UNIVERSITY OF BRISTOL

Practice 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. $\Sigma^*(abb \mid baa)\Sigma^*$
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

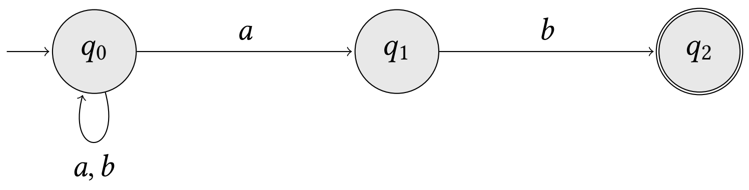
v. 5

vi. 1

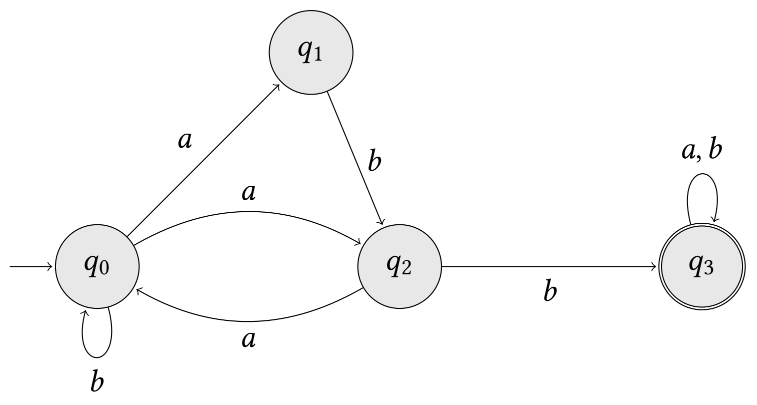
vii. 3

(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.



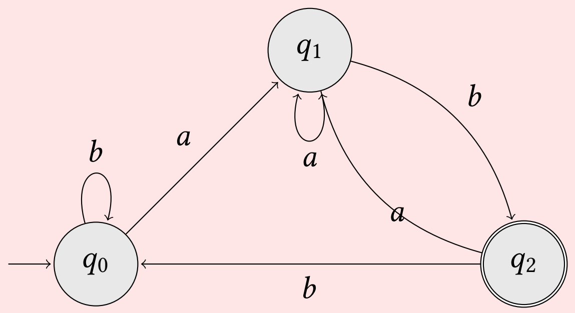
ii.



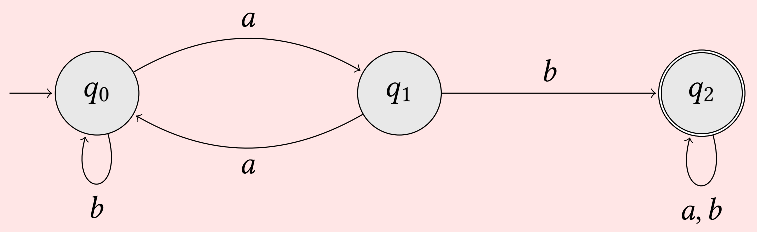
[10 marks]

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 Problem Sheet 2, Question 7, 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.

$$\begin{tikzcd}
{} \arrow[r] & q_I \arrow[dl, "\binom{0}{1}"'] \arrow[dr, "{\binom{0}{0}, \binom{0}{1}, \binom{1}{1}}"] & \\
q_0 \arrow[rr, bend left, "\binom{1}{0}"] \arrow[loop below, "{\binom{0}{0}, \binom{0}{1}, \binom{1}{1}}"] & & q_1 \arrow[ll, bend left, "\binom{0}{1}"] \arrow[loop below, "{\binom{0}{0}, \binom{0}{1}, \binom{1}{1}}"]
\end{tikzcd}$$

ii.

$$\begin{tikzcd}
{} \arrow[r] & q_I \arrow[r, "{\binom{0}{0}, \binom{1}{0}}"] & q_0 \arrow[loop right, "{\binom{0}{0}, \binom{1}{0}}"]
\end{tikzcd}$$

(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 finite state automata. Describe a finite state automaton that recognises the following language:

$$\{a_1 b_1 a_2 b_2 \cdots a_n b_n \mid n \geq 0 \wedge a_1 a_2 \cdots a_n \in L(M_1) \wedge b_1 b_2 \cdots b_n \in L(M_2)\}$$

(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 an 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:

$$\begin{array}{rcl} \delta(1, q_1, q_2) & = & \{(2, q_1', q_2) \mid q_1' \in \delta_1(q_1)\} \\ \delta(2, q_1, q_2) & = & \{(1, q_1, q_2') \mid q_2' \in \delta_2(q_2)\} \end{array}$$

(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:

$$0^p 1^p$$

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, $xy^2z = 0^k 0^\ell 0^{p-k} 1^p$ 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: 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) 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 $[\mathrm{x} \mapsto 4]$.

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 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 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 condition.

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

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
}
```

(cont.)

(b) This question part is about the Abstract Machine's bytecode language and about translating from While to the Abstract Machine.

Consider the program c_{Q2b} shown in Figure 1. It is written in the Abstract Machine's bytecode language.

```
PUSH 0; STORE q;
LOOP(LOAD a; LOAD b; LE;
    , PUSH 1; LOAD q; ADD; STORE q;
      LOAD b; LOAD a; SUB; STORE a;);
FETCH a; STORE r;
```

Figure 1: The Abstract Machine program c_{Q2b}.

For $m, n \in \mathbb{Z}$, define the initial state $s(m,n)$ as $s(m,n) = \begin{bmatrix} \text{a}\mapsto m; \\ \text{b}\mapsto n \end{bmatrix}$.

i. What are the values of variables q and r in the final state of the execution of c_{Q2b} in $s(5,4)$ (if it exists)?

ii. What are the values of variables q and r in the final state of the execution of c_{Q2b} in $s(100,-10)$ (if it exists)?

iii. Under what conditions on the values of a and b does the adversary terminate when run in state $s(a,b)$?

iv. More generally, given a and b such that $0 \leq a$ and $0 < b$, what is the value of variables q and r in the final state of the execution of c_{Q2b}?

v. Euclid's algorithm computes the greatest common divisor of two positive integers a and b. We define it below as a function gcd(a,b), where $b \mod a$ is the remainder in the Euclidean division of b by a.

$$\text{gcd}(a,b) = \begin{cases} b & \text{if } a = 0 \\ \text{gcd}(b,a) & \text{if } b < a \\ \text{gcd}(b \mod a, a) & \text{otherwise} \end{cases}$$

Produce a program in the Abstract Machine's bytecode language which implements Euclid's algorithm. You may assume that inputs are given in variables a and b, and that your program will only be run in states that give them positive values. Place the output in variable result.

[15 marks]

Solution:

i. (1) q $\mapsto 2$; r $\mapsto 1$

ii. (1) Execution does not terminate.

iii. (3) The program does not terminate when $b \leq 0$ and $b \leq a$.

iv. (5) The program computes the Euclidian division of $s(\mathrm{a})$ by $s(\mathrm{b})$, leaving the quotient in variable q and the remainder in variable r so that $s(\mathrm{a}) = s(\mathrm{b}) \times s'(\mathrm{q}) + s'(\mathrm{r})$ and $0 \leq s'(\mathrm{r}) < s(\mathrm{b})$.

v. (5) Don't try to understand the program, interpret it instead (using the solution to lab 4). Be forgiving of minor syntax issues; although I expect anyone who actually answers this will have produced their answer by writing an Imp program and compiling it (which is the way I expect them to solve it), transcription might still be an issue. Mine (below) is generated from a memory-optimized Imp programit could be optimized further by using result instead of r for intermediate remainders.

Because it's an open book exam, I expect some will also provide subtraction-based implementations, since Wikipedia proposes it, and not all students will connect this to the previous question (about division by repeated subtraction), or indeed will have solved the previous question.

```
LOOP(PUSH 0; LOAD a; EQ; NOT;
   , LOAD b; LOAD a; LE; NOT;
     IF(LOAD b; LOAD a; ADD; STORE a;
        LOAD b; LOAD a; SUB; STORE b;
        LOAD b; LOAD a; SUB; STORE a;
      , LOAD b; STORE r;
        LOOP(LOAD r; LOAD a; LE;
            , LOAD a; LOAD r; SUB; STORE r;
             );
        LOAD a; STORE b; LOAD r; STORE a;
       );
    );
LOAD b; STORE result;
```

(c) This question part is a more advanced compilation problem. You probably should not attempt it until you are convinced you cannot do more on the rest of the exam.

We consider a knock-off Abstract Machine (the Knock-Off Machine, below) that can only hold two elements on its stack. We give (part of) its structural operational semantics as a transition relation $\triangleright_{\mathsf{KO}}$ (Figure 2).

Define and argue correct a translation function $\mathcal{CA}_{\mathsf{KO}}[\![\cdot]\!]$ from Imp's *arithmetic expressions only* to the Abstract Machine's bytecode language.

When the source expression a has defined semantics $\mathcal{A}[\![a]\!](s)$ in a given state s, executing the translated program $\mathcal{CA}_{\mathsf{KO}}[\![a]\!]$ with the empty evaluation stack in state s should yield a configuration with an empty program, a stack containing exactly one element whose value is $\mathcal{A}[\![a]\!](s)$, and a state that must be larger than s in the sense that it is defined and equal to s wherever s is defined, and may also

$$
\begin{array}{lll}
\langle \texttt{PUSH}\ n; c, e, s\rangle \triangleright_{\mathsf{KO}} \langle c, n : e, s\rangle & & \text{if } |e| \leq 1 \\
\langle \texttt{ADD}; c, z_1 : z_2, s\rangle \triangleright_{\mathsf{KO}} \langle c, z_1 + z_2, s\rangle & & \text{if } z_1, z_2 \in \mathbb{Z} \\
\langle \texttt{MUL}; c, z_1 : z_2, s\rangle \triangleright_{\mathsf{KO}} \langle c, z_1 * z_2, s\rangle & & \text{if } z_1, z_2 \in \mathbb{Z} \\
\langle \texttt{SUB}; c, z_1 : z_2, s\rangle \triangleright_{\mathsf{KO}} \langle c, z_1 - z_2, s\rangle & & \text{if } z_1, z_2 \in \mathbb{Z} \\
\langle \texttt{FETCH}\ \mathsf{x}; c, e, s\rangle \triangleright_{\mathsf{KO}} \langle c, s(\mathsf{x}) : e, s\rangle & & \text{if } |e| \leq 1 \\
\langle \texttt{STORE}\ \mathsf{x}; c, z : e, s\rangle \triangleright_{\mathsf{KO}} \langle c, e, s[\mathsf{x} \mapsto z]\rangle & & \text{if } z \in \mathbb{Z}
\end{array}
$$

Figure 2: Semantics for the Knock-Off Machine's arithmetic instructions

give values to variables not defined in s.

Your translation function may rely on an environment and does not need to be optimal. In fact, simpler is better.

[5 marks]

Solution: 2 points for the idea. 2 points for some mostly correct realization. 1 point for a reasonable justification. (Exercise judgment.)

The central idea is to offload overflow stack elements to the state where necessary. In the below, we only use the stack for immediate operations (storing values as soon as possible, and fetching values as late as possible); this is likely not optimal, but optimality was not asked for. Other concrete solutions are possible.

$$
\begin{array}{rcl}
\mathcal{CA}_{\mathsf{KO}}[\![n]\!](\gamma) & = & (\texttt{PUSH}\ n; \texttt{STORE}\ \mathsf{stack}_\gamma, \gamma + 1) \\
\mathcal{CA}_{\mathsf{KO}}[\![\mathsf{x}]\!](\gamma) & = & (\texttt{FETCH}\ \mathsf{x}; \texttt{STORE}\ \mathsf{stack}_\gamma, \gamma + 1) \\
\mathcal{CA}_{\mathsf{KO}}[\![a_1 + a_2]\!](\gamma) & = & \text{let } (c_2, \gamma_2) = \mathcal{CA}_{\mathsf{KO}}[\![a_2]\!](\gamma) \text{ in} \\
 & & \text{let } (c_1, \gamma_1) = \mathcal{CA}_{\mathsf{KO}}[\![a_1]\!](\gamma_2) \text{ in} \\
 & & (c_2; c_1; \texttt{FETCH}\ \mathsf{stack}_{\gamma_1}; \texttt{FETCH}\ \mathsf{stack}_\gamma; \texttt{ADD}; \texttt{STORE}\ \mathsf{stack}_\gamma, \gamma_1)
\end{array}
$$

In this concrete translation (multiplication and subtraction are similar to addition), the stack_i variables are variables not used in the rest of the program, and the environment γ simply keeps track of the stack's depth. Translating an expression simply calls the translation function with 0 as environment, and removes the final environment (which will be 0 as well) before outputting the final program. Note that all intermediate programs produced in fact leave the stack entirely empty.

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.

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

[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:

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

[8 marks]

(cont.)

Solution: We construct a reduction f : 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 $F : \mathbf{Stmt} \times \mathbb{N} \to \mathbf{Stmt} \times \mathbf{Stmt}$ by

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

: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 $\phi(\gamma(S), \gamma(T)) \in P$ since, for all m:

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

- but otherwise we have $\phi(\gamma(S), \gamma(T)) \notin P$ 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 $\phi(\gamma(\mathtt{D}),\ n)$ to obtain D and n.
2. Construct the program $S_{\mathtt{D},n}$ as:

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

3. Return $\phi(\gamma(S_{\mathtt{D},n}),\ \gamma(\mathtt{x:=1}))$

END OF PAPER