CS 620 – Theory of Computation – Fall 2017
Instructor: Marc Pomplun

Midterm Practice Exam

Duration: 1 hour and 15 minutes

You only need your writing utensils to complete this exam. No calculators, no books, and no notes allowed.

Question 1: ____ out of ____ points
Question 2: ____ out of ____ points
Question 3: ____ out of ____ points
Question 4: ____ out of ____ points

Total Score:

Grade:
Question 1: True or False?

Are the following statements true or false? Check the appropriate box for each statement. Notice that you will get 2 points for every correct answer but lose 1 point for an incorrect one; you can leave both boxes blank if you are not sure which answer is correct.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>true</strong></td>
<td><strong>false</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a) Every partially computable function is also computable.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b) There are infinitely many programs in the language $L$ that compute the function $f(x) = 2x$.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c) The class of primitive recursive functions is a subset of every PRC class.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d) If the language $L$ included the instruction “GOTO L” instead of the instruction “IF $V \neq 0$ GOTO L,” then HALT($x, y$) were computable.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e) $[3, 2, 1] = 360$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f) If function $g(x)$ is partially computable and function $h(x)$ is computable, then function $f(x) = g(h(x))$ is partially computable.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g) Every function in a PRC class can be derived from the initial functions by applying composition and primitive recursion a finite number of times.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h) $&lt;3, &lt;2, 1&gt;&gt; = 112$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i) The snapshot of a computation in the language $L$ is given by the values of all its variables and the number of the next instruction to be executed.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>j) The function $f(x, y) = x – y$ is computable in the language $L$.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Question 2: Expansive Code

Please take a look at the following program called MYSTERY:

[A] IF X≠0 GOTO B
    Y ← Y+1
    IF Y≠0 GOTO E
[B] X ← X-1
    IF X≠0 GOTO C
    Z ← Z+1
    IF Z≠0 GOTO E
[C] X ← X-1
    Z ← Z+1
    IF Z≠0 GOTO A

(a) What function does this code compute? You can describe it in English words instead of equations if you like.

(b) You now employ a macro MYSTERY(x) to use this program in another, embedding one named ENIGMA(x):

[A₃] X ← X + 1
    Y ← MYFUNC(X)

Expand the macro MYFUNC(X) and show the resulting, expanded version of ENIGMA(x) that now only contains actual L instructions. You get full points if your program is correct and bonus points if you perform the expansion exactly as described in the textbook.
Question 3: A New Type of Prime Number

Dealing with the same old prime numbers has become boring. Let us define a new type of prime number, which we will call square prime. A square prime is a natural number \( n \) greater than 1 that is a perfect square and has no other divisors than 1, \( n \), and the square root of \( n \).

For example, the smallest square prime is 4, because 4 is a perfect square (\( 2 \cdot 2 = 4 \)) and its only divisors are 1, 2, and 4. The number 9 is the next square prime, because we have \( 3 \cdot 3 = 9 \), and its only divisors are 1, 3, and 9. However, the next perfect square, 16, is not a square prime: Besides the divisors 1, 4, and 16, it is also divisible by 2 and 8. The following one, 25, is a square prime, being only divided by 1, 5, and 25. And so on…

The predicate \( \text{SquarePrime}(x) \) is TRUE, if \( x \) is a square prime, and FALSE otherwise. Show that \( \text{SquarePrime}(x) \) is a primitive recursive predicate. In your proof, you can refer to all functions and predicates that we have already shown in class to be primitive recursive.
Question 4: About Programs and Instructions

a) In our enumeration scheme for programs in the language L, each instruction is completely described by the variables a, b, and c. The variable c is defined as

\[ c = #(V) - 1. \]

Explain why the subtraction of 1 is necessary to make our enumeration scheme work.

b) Write down the instruction I for which \( #(I) = 93 \). Explain every step of your calculation.