C1 **Theory** (25 points)

a. (6.25 points)
Show that

$$L_1 = \{w \in \{a,b\}^* \mid w \text{ contains the subword } bba \text{ or } w \text{ ends in } baa\} \quad (1)$$

is regular. You may use with*out* proof any standard text book results about regular sets *provided you clearly say which results you are using when.*[1]

b. (6.25 points)
Find a *deterministic* finite automata $\mathcal{M}'$ which accepts the same language (over $\{a,b\}$) as the non-deterministic finite automata $\mathcal{M}$ depicted in table form just below.

δ	a	b
start 1	$\{1\}$	$\{1,2\}$
2	$\{3\}$	$\{3\}$
3	$\{4\}$	$\{4\}$
final 4	$\emptyset$	$\emptyset$.

c. (6.25 points)
Employ an appropriate pumping lemma to show that

$$L_2 = \{a^n b^m \mid n = 2m\} \quad (2)$$

is *not* regular.

d. (6.25 points)
Employ an appropriate pumping lemma to show that

$$L_3 = \{a^m b^n \mid m,n \in N \text{ and } \textit{either } m \text{ is odd } \textit{or } n < m\} \quad (3)$$

is *not* regular.

[1] L_1 is *not* a standard text book regular language. (☺)

C2 **Theory** (25 points)

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

$$L = \{w \in \Sigma^* \mid w \text{ is } \textit{missing} \text{ at least one symbol of } \Sigma\}. \tag{4}$$

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 (4) 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 (4) 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 (4) above is $\equiv_L$, where, for words $x, y \in \Sigma^*$,

$$x \equiv_L y \overset{\text{def}}{\Leftrightarrow} (\forall z \in \Sigma^*)[xz \in L \Leftrightarrow yz \in L]. \tag{5}$$

Suppose $\mathcal{M}_3$ is *any* (deterministic) finite automaton such that $L(\mathcal{M}_3) = L$ (again, where L is defined in (4) above). *Prove* that $\mathcal{M}_3$ has *at least* 2^n *states*!

Hint: For each $A \subseteq \Sigma$, let $L_A = \{w \mid w \text{ contains each symbol in } A \text{ and no other symbols}\}$.[2]

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.

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

C3 **Theory** (25 points)

a. (12.5 points)

It is convenient to use the symbol '$\circ$' for functional composition: $f \circ g(x)$ denotes $f(g(x))$.

Definition (Iterated Self-Composition) *For each natural number k,*

$$g^{(k)}(x) \stackrel{\text{def}}{=} \overbrace{g \circ g \circ \cdots \circ g}^{k\ g\text{'s}}(x). \tag{6}$$

For example, $g^{(3)}(x) = g(g(g(x)))$.[3]

Suppose g is primitive recursive (mapping the natural numbers into the natural numbers). For each pair of natural numbers $x \,\&\, y$, let

$$h(x, y) = g^{(y)}(x). \tag{7}$$

Show that h is also primitive recursive.[4]

b. (12.5points)

Suppose g is primitive recursive, mapping the natural numbers into the natural numbers. For each natural number x, let

$$f(x, 0) = g(x), \tag{8}$$

and, for each natural number y, let

$$f(x, y + 1) = f(f(x, y), y). \tag{9}$$

Clearly Equations (8) and (9) do *not* define f from g *by primitive* recursion.

Show that, nonetheless, f *is* primitive recursive!

For full credit, you need *not* explicitly detail all the projection or other base functions you use.

Hint: Try computing a few values in the sequence $f(x, 0), f(x, 1), f(x, 2), \ldots$ until you see a relevant pattern you can *and do* subsequently exploit.

---

[3] Of course, $g^{(0)}(x) = x$.

[4] For full credit, you need *not* explicitly detail all the projection or other base functions you use.

C4 **Theory** (25 points)

Let N denote the set of non-negative integers.

If ψ is a partial function mapping N into N, then $\delta\psi$ denotes the *domain* of ψ, i.e., $\{x \mid (\exists y)[\psi(x) = y]\}$, and $\rho\psi$ denotes the *range* of ψ, i.e., $\{y \mid (\exists x)[\psi(x) = y]\}$. For partial functions ψ, θ mapping N into N, $\psi =^* \theta$ means that $(\exists w)(\forall x \geq w)[\psi(x) = \theta(x)]$.

Fix a standard programming formalism φ for computing all the *one-argument* partial computable functions which map N into N.[5] Code (Gödel) number the φ-programs *onto* N. Let φ_p denote the partial function computed by program (number) p in the φ-system.

Let $\Phi_p(x) \stackrel{\text{def}}{=}$ the number of steps φ-program p executes on input x if p on x halts and undefined if p on x does not halt. You may assume: $\Phi_p(x)$ defines a *partially* computable function of p, x, $\Phi_p(x)$ is defined exactly when $\varphi_p(x)$ is defined, and

$$\{(p, x, t) \mid \Phi_p(x) \leq t\} \text{ is a computable set.} \tag{10}$$

We write $\Phi_p(x) > t$ to mean *either* $\Phi_p(x)$ is defined to be some number $> t$ *or* $\Phi_p(x)$ is undefined.

You may assume with*out* proof that, in the φ-system, Universality, S-m-n, and the Kleene Recursion Theorem (KRT) hold.

You may *assume* there is a fixed coding from N 1-1 onto the set of finite (partial) functions each mapping a finite subset of N into N. Let F_z be the finite function with code number z in this coding. You may assume there is an algorithm for extracting from any $z \in N$ a table of all and only the input/output pairs for F_z.

You may also *assume* with*out* proof that the following theorem holds.

Theorem *Suppose H is any computable function.*[6] *Then there is a computable function f such that*

$$\rho f \subseteq \{0, 1\} \tag{11}$$

and

$$(\forall q \in N \mid \varphi_q = f)(\exists w)(\forall x \geq w)[\Phi_q(x) > H(x)].^{7} \tag{12}$$

In your working this problem, clearly say which assumptions you are using when. Not all assumptions are needed.

a. (6.25 points)

Show that there is a computable *function* q such that, for all x, p, z,

$$\varphi_{q(p,z)}(x) = \begin{cases} F_z(x), & \text{if } x \in \delta F_z; \\ \varphi_p(x), & \text{otherwise.} \end{cases} \tag{13}$$

b. (6.25 points)

Suppose h is any computable function mapping N into N.

For each $x \in N$, let

$$H(x) = \max(\{\Phi_{q(p,z)}(x) \mid [p, z \leq x \ \wedge \ \Phi_p(x) \leq h(x)]\}). \tag{14}$$

Informally *show* that H is computable.

(Problem C4 continues onto the next page.)

[5] N.B. We are using a *lower* case Greek phi for this notation. Below we employ the upper case of this Greek letter but for a different purpose.

[6] This theorem holds for any computable H, *but* it is especially interesting when H is very fast growing and with large values.

[7] (12) implies that *every* program for f has run time on *all but finitely many inputs* worse than H of those inputs (again, H may be horrendous). Intuitively: f is *inherently* more computationally complex than H on all but finitely many inputs; there is *no* fast way to compute f. Furthermore, f's inherent computational complexity is not because it's output values are so large it takes a long time to output them — by (11), the output values are in $\{0, 1\}$ — instead, the (inherent) computational complexity is in figuring out *which* of 0 and 1 to output.

(This page is the continuation of Problem C4.)

c. (6.25 points)

Let f a computable function which then exists, for the H just above, from the assumed Theorem above. Suppose p is a φ-program such that

$$\varphi_p =^* f. \tag{15}$$

For the computable *function* q from C4a above, *show* that

$$(\exists z)[\varphi_{q(p,z)} = f]. \tag{16}$$

d. (6.25 points)

Under the assumptions above, *prove* that

$$(\exists w)(\forall x \geq w)[\Phi_p(x) > h(x)]. \tag{17}$$

Hint: Let z be from C4c above. Apply the assumed theorem above to the φ-*program* $q(p,z)$.[8] Consider x suitably large and suppose for contradiction $\Phi_p(x) \leq h(x)$. *You* have to say how large is suitably large: make $x \geq$ some bound from your applying the *assumed* theorem above to φ-*program* $q(p,z)$ *and also* make $x \geq$ both p, z. Then, chain some inequalities to get a contradiction.

[8] While q in *this* context is a (computable) ***function***, its ***value*** at (p,z), i.e., $q(p,z)$, is, of course, the code ***number*** of a program.