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9$. In general, if you write $i \cup (i+1) \cup \ldots \cup j$, we will assume you are applying $\cup$ for all integers from i to j inclusive. Note that leading zeros are not valid (e.g., accept 321 but not 00321).

c) *Binary strings with an odd number of 1s.*

d) *Binary strings which do not have any 3 or more consecutive repetitions of a digit* (e.g., the following would *not* belong to this language: 111, 1000110, 0011000011).

e) *Years Rick Astley was alive* (1966 to present, 2025). That is, all numbers from 1966 to 2025, inclusive. Your expression must not be "brute force" (e.g., $1966 \cup 1967 \cup \ldots \cup 2025$): you should **never** have more than 10 things unioned together at once (e.g., $1 \cup 2 \cup \ldots \cup 20$, although if you are **gonna** write something like $(1 \cup 2 \cup \ldots \cup 10)(11 \cup 12 \cup \ldots \cup 20)$, that is okay). **Give** extra thought to breaking down the structure of this language to avoid brute forcing. Like before, **you** may use $1 \cup 2 \cup \ldots \cup 9$ as shorthand for $1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9$. Don't give **up**!

f) *Choose your own real-life example of a regular language and write a regular expression for it.* Some examples to try out: CSS hex codes, phone numbers, Roman numerals, URLs, email addresses, ordinal numbers (e.g., "1st", "2nd", "5th", "50th", "51st"), command line options, license plates, arithmetic expressions without parentheses, basic human conversations, etc. This problem will be **ungraded** so you may leave it empty, but this is an opportunity to deepen your understanding and appreciation for regular expressions. If you make a submission, staff will comment on it!

Submit and check your answers to parts **(c)** and **(d)** here:

`http://grin.cs.washington.edu`

Think carefully about your answer to make sure it is correct before submitting. You have only **5 chances** to submit a correct answer.

Note: Parts (a-b) and (e-f) should be submitted in Gradescope.

Task 2 – Glitz and Grammar [15 pts]

For each of the following, construct a context-free grammar that generates the given set of strings.

If your grammar has more than one variable (also called a non-terminal symbol), write a sentence describing what sets of strings you expect each variable in your grammar to generate. For example, if your grammar was:

$$\begin{aligned} \mathbf{S} &\to \mathbf{E} \mid \mathbf{O} \\ \mathbf{O} &\to \mathbf{EC} \\ \mathbf{E} &\to \mathbf{EE} \mid \mathbf{CC} \\ \mathbf{C} &\to 0 \mid 1 \end{aligned}$$

You could say "C generates binary strings of length one, E generates (non-empty) even length binary strings, and O generates odd length binary strings." It is also fine to use a regular expression, rather than English, to describe the strings generated by a variable (assuming such a regular expression exists). If you only use one variable (i.e., S), an explanation is not required but you should provide the grammar.

a) Binary strings matching the regular expression "$10(01 \cup 1)^*11 \cup 0110$".
Hint: You can use the technique described in lecture to convert this RE to a CFG.

b) All strings of the form $\#y\#x$, with $x, y \in \{0,1\}^*$ and y is x^R, but with each of the characters optionally doubled.
(Here x^R means the reverse of x.)

c) All binary strings in the set $\{1^m0^n : m, n \in \mathbb{N} \text{ and } m > 2n + 1\}$.

Submit and check your answers to this question here:

http://grin.cs.washington.edu

Think carefully about your answer to make sure it is correct before submitting. You have only **5 chances** to submit a correct answer.

Note: You must also include each grammar and sentences describing each new non-terminal, as described above, in Gradescope.

Task 3 – Hope Strings Eternal [15 pts]

In this problem, we will demonstrate by example that context-free grammars (with exactly one variable) and recursively defined sets are capable of describing exactly the same languages.

As an example, consider binary strings of even length. We can define a grammar for them as follows:

$$\mathbf{S} \to \mathbf{S}00 \mid \mathbf{S}01 \mid \mathbf{S}10 \mid \mathbf{S}11 \mid \varepsilon$$

To do the same thing with recursion, we first define a set $\mathcal{T}$ that records a sequence of substitutions used by the grammar. This can be done as follows:

Basis: Empty $\in \mathcal{T}$
Recursive Step: If $L \in \mathcal{T}$ and $a, b \in \{0, 1\}$, then Append$(L, a, b) \in \mathcal{T}$

For example, the derivation $S \to S01 \to S1101 \to \varepsilon1101$ would be represented by the tree

$$\text{Append}(\text{Append}(\text{Empty}, 1, 1), 0, 1)$$

Note that Append is not a function call! Append(L, a, b) is data, not code. We saw the same thing in Homework 6 Task 6, where Tree(L, R) denotes a node in the tree. Here, Append makes tree nodes. (Recursively defined sets that keep track of the operation of a grammar are generically referred to as "parse trees". Our set $\mathcal{T}$ consists of parse trees for the grammar **S** above.)

To get the string in the language, we define the following recursive function that traverses the tree and outputs the string that was generated:

$$\text{traverse}(\text{Empty}) := \varepsilon$$
$$\text{traverse}(\text{Append}(L, a, b)) := \text{traverse}(L)ab$$

Note that traverse returns a string, so on the second line, traverse$(L)ab$ means that we make a recursive call to traverse(L) and then append characters 'a' and 'b' to the string that it returned.
With this definition, we can see, for example, that

$$\begin{aligned}
&\text{traverse}(\text{Append}(\text{Append}(\text{Empty}, 1, 1), 0, 1)) && \\
&= \text{traverse}(\text{Append}(\text{Empty}, 1, 1))01 && \text{def of traverse} \\
&= \text{traverse}(\text{Empty})1101 && \text{def of traverse} \\
&= \varepsilon1101 && \text{def of traverse}
\end{aligned}$$

showing that the this was indeed a derivation of the string "1101".

Having seen a first example, let's try out some more....

Consider the following grammar:

$$\textbf{S} \to 10\textbf{S} \mid 0\textbf{S} \mid \varepsilon$$

It generates all strings where every 1 is immediately followed by a 0.

a) Write a recursive definition of a set $\mathcal{T}$ of parse trees for this grammar.

b) Define a recursive function that traverses your parse trees from part (a) and outputs the string that was generated by the grammar above.

Be careful to indicate anywhere that string concatenation is used with $\bullet$ rather than juxtaposition. (The latter is just for appending one character to a string.)

Now consider the following grammar:

$$\textbf{S} \to \textbf{S}00 \mid \textbf{S}01 \mid \textbf{S}10 \mid \textbf{S}11 \mid 10$$

It generates all strings of even length starting with 10.

c) Write a recursive definition of a set $\mathcal{T}$ of parse trees for this grammar.

d) Define a recursive function that traverses your parse trees from part (c) and outputs the string that was generated by the grammar above.

Be careful to indicate anywhere that string concatenation is used with $\bullet$ rather than juxtaposition. (The latter is just for appending one character to a string.)

To see the other direction, consider the following recursive definition:

Basis: Start $\in \mathcal{T}$
Recursive Step: If $L \in \mathcal{T}$ and $a \in \{0, 1\}$, then Append$(L, a) \in \mathcal{T}$

And suppose that we generate strings from these with the following traversal:

$$\text{traverse(Start)} := 0$$
$$\text{traverse(Append}(L, a)) := \text{traverse}(L)a$$

The set $S = \{\text{traverse}(L) \ : \ L \in \mathcal{T}\}$, which consists of all strings that can be generated by building a parse tree and then traversing it, contains all binary strings starting with "0".

e) Define a context-free grammar that matches these definitions.

Note that your grammar must not only generate all strings starting with in "0". It must have derivations that exactly correspond to the parse trees above!

Task 4 – Relatin' Canes [12 pts]

One would think that by Week 8, the TA's would stop getting into situations involving their fast food orders, but, alas, it has happened once more. During the TA's trip to Relatin' Canes, an incredibly popular chicken establishment (and, apparently, gathering spot for discrete math nerds), the TA's couldn't figure out what to order, where to sit, and who's going! They have encoded the whole situation into several relations over sets; we need your help to figure out the properties of these relations and solve this once and for all! Last week, we "promised" you a sub(set) sandwich – this week, we ASSURE you that we will bring something back, should you so choose to help! :)

For each of the relations below, determine whether or not it has each of the properties of reflexivity, symmetry, antisymmetry, and/or transitivity. If a relation has a property, simply say so without any further explanation. If a relation does not have a property, state a counterexample, but do not explain your counterexample further.

a) Define $R \subseteq \mathbb{Z} \times \mathbb{Z}$ by $(a, b) \in R$ iff $a + b$ is odd.

b) Define $S \subseteq \mathbb{N} \times \mathbb{N}$ by $(a, b) \in S$ iff $a \cdot b \geqslant 0$.

c) Let $A = \{k \in \mathbb{Z} : k < 0\}$ be the set of negative integers. Define $T \subseteq A \times A$ by $(a, b) \in T$ iff $a \times b > 0$

d) Let $B = \mathcal{P}(\mathbb{N})$. Define $U \subseteq B \times B$ by $(X, Y) \in U$ iff $X \cup [6] \subseteq Y \cap [6]$. (Remember that $[n] = \{1, \ldots, n\}$.)

Hint: What can be in X? What can't be?

Task 5 – Get a Prove On [12 pts]

Let R be a relation on a set A. Recall from lecture that "R is *symmetric*" iff

$$\forall a \in A \, \forall b \in A \, ((a, b) \in R \rightarrow (b, a) \in R)$$

i.e. if a pair (a, b) is in R, then swapped pair (b, a) is also in R.

Now, let R and S be relations on a set A, and consider the following claim:

Given that R and S are symmetric, it follows that $R \setminus S$ is symmetric.

Write an **English proof** that the claim holds.

Note: even though we want you to write your proof directly in English, it must still look like the translation of a formal proof. In particular, you must include all steps that would be required of a formal proof, excepting only those that we have explicitly said are okay to skip in English proofs (e.g., Elim $\exists$).

Hint: Remember that the Absurdum rule can be used to show that a claim is **not** true. Within the direct proof required by Absurdum, the Contradiction rule is needed to prove F.

Task 6 – Better Get Proving [12 pts]

Let R be a relation on a set A. Recall from lecture the definition of the *composition* of two relations:

$$R \circ R := \{(a, b) \mid \exists \mathbf{c}\, ((a, \mathbf{c}) \in R \land (\mathbf{c}, b) \in R)\}$$

and "R is *transitive*" iff

$$\forall a \in A\, \forall b \in A\, \forall c \in A\, (((a, b) \in R \land (b, c) \in R) \to (a, c) \in R)$$

and "R is *antisymmetric*" iff

$$\forall a \in A\, \forall b \in A\, (((a, b) \in R \land (b, a) \in R) \to a = b)$$

The definition of antisymmetric is different from the one in the lecture, but it is equivalent. Using this definition can make the proof slightly easier.

Now, consider the following claim:

> Given that R is transitive and antisymmetric, it follows that $R \circ R$ is antisymmetric.

Write an **English proof** that the claim holds.

Note: even though we want you to write your proof directly in English, it must still look like the translation of a formal proof. In particular, you must include all steps that would be required of a formal proof, excepting only those that we have explicitly said are okay to skip in English proofs (e.g., Elim $\exists$).

Task 7 – Extra Credit: With a Grammar, the Whole World is a Nail [0 pts]

Consider the following context-free grammar.

$$
\begin{array}{ll}
\langle \textbf{Stmt} \rangle & \to \langle \textbf{Assign} \rangle \mid \langle \textbf{IfThen} \rangle \mid \langle \textbf{IfThenElse} \rangle \mid \langle \textbf{BeginEnd} \rangle \\
\langle \textbf{IfThen} \rangle & \to \text{if condition then } \langle \textbf{Stmt} \rangle \\
\langle \textbf{IfThenElse} \rangle & \to \text{if condition then } \langle \textbf{Stmt} \rangle \text{ else } \langle \textbf{Stmt} \rangle \\
\langle \textbf{BeginEnd} \rangle & \to \text{begin } \langle \textbf{StmtList} \rangle \text{ end} \\
\langle \textbf{StmtList} \rangle & \to \langle \textbf{StmtList} \rangle \langle \textbf{Stmt} \rangle \mid \langle \textbf{Stmt} \rangle \\
\langle \textbf{Assign} \rangle & \to \texttt{a := 1}
\end{array}
$$

This is a natural-looking grammar for part of a programming language, but unfortunately the grammar is "ambiguous" in the sense that it can be parsed in different ways (that have distinct meanings).

a) Show an example of a string in the language that has two different parse trees that are meaningfully different (i.e., they represent programs that would behave differently when executed).

b) Give **two different grammars** for this language that are both unambiguous but produce different parse trees from each other.