UNIVERSITY OF BRISTOL

January Examination Period

FACULTY OF ENGINEERING

Second Year Examination for the Degrees
of
Bachelor of Science
Master of Engineering

COMS20007J
Programming Languages and Computation

TIME ALLOWED:
3 Hours

Answers to COMS20007J: Programming Languages and Computation

Intended Learning Outcomes:

Q1. This question is about regular languages.

(a) Consider each of the following regular expressions over the alphabet $\Sigma = \{a, b, c\}$:

1. $((b+c)^*a(b+c)^*a(b+c)^*)^*$
2. $\Sigma^* abb \Sigma^*$
3. $\Sigma^*(abb + baa)\Sigma^*$
4. $b\Sigma^* b$
5. $(\Sigma\Sigma)^*$
6. $(b+c)^*$
7. $abb\Sigma^*$

Match each of the following descriptions of languages to the regular expression above that denotes it:

i. The language of all words that start and end with b
ii. The language of all words that contain abb as a substring
iii. The language of all words that start with abb.
iv. The language of all words that do not contain a.
v. The language of all even length words.
vi. The language of all words containing an even number of a.
vii. The language of all words that either contain abb or bba

[7 marks]

Solution:

i. 4

ii. 2

iii. 7

iv. 6

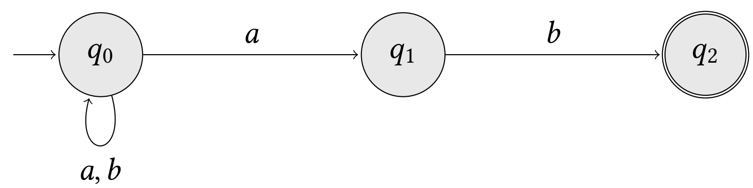
v. 5

vi. 1

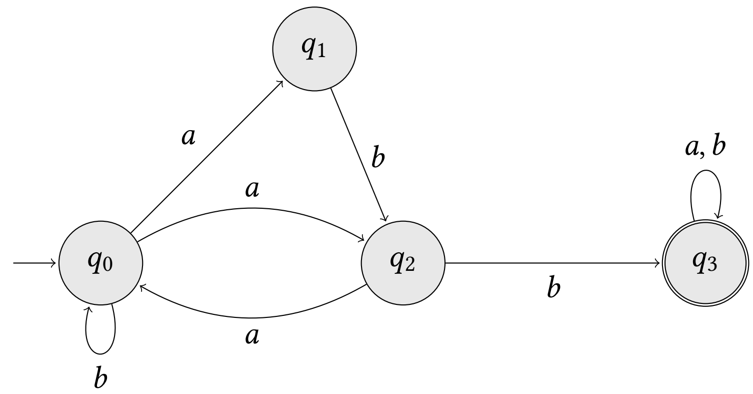
vii. 3

(b) For each of the following automata, give a word that is accepted by the automaton *and* a *deterministic* automaton that recognises the same language.

i.



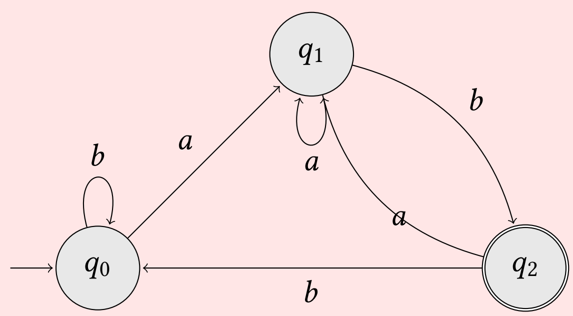
ii.



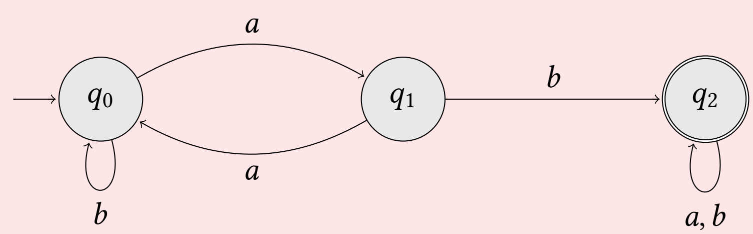
[10 marks]

Solution: Lots of answers are possible for the accepted words, one mark for each of i and ii. For the determinisation, it is also possible to use the subset construction, but the automata may be much larger (full marks are given, though) than the following:

i.



ii.



(c) As in the Week 3 Problem Sheet, Question 9, we shall encode pairs of natural numbers by sequences of vectors of bits, with least significant bit first (left-most). Let Σ be the following set of binary vectors:

$$\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}0\\1\end{pmatrix}, \begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}1\\1\end{pmatrix}$$

We use each word w over alphabet Σ to encode a pair of natural numbers, written

(cont.)

$[\![w]\!]$, which is defined by the following recursive function:

$$\begin{aligned} [\![\epsilon]\!] &= (0,0) \\ [\![\begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \cdot w]\!] &= (2*m+b_1,\ 2*n+b_2) \\ & \qquad \textit{where } (m,\ n) = [\![w]\!] \end{aligned}$$

For example:

$$[\![\begin{pmatrix}0\\1\end{pmatrix}\begin{pmatrix}0\\0\end{pmatrix}\begin{pmatrix}1\\1\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix}]\!] = (4,\ 13)$$

$$[\![\begin{pmatrix}0\\1\end{pmatrix}\begin{pmatrix}1\\1\end{pmatrix}\begin{pmatrix}0\\0\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix}]\!] = (26,\ 3)$$

$$[\![\begin{pmatrix}1\\1\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix}\begin{pmatrix}0\\0\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix}]\!] = (1,\ 23)$$

For each of the following, give a finite state automaton that recognises the (encoding of the) language and has at most 3 states. For this problem, your automata should *not* accept the empty word.

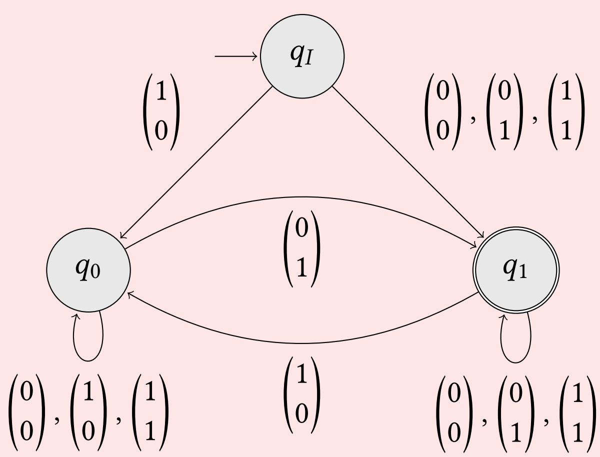
i. $\{w \mid [\![w]\!] = (n,m) \wedge n \leq m\}$

ii. $\{w \mid [\![w]\!] = (n,m) \wedge m = 2*m\}$

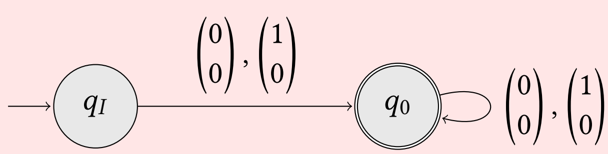
[8 marks]

Solution:

i.



ii. In this part, you need to realise that $m = 2*m$ implies that m must be 0.



(d) Let Σ be an alphabet and suppose $M_1 = (Q_1, \Sigma, \delta_1, p_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, p_2, F_2)$ are deterministic finite state automata. Describe a finite state automaton that recognises the following language:

$$\{a_1b_1a_2b_2\cdots a_nb_n \mid n \geq 0 \wedge a_1a_2\cdots a_n \in L(M_1) \wedge b_1b_2\cdots b_n \in L(M_2)\}$$

(i.e. those words of even length where concatenating the letters at even numbered positions yields a word accepted by M_1 and concatenating the letters at odd-numbered positions yields a word accepted by M_2). *[6 marks]*

Solution: We construct a deterministic automaton $M = (Q, \Sigma, \delta, q_0, F)$ where:

- $Q = \{1, 2\} \times Q_1 \times Q_2$
- $q_0 = (1, p_1, p_2)$
- $F = \{1\} \times F_1 \times F_2$
- and δ is given by the following:

$$\begin{aligned} \delta((1, q_1, q_2),\ a) &= (2, \delta_1(q_1, a), q_2) \\ \delta((2, q_1, q_2),\ a) &= (1, q_1, \delta_2(q_2, a)) \end{aligned}$$

(e) Let $\#_0(w)$ be the number of '0' letters in word w and $\#_1(w)$ be the number of '1' letters in word w. For example, $\#_0(01101) = 2$ and $\#_1(01101) = 3$.

Prove that the language $\{w \in \{0, 1\}^* \mid \#_0(w) = \#_1(w)\}$ is not regular.

[6 marks]

Solution: Call this language L. Suppose L is regular and let p be the pumping length given by the Pumping Lemma. Then consider the word w:

$$0^p1^p$$

This word has length $2p$ and so it follows by the Pumping Lemma that there is a decomposition $w = xyz$ with $|y| > 0$ and $|xy| \leq p$. It follows that, necessarily:

1. $xy = 0^k$ for some $k \leq p$,
2. $y = 0^\ell$ for some $0 < \ell \leq k$
3. $z = 0^{p-k}1^p$

By pumping, it follows that, e.g. $xy^2z \in L$. However, $xy^2z = 0^k0^\ell0^{p-k}1^p$ and so $xy^2z \notin L$, since:

$$\#_0(xy^2z) = \ell + p \neq p = \#_1(xy^2z)$$

Contradiction.

(cont.)

(f) Given two languages A and B over a common alphabet Σ, define:

$$A \triangleleft B := \{w \in \Sigma^* \mid \exists v.\, wv \in A \wedge v \in B\}$$

Suppose A is regular. Show that there is a finite automaton recognising $A \triangleleft B$ (irrespective of whether or not B is regular).

[8 marks]

Solution: This question is of very high difficulty.

Since A is regular, it is recognised by a some finite automaton M. Let $M = (Q, \Sigma, \delta, q_0, F)$. Define the indexed family of automata $(M_q)_{q \in Q}$ by $M_q = (Q, \Sigma, \delta, q, F)$. Then the automaton $(Q, \Sigma, \delta, q_0, F')$ recognises $A \triangleleft B$, where $q \in F'$ iff $L(M_q) \cap B \neq \emptyset$. Note, in general it is not possible to carry out this construction effectively.

Q2. This question is about the While language.

(a) For each of the following, indicate whether it is a syntactically valid Boolean expression in the While language. You may assume that x, y and z are variables.

i. `! true`
ii. `(!x) = true`
iii. `true && 1 = 1`
iv. `true && (false || !x=3)`
v. `x < y < z`

[5 marks]

Solution:

i. yes
ii. no
iii. yes
iv. yes
v. no

(b) For each of the following arithmetic expressions a, give the number it evaluates to $[\![a]\!]^{\mathcal{A}}([x \mapsto 3, y \mapsto 5])$ when evaluated in state $[x \mapsto 3, y \mapsto 5]$:

i. `23`
ii. `x + x`
iii. `(3 - x) + z`
iv. `5 * (x + y)`
v. `1 + y * z`

[5 marks]

Solution:

i. 23

ii. 6

iii. 0

iv. 40

v. 1

(c) i. Consider the following While program.

```
n := 1
r := 0
while (n <= x) {
  n := 2 * n
  r := r + 1
}
```

Describe the 7th configuration in its execution trace starting in initial state $[\mathtt{x} \mapsto 4]$.

ii. Write a program in While that always terminates in a state where variable z has value x^y, where x (resp. y) is the initial value of variable x (resp. y). You may assume that we only run this program in initial states where y is non-negative.

iii. Consider the nth Fibonacci number $\mathsf{fib}(n)$, defined inductively as follows.

$$\mathsf{fib}(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ \mathsf{fib}(n-2) + \mathsf{fib}(n-1) & \text{otherwise} \end{cases}$$

Write a While program that computes the nth Fibonacci number for any given $n \in \mathbb{N}$. Your program should read the value of n from variable n in its initial state (you can assume it is always non-negative), and write the value of $\mathsf{fib}(n)$ into variable result in its final state. You can use as many additional variables as desired.

[15 marks]

Solution:

i. The 7th configuration is that in which the second iteration of the loop evaluates its first instruction.

$$\left\langle \begin{array}{l} \mathtt{n\ :=\ 2\ *\ n} \\ \mathtt{r\ :=\ r\ +\ 1} \\ \mathtt{while\ (n\ <=\ x)\ \{} \\ \quad \mathtt{n\ :=\ 2\ *\ n} \\ \quad \mathtt{r\ :=\ r\ +\ 1} \\ \mathtt{\}} \end{array}, \begin{bmatrix} \mathtt{x} \mapsto 4 \\ \mathtt{n} \mapsto 2 \\ \mathtt{r} \mapsto 1 \end{bmatrix} \right\rangle$$

Note that the program is a really horrendous way of computing (the integer floor of) a base 2 logarithm. Which is completely irrelevant to the question.

ii. The following program works.

```
z := 1
while !y = 0 {
  z := z * x
  y := y - 1
}
```

iii. The following uses the concrete syntax given in lectures. Other solutions are likely.

- The prologue here could be cleaned up.
- Another strategy is to keep the loop one-step behind, and have an epilogue that computes a final sum. This removes the need to special case $n = 1$.

Minor syntax issues should not be penalized as long as the program remains unambiguous. Checking functionality could be done through the interpreters for the more convoluted proposals.

```
if n <= 0 then {
  result := 0
} else { }
if n = 1 then {
  result := 1
} else { }
fib2 := 0
fib1 := 1
i := 2
while (i <= n) {
  result := fib2 + fib1
  fib2 := fib1
  fib1 := result
  i := i + 1
}
```

(d) This question is about compiling arithmetic expressions to machine code. The abstract syntax for the machine language is slightly simpler than that for the While language, although it retains a recursive (inductive) definition since its control-flow is still structured.

Abstract machine instructions, typically C, are either:

- `PUSH` v whenever $v \in \mathbb{Z}$;
- `LOAD x` whenever `x` is a valid While variable identifier;
- `ADD`, `SUB`, or `MUL`

The set of abstract machine programs P, is the smallest set such that:

- ϵ, the empty program, is in P.
- $C\,;\,\pi$, the program whose first instruction is C and after that is program π, is in P whenever $\pi \in P$.

We will use π to stand for an arbitrary machine program.

The small-step semantics of the machine is specified over configurations $\langle \pi, s, \sigma \rangle$ composed of a machine language program π, a state σ (mapping variable names to values in $\mathbb{Z}$), and a stack of integer values, the top of which serves as working memory both for arithmetic operators and control-flow statements.

We use Haskell list notations for stacks (this means their top is denoted to the left), using a lowercase letter s to denote an abstract stack, and [] to denote an empty stack.

$$\begin{array}{rcl}
\langle \texttt{PUSH}\ v;\ \pi, s, \sigma \rangle & \rightarrow & \langle \pi, v : s, \sigma \rangle \\
\langle \texttt{LOAD x};\ \pi, s, \sigma \rangle & \rightarrow & \langle \pi, \sigma(\texttt{x}) : s, \sigma \rangle \\
\langle \texttt{ADD};\ \pi, i_2 : i_1 : s, \sigma \rangle & \rightarrow & \langle \pi, (i_1 + i_2) : s, \sigma \rangle \text{ when } i_1, i_2 \in \mathbb{Z} \\
\langle \texttt{SUB};\ \pi, i_2 : i_1 : s, \sigma \rangle & \rightarrow & \langle \pi, (i_1 - i_2) : s, \sigma \rangle \text{ when } i_1, i_2 \in \mathbb{Z} \\
\langle \texttt{MUL};\ \pi, i_2 : i_1 : s, \sigma \rangle & \rightarrow & \langle \pi, (i_1 * i_2) : s, \sigma \rangle \text{ when } i_1, i_2 \in \mathbb{Z}
\end{array}$$

Figure 1: Semantics for the machine's arithmetic instructions

i. Give the complete trace of the following program configuration:

$$\langle \texttt{PUSH 3};\ \texttt{LOAD x};\ \texttt{ADD};\ \epsilon,\ [],\ [x \mapsto 2] \rangle$$

ii. Construct a machine language program π such that:

$$\langle \pi,\ [],\ \sigma \rangle \rightarrow^* \langle \epsilon,\ [[\texttt{1 + (x * y)}]]^{\mathcal{A}}(\sigma) : [],\ \sigma \rangle$$

In other words, executing the machine language program π starting from an empty stack and in any state σ, yields a stack with one element, which is exactly the interpretation of $1 + (x * y)$ under σ.

iii. Define a function $\mathcal{C}$ from While arithmetic expressions to machine programs in such a way that, for all arithmetic expressions $a \in \mathcal{A}$:

$$\langle \mathcal{C}(a),\ [],\ \sigma \rangle \rightarrow^* \langle \epsilon,\ [[a]]^{\mathcal{A}}(\sigma) : [],\ \sigma \rangle$$

In other words, executing the machine language program $\mathcal{C}(a)$ starting from an empty stack and in any state σ, yields a stack with one element, which is exactly the interpretation $[[a]]^{\mathcal{A}}(\sigma)$ of a under σ.

(cont.)

You may find the following machine program concatenation operator $\oplus$ useful when defining this function. For all instructions $C_1, C_2, \ldots, C_n$ and $C'_1, C'_2, \ldots, C'_m$ it satisfies:

$$(C_1;\ C_2;\ \cdots;\ C_n;\ \epsilon)\oplus(C'_1;\ C'_2;\ \cdots;\ C'_m;\ \epsilon) = (C_1;\ C_2;\ \cdots;\ C_n;\ C'_1;\ C'_2;\ \cdots;\ C'_m;\ \epsilon)$$

[10 marks]

Solution:

i.

$$\begin{aligned} &\langle \mathtt{PUSH}\ 3; \mathtt{LOAD}\ \ \mathtt{x}; \mathtt{ADD}; \epsilon,\ [],\ [x \mapsto 2]\rangle \\ \longrightarrow\ &\langle \mathtt{LOAD}\ \ \mathtt{x}; \mathtt{ADD}; \epsilon,\ 3 : [],\ [x \mapsto 2]\rangle \\ \longrightarrow\ &\langle \mathtt{ADD}; \epsilon,\ 2 : 3 : [],\ [x \mapsto 2]\rangle \\ \longrightarrow\ &\langle \epsilon,\ 5 : [],\ [x \mapsto 2]\rangle \end{aligned}$$

ii. `PUSH 1; LOAD x; LOAD y; MUL; ADD;`

iii.

$$\begin{aligned} \mathcal{C}[[i]] &= \mathtt{PUSH}\ i;\ \epsilon \\ \mathcal{C}[[\mathtt{x}]] &= \mathtt{LOAD}\ \ \mathtt{x};\ \epsilon \\ \mathcal{C}[[e_1 + e_2]] &= \mathcal{C}[[e_1]] \oplus \mathcal{C}[[e_2]] \oplus (\mathtt{ADD};\ \epsilon) \\ \mathcal{C}[[e_1 - e_2]] &= \mathcal{C}[[e_1]] \oplus \mathcal{C}[[e_2]] \oplus (\mathtt{SUB};\ \epsilon) \\ \mathcal{C}[[e_1 * e_2]] &= \mathcal{C}[[e_1]] \oplus \mathcal{C}[[e_2]] \oplus (\mathtt{MUL};\ \epsilon) \end{aligned}$$

Q3. This question is about computability.

(a) A perfect square is a number which is of the form n^2 for some $n \in \mathbb{N}$. Show that the set

$$U = \{n \in \mathbb{N} \mid\ n \text{ is a perfect square } \}$$

is decidable.

[2 marks]

Solution: Any program that tries all possible integer square roots works. For example, if we are computing wrt `n`:

```
i := 1;
while (i * i < n) do { i := i + 1 }
if (i * i = n) then { n := 1 } else { n := 0 }
```

(b) Let $f : A \longrightarrow B$ and $g : B \longrightarrow C$. Show that if $g \circ f : A \longrightarrow C$ is injective, then so is f.

[2 marks]

Solution: Suppose $f(a_1) = f(a_2)$. Then $g(f(a_1)) = g(f(a_2))$, which is to say that $(g \circ f)(a_1) = (g \circ f)(a_2)$. As $g \circ f$ is injective, it follows that $a_1 = a_2$, which is what we wanted to prove.

(c) Let us say that a state σ is 2022-*bounded* just if, for all variables x, $-2022 \leq \sigma(x) \leq 2022$.

One of the following two predicates P and Q is semi-decidable and the other is not. Determine which one is semi-decidable and justify your answer.

$$P = \{\ulcorner S \urcorner \mid \text{for all 2022-bounded } \sigma,\ S \text{ terminates when started in } \sigma\}$$
$$Q = \{\ulcorner S \urcorner \mid \text{there is 2022-bounded } \sigma \text{ and } S \text{ does not terminate when started in } \sigma\}$$

[8 marks]

Solution: The predicate P can be semi-decided by the following algorithm. Termination of the algorithm rests on the fact, for a given program S, only finitely many variables can occur in S and the behaviour of S only depends upon those variables.

Given input S, do as follows:

1. Determine the set X of variables that occur in S by parsing S.
2. Construct a list of the finitely many 2022-bounded states σ for which $\sigma(x) = 0$ whenever $x \notin X$.
3. For each of the states σ in the list: interpret S starting from state σ.
4. Return 1.

By the Church-Turing thesis, there is a While program that implements (the reflection of) this algorithm and thus P is semi-decidable.

(d) Show that the following predicate is undecidable:

$$P = \{\langle \ulcorner S_1 \urcorner, \ulcorner S_2 \urcorner \rangle \mid \text{for all } n \in \mathbb{N}\text{: } [\![S_1]\!]_x(n) \simeq 1 \text{ iff } [\![S_2]\!]_x(n) \simeq k \text{ where } k \neq 1\ \}$$

[8 marks]

Solution: We construct a reduction $f : \text{HALT} \lesssim P$. If P were decidable, then we could also decide the Halting Problem for While programs, which is impossible since this problem is known to be undecidable.

We define a code transformation $F : \textbf{Stmt} \times \mathbb{N} \to \textbf{Stmt} \times \textbf{Stmt}$ by

$$F(\mathtt{D}, n) = (\mathtt{D;\ x\ :=\ 1,\ x\ :=\ 0})$$

(cont.)

:19

We argue that this constitutes a reduction. Suppose $F(D,n) = (S,T)$. Recalling that by convention all our programs are assumed to compute wrt `x`, we see that D halts on input n iff $[\![S]\!]_x(m) \simeq 1$ for all $m \in \mathbb{N}$. We have that $[\![T]\!]_x(n) \simeq 0$ for all $n \in \mathbb{N}$, and clearly $0 \neq 1$. Hence:

- If D halts on n then $\langle \ulcorner S \urcorner, \ulcorner T \urcorner \rangle \in P$ since, for all m:

$$[\![S]\!]_x(m) \simeq 1 \quad \text{iff} \quad [\![T]\!]_x(m) \simeq 0$$

- but otherwise we have $\langle \ulcorner S \urcorner, \ulcorner T \urcorner \rangle \notin P$ since there is an m for which both:

$$[\![S]\!]_x(m) \not\simeq 1 \quad \textit{and} \quad [\![T]\!]_x(m) \simeq 0$$

  In fact, our construction ensures that this is true for every m!

The reflection of this transformation in $\mathbb{N} \to \mathbb{N}$ can be computed by the following algorithm. On input $m \in \mathbb{N}$:

1. Decode m as $\langle \ulcorner \mathtt{D} \urcorner, n \rangle$ to obtain `D` and n.
2. Construct the program $S_{\mathrm{D},n}$ as:

```
D; x := 1
```

3. Return $\langle \ulcorner S_{\mathrm{D},n} \urcorner, \ulcorner \mathtt{x:=1} \urcorner \rangle$

END OF PAPER