Comprehensive Solutions:
Automata, Languages, and Mathematical Theory of Computation (60 points) **Autumn 1989**

Problem 1 (20 points).

1a. L_1 consists of all the subsequences of the strings in L. Let M be a finite state automaton accepting L. Construct M_1 from M by adding an ϵ-edge between every pair (s,t) of states of M such that there is an edge from s to t labeled with some symbol of Σ. The language accepted by M_1 is L_1. So L_1 is necessarily regular (and hence, context-free).

1b. We will show that L_2 is not necessarily connext-free (hence not regular). Take $L = (ab)^*(cd)^*$. Let $L_3 = \{a^m c^n b^m d^n : m, n \geq 0\}$. L_3 is not context-free. We will show that $L_2 \cap a^*c^*b^*d^* = L_3$. Since any string in L_2 has equal number of a's and b's, and equal number of c's and d's, it follows that $L_2 \cap a^*c^*b^*d^* \subseteq L_3$. Furthermore, for any $m, n \geq 0$, $(ab)^m(cd)^n \in \mathit{shuffle}(a^m c^n, b^m d^n)$. Hence $L_3 \subseteq L_2 \cap a^*c^*b^*d^*$.
Since context-free languages are closed under intersection with regular sets, it follows that L_2 is not context-free.

Problem 2 (10 points). We will assume that there exists a polynomial-time program Q which solves the decision problem and use it to construct a polynomial time program for printing a Hamiltonian cycle, thus contradicting the assumption.

Let G be a given graph.

If Q when given G as the input returns "no", we are done. If not, use the following procedure to find a Hamiltonian cycle. Assume that the edges of G are enumerated in some fixed order. Initially let $G' = G$. Repeat the following step for every edge e one by one in the order of enumeration:

> If Q, when given $G' - \{e\}$ as the input, says "yes",
> then set G' to $G' - \{e\}$, else leave G' unchanged.

At the end G' will be a subgraph of G composed solely of a Hamiltonian cycle. The time taken by this program equals the product of the number of edges of G by the time taken by Q, and thus is polynomial in the size of G.

Problem 3 (20 points).

3a.

$$\frac{\{\phi\}\, P\, \{\psi\} \qquad \{\psi \wedge \neg E\}\, P\, \{\psi\}}{\{\phi\}\ \textbf{repeat}\ P\ \textbf{until}\ E\ \{\psi \wedge E\}}.$$

3b. Let

$p(x, y, quot, rem)$: $x = y \cdot quot + rem \wedge rem \geq 0$,
$q(x, y, quot, rem, i, z)$: $p(x, y, quot, rem) \wedge z = i \cdot y$,
$r(x, y, quot, rem, i, z)$: $q(x, y, quot, rem, i, z) \wedge rem \geq z$.

The assertions ϕ and ψ of (a) correspond to $r(x, y, quot, rem, i, z)$ and $q(x, y, quot, rem, i, z)$, respectively. All verification conditions are easy to check:

(i) $x \geq 0 \wedge y > 0 \rightarrow p(x, y, 0, x)$,
(ii) $p(x, y, quot, rem) \wedge rem \geq y \rightarrow r(x, y, quot, rem, 1, y)$,
(iii) $r(x, y, quot, rem, i, z) \rightarrow q(x, y, quot + i, rem - z, i + i, z + z)$,
(iv) $q(x, y, quot, rem, i, z) \wedge rem \not< z \rightarrow q(x, y, quot + i, rem - z, i + i, z + z)$,
(v) $q(x, y, quot, rem, i, z) \wedge rem < z \rightarrow p(x, y, quot, rem)$,
(vi) $p(x, y, quot, rem) \wedge rem \not\geq y \rightarrow x = y \cdot quot + rem \wedge 0 \leq rem < y$.

3c. The value of *rem* is a suitable variant function into $(N, <)$. To prove termination, show that:

(i) $rem \geq 0$ is an invariant for both loops (which follows from (b)).

(ii) $z > 0$ is an invariant of the repeat-loop (which requires stronger invariants than (b)).

While (i) guarantees that $rem \in N$, condition (ii) establishes that the repeat-loop reduces *rem* with every execution ($rem > rem - z$). So does the while-loop, because every execution of the while-loop includes at least one execution of the repeat-loop.

Problem 4 (20 points). The set of programs that accept infinitely many inputs is not r.e.

Take an instance of the no-input halting problem: Does program P_1 halt, when started with null input? From P_1, construct a program P_2 that accepts n if P_1 has no computation that halts in at most n steps. If P_1 halts, P_2 accepts finitely many inputs; if no, P_2 accepts everything.

Suppose the set of programs that accept infinitely many inputs were r.e., i.e., the range of a computable function f. To decide whether P_1 halts, for each n, test whether P_1 halts in n steps (then P_1 halts) and whether $P_2 = f(n)$ (then P_1 does not halt) until one is true.

This is an application of the argument of the second form of Rice's theorem.