Name:__________________

CS 620 – Theory of Computation – Spring 2019
Instructor: Marc Pomplun

Midterm Practice Exam

Sample Solutions

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. Give it your best guess if you are not sure which answer is correct.

<table>
<thead>
<tr>
<th>Statement</th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>a) Every partially computable function is also computable.</td>
<td></td>
<td>[X]</td>
</tr>
<tr>
<td>b) There are infinitely many programs in the language $\mathcal{L}$ that</td>
<td>[X]</td>
<td></td>
</tr>
<tr>
<td>compute the function $f(x) = 2x$.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c) The class of primitive recursive functions is a subset of every</td>
<td>[X]</td>
<td></td>
</tr>
<tr>
<td>PRC class.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d) There is exactly one program $P$ in the language $\mathcal{L}$ that has</td>
<td>[X]</td>
<td></td>
</tr>
<tr>
<td>the number $#(P) = 347,554$.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e) $[3, 2, 1] = 360$</td>
<td>[X]</td>
<td></td>
</tr>
<tr>
<td>f) If function $g(x)$ is partially computable and function $h(x)$ is</td>
<td>[X]</td>
<td></td>
</tr>
<tr>
<td>computable, then function $f(x) = g(h(x))$ is partially computable.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>g) Every function in a PRC class can be derived from the initial functions</td>
<td>[ ]</td>
<td>[X]</td>
</tr>
<tr>
<td>by applying composition and primitive recursion a finite number of times.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>h) $&lt;3, &lt;2, 1&gt;&gt; = 112$</td>
<td>[ ]</td>
<td>[X]</td>
</tr>
<tr>
<td>i) The snapshot of a computation in the language $\mathcal{L}$ is given by</td>
<td>[X]</td>
<td></td>
</tr>
<tr>
<td>the values of all its variables and the number of the next instruction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>to be executed.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>j) The function $f(x, y) = x - y$ is computable in the language $\mathcal{L}$</td>
<td>[ ]</td>
<td>[X]</td>
</tr>
</tbody>
</table>
Question 2: Expansive Code

Please take a look at the following program called MYSTERY:

[A]       IF X\neq 0 GOTO B
          Y \leftarrow Y+1
          IF Y\neq 0 GOTO E
[B]       X \leftarrow X-1
          IF X\neq 0 GOTO C
          Z \leftarrow Z+1
          IF Z\neq 0 GOTO E
[C]       X \leftarrow X-1
          Z \leftarrow Z+1
          IF Z\neq 0 GOTO A

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

This function computes the predicate “is_even.”

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

[A3]       X \leftarrow X + 1
          Y \leftarrow \text{MYFUNC}(X)

Expand the macro MYFUNC(X) and show the resulting, expanded version of ENIGMA(x) that now only contains actual $L$ instructions and the simple macros used in our expansion scheme such as $V \leftarrow V'$. You get full points if your program is correct and bonus points if you perform the expansion exactly as described in the textbook.
[A3]  \( X \leftarrow X + 1 \)
      \( Z_4 \leftarrow 0 \)
      \( Z_5 \leftarrow X \)
      \( Z_6 \leftarrow 0 \)

[A5]  IF \( Z_5 \neq 0 \) GOTO A6
      \( Z_4 \leftarrow Z_4 + 1 \)
      IF \( Z_4 \neq 0 \) GOTO E4

[A6]  \( Z_5 \leftarrow Z_5 - 1 \)
      IF \( Z_5 \neq 0 \) GOTO A7
      \( Z_6 \leftarrow Z_6 + 1 \)
      IF \( Z_6 \neq 0 \) GOTO E4

[A7]  \( Z_5 \leftarrow Z_5 - 1 \)
      \( Z_6 \leftarrow Z_6 + 1 \)
      IF \( Z_6 \neq 0 \) GOTO A5

[E4]  \( Y \leftarrow Z_4 \)
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.

One possibility:

$$\text{SquarePrime}(x) = x > 1 \land (\exists t \leq x (t \cdot t = x)) \land (\forall s < x (\neg (s | x) \lor s \cdot s = x \lor s = 1))$$

In this equation, $\text{SquarePrime}$ is defined through composition of other functions that we have previously shown to be primitive recursive. Therefore, $\text{SquarePrime}$ must also be primitive recursive.
Question 4 (Bonus Question): 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.

If we did not subtract 1, then c could never be 0, because our enumeration of variables starts at number 1. That would be OK for translating programs into numbers; we would still associate every possible program with a unique number. However, there would be numbers that did not translate into programs. This is because some numbers would result in c = 0 for one or more instructions, and these instructions would be undefined.

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

\[ #(I) = \langle a, b, c \rangle \]
\[ \Rightarrow 93 + 1 = 2^a(2^{b,c} + 1) \]
\[ \Rightarrow a = 1 \]
\[ \Rightarrow 2^{b,c} + 1 = 47 \]
\[ \Rightarrow b,c = 23 \]
\[ \Rightarrow 24 = 2^b(2c + 1) \]
\[ \Rightarrow b = 3 \]
\[ \Rightarrow c = 1 \]

a = 1 means that the instruction is labeled [A].
b = 3 means that the instruction type is a conditional branch to label A.
c = 1 means that the variable in this instruction is X

The instruction then is:

[ A ] \quad \text{IF } X \neq 0 \text{ GOTO A}