**UNIVERSITY OF EDINBURGH**

**FACULTY OF SCIENCE AND ENGINEERING**

**COMPUTER SCIENCE THREE**

# COMPUTABILITY AND INTRACTABILITY

**Monday 4 June 2001**

**09:30 to 11:00**

**Examiners:**

**K. Turner (External)**

**G. Plotkin (Chairman)**

## INSTRUCTIONS TO CANDIDATES

**Answer any TWO questions.**

**All questions carry equal weight.**

1. (a) State *Turing's Thesis* and briefly explain its role in the study of computability theory. [*4 marks* ]

   (b) Prove that a Turing machine with $k$ singly infinite tapes each with an independently moving head together with a finite state control can be simulated by the standard Turing machine.

   [**Note:** Give a proof that is convincing without going to the level of detailed instructions.] [*8 marks* ]

   (c) Suppose that we tried to generalize the $k$-tape model by allowing infinitely many tapes and infinitely many independently moving heads together with a finite state control. Is this a reasonable model of computation? Give a brief but clear explanation of your answer. [*4 marks* ]

   (d) Prove carefully that if there is a Turing machine that accepts the language $L \subseteq \Sigma^*$, then there is a Turing machine that accepts the language

$$L' = \{x \in \Sigma^* \mid x \in L \text{ or } x^R \in L\},$$

   where $x^R$ denotes the reversal of the string $x$, i.e., if $x = x_1 x_2 \cdots x_n$ then $x^R = x_n x_{n-1} \cdots x_1$. [*9 marks* ]

2. (a) Explain what is meant by saying that a language $L$ is *recursive*. [*3 marks* ]

(b) Let $L$ be a language over some finite alphabet $\Sigma$. Prove carefully that $L$ is recursive if and only if its complement $\overline{L}$ is recursive. [*6 marks* ]

(c) Recall that the language

$$L_{\text{ne}} = \{\langle M \rangle \mid L(M) \neq \emptyset\},$$

is not recursive where $\langle M \rangle$ represents the encoding of the binary Turing machine $M$. Use this fact to deduce that the language

$$L_{\text{empty}} = \{\langle M \rangle \mid L(M) = \emptyset\},$$

is also not recursive, taking care to justify any claims you make (e.g., $L_{\text{empty}}$ is not all of the complement of $L_{\text{ne}}$). [*4 marks* ]

(d) i. Show that if a language $L$ can be reduced to $\emptyset$ then $L = \emptyset$. [*3 marks* ]

ii. Given two recursively enumerable languages (by means of Turing machines that accept them) it would be useful to be able decide if one can be reduced to the other. Let

$$L_{\text{red}} = \{\langle M_1 \rangle \$ \langle M_2 \rangle \mid \text{there is a reduction from } L(M_1) \text{ to } L(M_2)\}.$$

Prove that $L_{\text{red}}$ is not recursive.
[**Hint:** Use the fact, from part (c) above, that $L_{\text{empty}}$ is not recursive. You may assume this even if you do not attempt part (c).] [*6 marks* ]

iii. Recall that the language $L_{\text{uhalt}}$ is defined by

$$L_{\text{uhalt}} = \{\langle M \rangle \mid M \text{ halts on all inputs } x \in \{\mathtt{0}, \mathtt{1}\}^*\}.$$

Given that $L_{\text{uhalt}}$ can be reduced to $L_{\text{red}}$ is $L_{\text{red}}$ recursively enumerable? (Give a brief justification of your answer.) [*3 marks* ]

3. (a) Define the problem SAT and explain why it is in the class NP. [*4 marks*]

(b) In fact SAT is NP-complete. Explain the meaning of this and explain why it is widely conjectured that there is no polynomial time algorithm for SAT. [*4 marks*]

(c) Suppose we had a polynomial time algorithm for SAT. Explain how we could use it to create a polynomial time algorithm that finds a satisfying assignment for any instance of SAT that happens to be satisfiable (i.e., the algorithm returns NOT SATISFIABLE if the input formula has no satisfying assignment and a satisfying assignment otherwise).

[**Hint:** The assumed algorithm for SAT tells us if a formula is satisfiable and nothing else. You must use this 'black box' as a sub-routine to build the required algorithm for finding satisfying assignments. Thus you may not assume anything about the supposed algorithm other than the fact that it returns the correct answer and it runs in polynomial time.

Recall that a naïve search for possible satisfying assignments runs in exponential time because we have no way to rule out any possibilities as each variable is set to TRUE or FALSE. Use the assumed black box algorithm to focus the search.] [*8 marks*]

(d) Define the QPROG problem to be:

INSTANCE: An integer $m \times n$ matrix $A$, an integer $m$-vector $\mathbf{b}$ and a number of quadratic polynomials in the variables $x_1, x_2, \ldots, x_n$ with integer coefficients.

QUESTION: Does there exist an $n$-vector $\mathbf{x}$ of real numbers such that $A\mathbf{x} \leq \mathbf{b}$ and the quadratic polynomials all evaluate to 0 when $x_i$ is given the value of the $i$th component of $\mathbf{x}$ for $1 \leq i \leq n$? (The vector inequality signifies that each component on the left is to be less than or equal to the corresponding component on the right.)

Note that a quadratic polynomial in the variables $x_1, x_2, \ldots, x_n$ with integer coefficients is an expression such as $x_1^2 - 2x_1x_3 + x_4 + 1$ or $x_2^2 - x_5$.

i. Prove that QPROG is NP-hard. [*7 marks*]

ii. Explain briefly any difficulties we encounter in trying to prove that QPROG is in NP. [*2 marks*]