C1 Theory (25 points)
Let N denote the set of non-negative integers.

a. (5 points)
Explicitly exhibit a regular expression defining

$$L_1 = \{a^{2m+1} \cdot b \mid m \in N\} \cup \{a \cdot b^{2n} \mid n \in N\}. \tag{1}$$

b. (10 points)
Use a Pumping Lemma to prove that

$$L_2 = \{a^m b^n \mid m, n \in N \text{ and } \textit{either } m \text{ is even } \textit{or } n < m\}. \tag{2}$$

is *not* regular.

c. (10 points)
For a (deterministic) finite automaton $\mathcal{M}$, $L(\mathcal{M})$ denotes the language accepted by $\mathcal{M}$.
Show how to directly, algorithmically construct a (deterministic) finite automaton $\mathcal{M}_3 = (A, Q_3, \delta_3, F_3, q_1^3)$ from any two (deterministic) finite automata $\mathcal{M}_1 = (A, Q_1, \delta_1, F_1, q_1^1)$ and $\mathcal{M}_2 = (A, Q_2, \delta_2, F_2, q_1^2)$, where, A is a finite alphabet, the Q_i's are the state sets, the δ_i's are the transition functions, the F_i's are the sets of final (or accepting) states, the q_1^i's are the initial states,

$$Q_3 = Q_1 \times Q_2, \tag{3}$$

and

$$L(\mathcal{M}_3) = L(\mathcal{M}_1) \cap L(\mathcal{M}_2). \tag{4}$$

Hint: Intuitively, have $\mathcal{M}_3$ parallel process the running of $\mathcal{M}_1$ and $\mathcal{M}_2$. That's the point of the **requirement** in (3) above! Explicitly, algorithmically, define δ_3, F_3 and q_1^3 to do the job.

C2 **Theory** (25 points)

Suppose n is a fixed integer > 1. Let $\Sigma = \{a_1, \ldots, a_n\}$. Let

$$L = \{w \in \Sigma^* \mid w \text{ is } \textit{missing} \text{ at least one symbol of } \Sigma\}. \tag{5}$$

In finite automata theory, a *0-move* is (by definition) a machine state change accompanied by *no* forward move of the read head. In some text books 0-moves are called *ε-moves*.

a. (10 points)

Explicitly exhibit the state diagram of a 0-move, non-deterministic finite automaton, $\mathcal{M}_1$ having *exactly* $n + 1$ states and such that $L(\mathcal{M}_1) = L$, where L is defined in (5) above.

Hint: Consider 0-moves from the start state to each of the remaining n states. For the i-th of the n *non*-start states, consider having it jam on the symbol a_i.

b. (5 points)

Explicitly exhibit the state diagram of a non-deterministic finite automaton, $\mathcal{M}_2$ having *exactly* $n + 1$ states and such that $L(\mathcal{M}_2) = L$, *where $\mathcal{M}_2$ has NO zero moves* (and where L is defined in (5) above).

Hint: Compile your answer to Problem C2a to get rid of the 0-moves *or* think how to program it straightaway.

c. (10 points)

For this problem (C2c) you may use the Myhill-Nerode Theorem and/or its standard corollaries without proof; *however*, you must *explicitly, correctly state* any of these results you are using without proof.

Recall that the crucial equivalence relation for Myhill-Nerode, etc. applied to L from (5) above is $\equiv_L$, where, for words $x, y \in \Sigma^*$,

$$x \equiv_L y \overset{\text{def}}{\Leftrightarrow} (\forall z \in \Sigma^*)[xz \in L \Leftrightarrow yz \in L]. \tag{6}$$

Suppose $\mathcal{M}_3$ is *any* (deterministic) finite automaton such that $L(\mathcal{M}_3) = L$ (again, where L is defined in (5) above). *Prove* that $\mathcal{M}_3$ has *at least* 2^n *states*!

Hint: For each $A \subseteq \Sigma$, let $L_A = \{w \mid w \text{ contains each symbol in } A \text{ and no other symbols}\}$.[1]

How many A are subsets of Σ?

Suppose $A \subseteq \Sigma$ and that $x, y \in L_A$. Show that $x \equiv_L y$.

Suppose $A, B \subseteq \Sigma$. Suppose $A \neq B$, that $x \in L_A$, and that $y \in L_B$. Show that $x \not\equiv_L y$. To do this, consider separately the cases where $A \not\subseteq B$ and where $B \not\subseteq A$.

Explain *clearly* how showing the above is *relevant* to solving C2c.

[1] For example, $L_\emptyset = \{0\}$, where 0 denotes the empty word.

C3 **Theory** (25 points)

Let N denote the set of non-negative integers.

For each $x \in N$, let

$$f(x) = \begin{cases} x!, & \text{if, in the decimal expansion of } \pi, \text{ there is a} \\ & \quad \textit{consecutive} \text{ run of 7's of length} \geq x; \\ x^3, & \text{otherwise.} \end{cases}$$

Prove that f is primitive recursive only from the definition of 'primitive recursive' and the assumptions in the next sentence. You may assume with*out* proof that the primitive recursive functions are closed under definition by cases (or if-then-else) *and* that $\leq$ is a primitive recursive binary predicate.

Do put in all the Base Functions (e.g., projection functions), etc. for this problem.

Hint: Do a non-constructive, possible worlds, case analysis. Do *not* try to show messy predicates directly about the decimal expansion of π are primitive recursive!

C4 **Theory** (25 points)

Let N denote the set of non-negative integers. Fix a standard programming formalism φ for computing all the partial computable functions which map arguments from N into N. Code (Gödel) number the φ-programs *onto* the entire set N. Let $\varphi_p^{(n)}$ denote the partial function of $n \geq 1$ arguments computed by program (number) p in the φ-system. When $n = 1$, we omit the superscript. We will refer to p as a φ-program. Let Φ denote a standard Blum step-counting measure associated with φ.[2] You may assume *without* proof that in the φ-system Universality, S-m-n, and the Kleene Recursion Theorem (KRT) hold. Furthermore, you may assume that your favorite assembly language can be compiled into the φ-system!

Definition *Consider all the finite sets of equations defining primitive recursive functions and which contain a special one argument function letter* $\mathbf{f}$ *and whose remaining function letters are chosen from the list* $\mathbf{f_1}, \mathbf{f_2}, \mathbf{f_3}, \ldots$ *. Gödel number (code number) 1-1 onto* N *all these finite sets of equations.*

1. *Let* E_q *be (by definition) the finite set of equations with Gödel number* q*.*
2. $f_q \stackrel{\text{def}}{=}$ *the primitive recursive function which* $\mathbf{f}$ *defines in* E_q*.*[3]

a. (10 points)

Sketch **informally** (with*out* the messy coding/decoding and programming details) a proof that some φ-program computes $\lambda q, x.\, f_q(x)$.[4]

Hint: Think about how one can compile q's representing E_q's *into* your favorite assembly language.
In a step-wise refinement of programming such a compiler, also *in* your favorite assembly language, sketch only the top level refinements.
For example, you *should* give some very top level hints (only) on how to code/decode q, etc.
For another example, for the subtask of compiling a definition by primitive recur*sion* of some function, it suffices to sketch how to implement iteratively this kind of recursion in your favorite assembly language.[5]
Don't forget to *relevantly cite any assumptions given above about* φ *when you are using them.*

b. (15 points)

Suppose η and θ are any partial functions mapping N into N. $\eta(x)\!\downarrow \stackrel{\text{def}}{\Leftrightarrow} \eta(x)$ is defined.

$\eta =^{\infty} \theta \stackrel{\text{def}}{\Leftrightarrow} (\stackrel{\infty}{\exists} x)[\eta(x) = \theta(x)]$.[6]
Assuming what you need re standard primitive recursive functions and their standard closure properties, *prove* the following

Theorem *It is NOT the case that there is a partial computable* ψ *such that, for all* p *for which* φ_p *is primitive recursive, we have that* $\psi(p)\!\downarrow$ and

$$f_{\psi(p)} =^{\infty} \varphi_p. \tag{7}$$

This theorem says that one can*not* algorithmically transform any φ-program for a primitive recursive function into a finite set of equations defining a primitive recursive function so that the two primitive recursive functions agree at least infinitely often!

Hint: Suppose for contradiction otherwise. By KRT there is a self-referential φ-program e such that e, on input x, checks whether or not $\psi(e)\!\downarrow$ in $\leq x$ steps (with respect to some fixed φ-program for ψ) and ... . What else should e do? You may use without proof the result of Problem C4a. Show that your e yields a contradiction.

---

[2] Hence, (i) $(\forall p)[\text{domain}(\Phi_p) = \text{domain}(\varphi_p)]$, *and* (ii) $[\{(p, x, t) \mid \Phi_p(x) \leq t\}$ is an algorithmically decidable set].

[3] For *example*, consider equations by which $\mathbf{f_1}$ defines two-argument addition as primitive recursive (in the usual way), $\mathbf{f_2}$ defines two-argument multiplication as primitive recursive (using $\mathbf{f_1}$ in the usual way), and $\mathbf{f}$ defines factorial as primitive recursive (using $\mathbf{f_2}$ in the usual way). Let E be all and only the resultant finite set of equations. Let q be E's code number. Then $E_q = E$ and f_q is the factorial function.

[4] Church's lambda-notation is used to name functions defined by formulas (as is standard in Lisp). For example, $\lambda x.[x + 1]$ denotes the function that maps each non-negative integer x to $x + 1$; and (relevantly to this problem) $\lambda q, x.\, f_q(x)$ denotes that function of **two** non-negative integer arguments which, on input (q, x), returns the value $f_q(x)$.

[5] If you know and like it, you can use for your favorite assembly language the assembly(-like) language $\mathcal{L}$ from the textbook by Davis, Sigal and Weyuker.

[6] The quantifier '$(\stackrel{\infty}{\exists} x)$' means 'there are infinitely many natural numbers x such that'.