

Name: _____ SID: _____

CS 620 – Theory of Computation – Fall 2009
Instructor: Marc Pomplun

Midterm Practice Exam
Sample Solutions

Duration: 1 hour and 15 minutes

You only need your writing utensils to complete this exam. No calculators, no books, and no notes allowed. Use the back of any page for scratch work. There are two blank pages at the end in case you run out of room for an answer.

Question 1: ____ out of ____ points

Question 2: ____ out of ____ points

Question 3: ____ out of ____ points

Question 4: ____ out of ____ points

Question 5: ____ out of ____ points

Question 6: ____ 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.

- | | true | false |
|---|-------------------------------------|-------------------------------------|
| a) Every partially computable function is also computable. | <input type="checkbox"/> | <input checked="" type="checkbox"/> |
| b) There are infinitely many programs in the language \mathcal{L} that compute the function $f(x) = 2x$. | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| c) The class of primitive recursive functions is a subset of every PRC class. | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| d) If the language \mathcal{L} included the instruction "GOTO L" instead of the instruction "IF $V \neq 0$ GOTO L," then HALT(x, y) were computable. | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| e) $[3, 2, 1] = 360$ | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| f) If function $g(x)$ is partially computable and function $h(x)$ is computable, then function $f(x) = g(h(x))$ is partially computable. | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 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. | <input type="checkbox"/> | <input checked="" type="checkbox"/> |
| h) $\langle 3, \langle 2, 1 \rangle \rangle = 112$ | <input type="checkbox"/> | <input checked="" type="checkbox"/> |
| i) The snapshot of a computation in the language \mathcal{L} is given by the values of all its variables and the number of the next instruction to be executed. | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| j) The function $f(x, y) = x - y$ is computable in the language \mathcal{L} . | <input type="checkbox"/> | <input checked="" type="checkbox"/> |

Question 2: Say it in \mathcal{L}

Write a program in the language \mathcal{L} that computes the following function $f(x)$:

$$f(x) = \begin{cases} 0 & \text{if } x \text{ is divisible by } 3 \\ 1 & \text{otherwise} \end{cases}$$

Do not use any macros. Use comments to explain how your program works.

```

    IF X≠0 GOTO A    // If input is 0, just return 0
    GOTO E
[A]  X ← X - 1
    X ← X - 1
    IF X≠0 GOTO B    // If after 2 decrements X is 0, input was not divisible by 3
    Y ← Y + 1
    GOTO E           // return 1
[B]  X ← X - 1       // another decrement (each loop decrements X by 3)
    IF X≠0 GOTO A    // As long as X > 0 do another loop of decrement by 3
                          // If X = 0 here, the input was divisible by 3 -> return 0
```

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.

$$\text{SquarePrime}(x) = x > 1 \ \& \ (\exists t)_{\leq x}(t \cdot t = x) \ \& \ (\forall s)_{< x}(\sim(s|x) \vee s \cdot s = x \vee s = 1)$$

In this equation, SquarePrime is defined through composition of other functions that we have previously shown to be primitive recursive. Therefore, SquarePrime must also 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.

- a) 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) $\#(I) = \langle a, \langle b, c \rangle \rangle$
 $\Rightarrow 93 + 1 = 2^a(2\langle b, c \rangle + 1)$ // resolve first pairing to determine a and $\langle b, c \rangle$
 $\Rightarrow a = 1$
 $\Rightarrow 2\langle b, c \rangle + 1 = 47$
 $\Rightarrow \langle b, c \rangle = 23$
 $\Rightarrow 24 = 2^b(2c + 1)$ // resolve second pairing to determine b and c
 $\Rightarrow b = 3$
 $\Rightarrow c = 1$

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] IF X \neq 0 GOTO A

Question 5: A Mysterious Program

a) Take a look at the following program \mathcal{P} .

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

What is the function $p(x)$ computed by this program?

We can see that:

$p(0) = 0$
 $p(1) = 0$
 $p(2) = 1$
 $p(3) = 1$
 $p(4) = 1$
 $p(5) = 1$
 $p(6) = 2$
 $p(7) = 2$
 $p(8) = 2$
 $p(9) = 2$
 $p(10) = 3$
... and so on.

Therefore:

$$p(x) = \lfloor (x + 2)/4 \rfloor$$

Question 6 (Bonus Question): Investments

Arnold Schwarzenegger invests his money in a strange way: He put \$10,000,000 into account A, which gives him 10% annual interest that is automatically deposited into the same account after each year. This means that in year 0 Arnold had \$10,000,000 in account A, while in year 1 he has $\$10,000,000 + (0.1 \cdot \$10,000,000) = \$11,000,000$ in the same account, and so on. The bank always applies the floor function to the annual interest, so the account balance is always a natural number. Account B works the same way, except that Arnold's initial account balance is \$20,000,000, and the annual interest he collects is only 5%. Account C is a really bad investment, because Arnold deposited \$30,000,000 into it, but he does not collect any interest at all – its balance will be constant forever.

Now Arnold discovers his interest in mathematics and wants to define a function $s(n)$ that equals the sum off all his money in dollars in the three accounts for year n . So $s(0) = 60,000,000$. Please help Arnold to define this function $s(n)$ for all natural numbers n and show that it is primitive recursive.

$$a(0) = 10,000,000$$

$$a(n + 1) = a(n) + \lfloor a(n) / 10 \rfloor$$

$$b(0) = 20,000,000$$

$$b(n + 1) = b(n) + \lfloor b(n) / 20 \rfloor$$

$$c(n) = 30,000,000$$

Obviously, the functions $a(n)$ and $b(n)$ are primitive recursive, because they are derived from primitive recursive functions (composition of integer division and addition) by recursion. $c(n)$ is primitive recursive because it is a constant.

Then we have

$$s(n) = a(n) + b(n) + c(n) ,$$

which is primitive recursive, because it is derived by composition from the primitive recursive functions $a(n)$, $b(n)$, $c(n)$, and addition.