FOR EXTERNAL EXAMINER (date of this version: 6/3/2022)

UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INTRODUCTION TO THEORETICAL COMPUTER SCIENCE

November 2019

23:50 to 23:59

INSTRUCTIONS TO CANDIDATES

Answer ALL questions from Section A,
and ONE question from Section B.
Section A carries 30 marks, and Section B carries 20 marks.

This is an OPEN BOOK examination.

Year 3 Courses

Convener: ITO-Will-Determine
External Examiners: ITO-Will-Determine

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

FOR EXTERNAL EXAMINER (date of this version: 6/3/2022)

SECTION A

Answer ALL questions in this section.

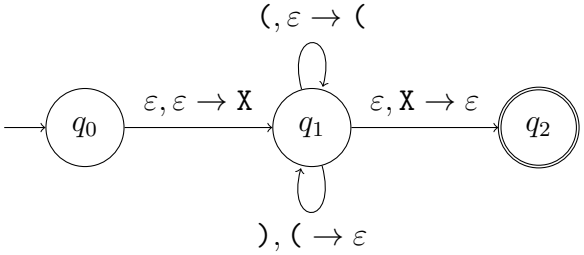
1. (a) Define the language operation $\oplus$, the *exclusive or*, as follows:

$$L_1 \oplus L_2 = \{w \mid w \in L_1 \text{ or } w \in L_2, \text{but not both}\}$$

i. Are the regular languages closed under $\oplus$? Why or why not? [*1 mark*]

ii. Are the context-free languages closed under $\oplus$? Why or why not? [*1 mark*]

(b) Consider this pushdown automaton P where $\Sigma = \{\texttt{(}, \texttt{)}\}$ and $\Gamma = \{\texttt{(}, \texttt{)}, \texttt{X}\}$:



i. Describe $\mathcal{L}(P)$ in simple English. [*2 marks*]

ii. Give a CFG that recognises $\mathcal{L}(P)$. [*2 marks*]

(c) Show that the context-free language $\mathcal{L}(P)$ is *not* regular, using any of the methods discussed in the course (Pumping, Myhill-Nerode, etc.). [*4 marks*]

2. (a) In the course, we proved that there is a universal Register Machine that can simulate any other Register Machine. The proof relied upon the use of a pairing function, and we chose $\langle x, y\rangle_2 \stackrel{\mathsf{def}}{=} 2^x 3^y$. The pairing function was then used to produce a sequence coding function $\langle\rangle$.

   Suppose that we are simulating a machine M with five registers, a program 100 lines long, and that during the course of M's execution, the highest value found in a register is 10.

   Construct, with justification, a rough estimate of the largest number that will occur in a register of the simulating machine. You can present your estimate in any appropriate notation. *[4 marks]*

   (b) An alternative pairing function, which is a bit harder to invert, but much more efficient, is $\langle x, y\rangle_2' \stackrel{\mathsf{def}}{=} \frac{1}{2}(x + y)(x + y + 1) + y$ (the Cantor pairing function). Give, without working, a similar rough estimate if this function is used in the simulation. *[2 marks]*

   (c) Consider an alternative Register Machine architecture, where the DECJZ(i, j) instruction is replaced by two simpler instructions:

   - DEC(i), which decrements R_i or does nothing if it is already zero, and
   - JZ(i, j), which jumps to instruction j if R_i is zero.

   Show that this architecture is equivalent to the standard architecture. *[4 marks]*

3. (a) The λ-term $(\lambda n.\ \lambda f.\ \lambda x.\ f\ (n\ f\ x))\ (\lambda f.\ \lambda x.\ f\ x)$ contains a subterm to which η-reduction can be applied. Identify the subterm and give the reduced term. [*2 marks*]

Let t_n be an infinite sequence of λ-terms where:

$$\begin{array}{lcl} t_0 & = & y\ x\ x \\ t_{n+1} & = & (\lambda x.\ t_n)\ t_0 \end{array}$$

So, for example, $t_1 = (\lambda x.\ y\ x\ x)\ (y\ x\ x)$ which β-reduces to $y\ (y\ x\ x)\ (y\ x\ x)$.

(b) If we measure the time cost of a given λ-term to be the number of β-reductions required, what is the time complexity class of computing the normal form of t_n? Assume a leftmost call-by-name reduction strategy. [*2 marks*]

(c) If we measure the space cost of a given λ-term to be the maximum size of the expression represented as a string over its evaluation, what is the space complexity of computing the normal form of t_n? [*2 marks*]

(d) We know that Turing machines can simulate the evaluation of λ-calculus terms. Results about complexity, however, are only preserved by such simulations if they are *reasonable* (i.e. introduce at most polynomial overhead). Consider the Turing machine that evaluates (again assuming leftmost call-by-name) a λ-term represented as a string on its tape. Is this simulation *reasonable*? Why or why not? [*4 marks*]

Hint: Consider the sequence t_n.

SECTION B

Answer ONE question in this section. If both questions are attempted, ONLY question 4 will be marked.

4. Consider the following problem, DFAInter: *Given a finite set of DFAs, is there a word w which is accepted by all of them?*

   This problem is known in general to be PSPACE-complete, but we shall examine a restricted form of the problem where the languages of all input DFAs are *finite*, called DFAInterF. This means that length of our common word w is bounded above by n, the maximum number of states in the input DFAs.

   (a) Show that DFAInterF is in NP. [*3 marks*]

   Hint: Use the fact that $|w| \leq n$

   (b) Recall that an instance of 3-SAT is a boolean formula φ over a set $V = \{v_1, \ldots, v_n\}$ of variables. The formula is in 3CNF, i.e. φ is a conjunction $\bigwedge_{1 \leq i \leq k} C_i$ of *clauses*, where each clause C_i is a disjunction $(\ell_1 \vee \ell_2 \vee \ell_3)$ of three *literals*, each of which is either of a variable v_m or a negated variable $\neg v_m$.

   Construct a reduction to DFAInterF from 3-SAT. That is, given an instance of 3-SAT, construct an instance of DFAInterF. [*5 marks*]

   Hint: For each clause in the given 3CNF formula, construct a DFA that accepts any satisfying truth assignments for that clause (encoded as a binary string).

   (c) Show that your reduction is polynomial time, and deduce that DFAInterF is NP-complete. [*4 marks*]

   Now consider the problem KPow: *Given a DFA D and a positive integer k, is there a non-empty word w such that D accepts k repetitions of w (i.e. w^k)?*

   (d) Show that this problem is NP-hard, by constructing a polynomial time reduction from DFAInterF to KPow. [*8 marks*]

   Hint: Given DFAs $A_1, \ldots A_k$, consider the language consisting of strings (where `#` is not in the alphabet of any input DFA)

   $$\mathcal{L}(A_1)\#\cdots\mathcal{L}(A_k)\#$$

5. The lambda-calculus is based on variables, lambda-abstraction and application. These can be a bit tricky to understand and manipulate.

   (a) Demonstrate the above by tracing how

   $$(\lambda x.\lambda y.xy)(\lambda x.y)w$$

   β-reduces under a call-by-name strategy, identifying the reduced term at each stage, and commenting on steps that require particular attention. [*4 marks*]

   Combinator calculus is a way to remove variable and lambda-abstraction. It gives names to three special λ-terms, and gives them *derived β rules* (which follow directly from their definitions), as follows:

combinator	derived β rule
$\mathsf{I} \stackrel{\mathsf{def}}{=} \lambda x.x$	$\mathsf{I}U \stackrel{\beta}{\to} U$
$\mathsf{K} \stackrel{\mathsf{def}}{=} \lambda x.\lambda y.x$	$\mathsf{K}UV \stackrel{\beta}{\to} U$
$\mathsf{S} \stackrel{\mathsf{def}}{=} \lambda x.\lambda y.\lambda z.xz(yz)$	$\mathsf{S}UVW \stackrel{\beta}{\to} UW(VW)$

   Here U, V, W are arbitrary terms. Remember that application associates to the left, so that $\mathsf{S}UVW$ means $((\mathsf{S}U)V)W$

   A SKI-*term* is a term written using only $\mathsf{I}, \mathsf{S}, \mathsf{K}$ and application (i.e. with no explicit λs). A SKI-term may β-reduce to another such term using the derived rules. For example, $\mathsf{SKSK} \stackrel{\beta}{\to} \mathsf{KK}(\mathsf{SK}) \stackrel{\beta}{\to} \mathsf{K}$.

   We say that two SKI-terms U and V are *β-equal*, $U \stackrel{\beta}{=} V$, if they can be converted into each other by forward and backward uses of the combinator $\stackrel{\beta}{\to}$ rules.

   We say that U and V are *(extensionally) equal*, $U = V$, if either $U \stackrel{\beta}{=} V$, or for some n, for any terms $X_1, \ldots, X_n$, $UX_1 \ldots X_n = VX_1 \ldots X_n$. That is, U and V need not be directly interconvertible, but when applied to enough arguments, they are.

   It can be shown that if $U \stackrel{\beta}{=} V$, then there is a term W such that $U \stackrel{\beta}{\to} \ldots \stackrel{\beta}{\to} W$ and $V \stackrel{\beta}{\to} \ldots \stackrel{\beta}{\to} W$ – the Church–Rosser theorem.

   (b) Show that $\mathsf{SKK} = \mathsf{I}$, but $\mathsf{SKK} \stackrel{\beta}{\neq} \mathsf{I}$. [*4 marks*]

QUESTION CONTINUES ON NEXT PAGE

Since combinator calculus is equivalent to lambda-calculus, it is equivalent to register/Turing machines. We can show this directly by encoding register machines into combinators. The following additional combinator is useful:

$$\mathsf{B} \stackrel{\mathsf{def}}{=} \lambda x.\lambda y.\lambda z.x(yz) \qquad \mathsf{B}UVW \stackrel{\beta}{\to} U(VW)$$

One way of encoding the natural numbers into combinator calculus, where $\ulcorner n \urcorner$ denotes the coding of n, is:

$$\ulcorner 0 \urcorner \stackrel{\mathsf{def}}{=} \mathsf{KI}$$
$$\ulcorner 1 \urcorner \stackrel{\mathsf{def}}{=} \mathsf{I}$$
$$\ulcorner n+1 \urcorner \stackrel{\mathsf{def}}{=} \mathsf{SB}(\ulcorner n \urcorner) \text{ for } n \geq 1$$

(c) In the preceding definition SB appears to be representing the successor (add 1) function, but we first use it in $\ulcorner 2 \urcorner$. Show that $\mathsf{SB}(\ulcorner 0 \urcorner) = \ulcorner 1 \urcorner$. (Hint: you will need to give the terms two arguments.) [*2 marks*]

(d) Show, by induction on n, that for arbitrary terms X, Y, we have $\ulcorner n \urcorner XY = X^n(Y)$, where $X^n(Y)$ means $X(X(X \ldots (X(Y)) \ldots))$ with n X's. [*4 marks*]

Similarly, combinators can express decrement, zero-test, addition, multiplication, and other arithmetic functions. The fixed point Y combinator from lectures can also be expressed in terms of $\mathsf{S}, \mathsf{K}, \mathsf{I}$, and allows the definition of recursive functions or loops.

(e) Outline how you would simulate register machines in combinatory calculus, assuming that the above-mentioned constructions are available. Assume also that the simulation is used for providing a reduction from the halting problem for register machines to a problem in combinatory calculus. Explain what the target problem is, and how the reduction preserves membership of the source and target queries.

Your answer should discuss (briefly) how to represent machines as combinator terms, how to simulate execution of the machine, and how to simulte halting of the machine. [*6 marks*]