C1 **Theory** (25 points)

a. (6.25 points)
Show that

$$L_1 = \{w \in \{a,b\}^* \mid \text{80-th from last symbol of } w \text{ is } a \text{ } or \text{ 23-rd from last symbol of } w \text{ is } b\} \quad (1)$$

is regular. You may use without proof any standard text book results about regular sets *provided you clearly say which results you are using when.*[1]

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

$$L_2 = \{w \cdot w^R \mid w \in \{a,b\}^*\} \quad (2)$$

is *not* regular, *where* w^R is (by definition) w written in reverse. For example, $abb^R = bba$, and, hence, $abbbba \in L_2$.

c. (6.25 points)
Explicitly program a push-down automaton which accepts by final state all and only the strings in

$$L_3 = \{a^n b^{2n} \mid n > 0\}. \quad (3)$$

d. (6.25 points)
Employ an appropriate pumping lemma to show that *no* push-down automaton accepts all and only the strings in

$$L_4 = \{a^n b^n a^n \mid n \geq 0\}. \quad (4)$$

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

C2 **Theory** (25 points)

Suppose Σ is a finite (non-empty) alphabet. Suppose $x, y \in \Sigma^*$. We write $x \parallel y$ for the *set* of all strings that can be obtained by shuffling *strings* x and y together like a deck of cards; for example,

$$ab \parallel cd = \{abcd, acbd, acdb, cabd, cadb, cdab\}. \tag{5}$$

0 denotes the empty word.

A recursive definition of the result of shuffling two strings together is as follows, where $x, y \in \Sigma^*$ and $s_1, s_2 \in \Sigma$.

$$0 \parallel y \stackrel{\text{def}}{=} \{y\}; \tag{6}$$

$$x \parallel 0 \stackrel{\text{def}}{=} \{x\}; \tag{7}$$

$$xs_1 \parallel ys_2 \stackrel{\text{def}}{=} (x \parallel ys_2) \cdot \{s_1\} \cup (xs_1 \parallel y) \cdot \{s_2\}. \tag{8}$$

The *shuffle* of two languages L_1 and L_2, denoted $L_1 \parallel L_2$, is the set of all strings obtained by shuffling a string from L_1 with a string from L_2:

$$L_1 \parallel L_2 \stackrel{\text{def}}{=} \bigcup_{x \in L_1 \ \wedge \ y \in L_2} x \parallel y. \tag{9}$$

For example,

$$\{ab\} \parallel \{cd, e\} = \{abe, aeb, eab, abcd, acbd, acdb, cabd, cadb, cdab\}. \tag{10}$$

Show how to algorithmically construct, from (deterministic) finite automata $\mathcal{M}_1$ and $\mathcal{M}_2$ each over Σ, a non-deterministic finite automata $\mathcal{M}_3$ also over Σ such that

$$L(\mathcal{M}_3) = L(\mathcal{M}_1) \parallel L(\mathcal{M}_2). \tag{11}$$

For each $i \in \{1, 2\}$, write $\mathcal{M}_i = (\Sigma, Q_i, \delta_i, F_i, q_1^i)$.

It is ok to draw a diagram, **but**, if **your** $\mathcal{M}_3 = (\Sigma, Q_3, \delta_3, F_3, q_1^3)$, **you must provide explicit mathematical formulas for** Q_3, δ_3, F_3, and q_1^3 — in terms of Q_i, δ_i, F_i, and q_1^i, for $i \in \{1, 2\}$.

Hint: Have your $\mathcal{M}_3$ run $\mathcal{M}_1$ and $\mathcal{M}_2$ deciding non-deterministically which one will be active at any point in the input string.

C3 **Theory** (25 points)

Let $f(0) = 0$, $f(1) = 1$, $f(2) = 2^2$, $f(3) = 3^{(3^3)} = 3^{27}$, $f(4) = 4^{(4^{(4^4)})}$, and, in general, for *positive* n, $f(n)$ is written as a (right-associated) stack n high of n's as base and exponents.

Prove that f is primitive recursive. You may and should proceed informally, with*out* putting in all the Base Functions, etc. You may use with*out* proof the primitive recursiveness of standard text book primitive recursive functions[2] and any standard text book closure properties of the class of primitive recursive functions *provided you clearly say which ones you are using when.*

Hint: We employ here Church's lambda-notation 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 $\lambda x, y.(x + y)$ denotes the function of two non-negative integer arguments mapping them to their sum.

For dealing with ordinary exponentiation, you may and should adopt the convention that $0^0 = 1$ so that ordinary exponentiation, $\lambda x, y.(x^y)$, is total and is, then, a standard text book primitive recursive function.

First work with *a* function $\lambda x, y.(g(x, y))$, so that, for all $y > 0$, $f(y) = g(y, y)$. Be sure to clearly and explicitly say *which* such g *you* are working with.

Do *not* subsequently neglect handling $f(y)$ in the case $y = 0$ *and* be careful how you handle this case.

[2] f above is *not* a standard text book primitive recursive function. (⌣̈)

C4 **Theory** (25 points)

Let N denote the set of non-negative integers. Fix a standard programming formalism φ for computing all the *one-argument* partial computable functions which map N into N. Code (Gödel) number the φ-programs *onto* N. Let φ_p denote the partial function computed by program (number) p in the φ-system. You may assume with*out* proof that, in the φ-system, Universality, S-m-n, and the Kleene Recursion Theorem (KRT) hold. You may also assume with*out* proof that the following theorem holds.

Padding Theorem

$$(\forall p)(\exists p' \neq p)[\varphi_{p'} = \varphi_p]. \tag{12}$$

Let Φ be a run-time function for φ satisfying the Blum Axioms.

You may not need all these assumptions — just clearly say which ones you are using when.

Let $K \stackrel{\text{def}}{=} \{p \in N \mid \varphi_p(p) \text{ is defined}\}$.

Let $\mathcal{C}$ be any collection of partial computable functions. *Prove* that $K \neq \{x \mid \varphi_x \in \mathcal{C}\}$.

Hint: Apply KRT to obtain e_0 so that $e_0 \in K$ but also so that $(\exists e_1)[\varphi_{e_1} = \varphi_{e_0} \wedge e_1 \notin K]$.
Then: consider cases about whether $\varphi_{e_0} \in \mathcal{C}$.