[redacted]

Theory of Computations,
Test 1 for the course
CS 5315, Spring 2017

$\frac{10}{10}$

Name: [redacted]

Up to 5 handwritten pages are allowed.

1. Translate, step-by-step, the following for-loop into a primitive recursive expression:

```
int z = a * b;
for (int j = 1; j <= b; j++)
   {z = z * a;}
```

You can use mult(., .) (product) in this expression.
What is the value of this function when a = 2 and b = 1?

Using mathematical notation, we get

$z(a, b, 0) = a * b$

$z(a, b, m+1) = z(a, b, m) * a$

Renaming the function z to f and the parameters a, b to n_1, n_2 respectively, we get

$f(n_1, n_2, 0) = n_1 * n_2$

$f(n_1, n_2, m+1) = f(n_1, n_2, m) * n_1$

In general, for a primitive recursive function of 3 variables.

$f(n_1, n_2, 0) = g(n_1, n_2)$

$f(n_1, n_2, m+1) = h(n_1, n_2, m, f(n_1, n_2, m))$

In this case,

$g(n_1, n_2) = mult(\pi^2_1, \pi^2_2)$

$h(n_1, n_2, m, f(n_1, n_2, m)) = mult(\pi^4_4, \pi^4_1)$

So, $z(a, b, m) \equiv PR(mult(\pi^2_1, \pi^2_2), mult(\pi^4_4, \pi^4_1))$

Here, z is a function of 3 variables. But we want a function of 2 variables.

So, $F(a, b) \equiv z(a, b, b)$

Here, $a = \pi^2_1$, $b = \pi^2_2$.

So, $F \equiv PR(mult(\pi^2_1, \pi^2_2), mult(\pi^4_4, \pi^4_1))(\pi^2_1, \pi^2_2, \pi^2_2)$

Value:

If $a = 2$ and $b = 1$, then

$$z(2,1,0) = 2 * 1$$
$$= 2$$

$$z(2,1,1) = 2 * 2$$
$$= 4$$

2. Translate, step-by-step, the following for-loop into a primitive recursive expression:

```
int z = a * b;
for(int i = 1; i <= b; i++)
  {for (int j = 1; j <= c; j++)
    {z = z * a;}}
```

$G(z, a, c)$

You can use mult(., .) in this expression.
What is the value of this function when a = 1, b = 2, and c = 2?

Let's consider the inner for loop first.

for (int j = 1 ; j <= c ; j++)
{
z = z * a ;}

Using mathematical notation, we get. where z_0 is the initial value.

$z'(z_0, a, 0) = z_0$

$z'(z_0, a, m+1) = z'(z_0, a, m) * a$

Renaming the function z' to f and the parameters z_0, a to n_1, n_2 respectively, we get

$f(n_1, n_2, 0) = n_1$

$f(n_1, n_2, m+1) = f(n_1, n_2, m) * n_2$

In general, for a primitive recursive function of 3 variables.

$f(n_1, n_2, 0) = g(n_1, n_2)$

$f(n_1, n_2, m+1) = h(n_1, n_2, m, f(n_1, n_2, m))$

In this case,

$g(n_1, n_2) = \pi^2_1$

$h(n_1, n_2, m, f(n_1, n_2, m)) = \text{mult}(\pi^4_4, \pi^4_2)$

So, $G(z, a, c) \equiv z'(z_0, a, m) \equiv PR(\pi^2_1, \text{mult}(\pi^4_4, \pi^4_2))$

Now, let's consider the outer for loop

int z = a * b;
for (int i = 1 ; i <= b ; i++)
{ z = G(z, a, c) ;
}

Using mathematical notation, we get

$z(a, b, c, 0) = a * b$

$z(a, b, c, m+1) = G(z(a, b, c, m), a, c)$

Renaming the function z to f and the parameters a, b, c to n_1, n_2, n_3 respectively, we get,

$$f(n_1, n_2, n_3, 0) = n_1 * n_2$$
$$f(n_1, n_2, n_3, m+1) = G(f(n_1, n_2, n_3, m), n_1, n_3)$$

In general, for a primitive recursive function of 4 variables,

$$f(n_1, n_2, n_3, 0) = g(n_1, n_2, n_3)$$
$$f(n_1, n_2, n_3, m+1) = h(n_1, n_2, n_3, m, f(n_1, n_2, n_3, m))$$

In this case,

$$g(n_1, n_2, n_3) = \text{mult}(\pi^3_1, \pi^3_2)$$
$$h(n_1, n_2, n_3, m, f(n_1, n_2, n_3, m)) = G(\pi^5_5, \pi^5_1, \pi^5_3)$$

So, $z(a, b, c, m) \equiv PR(\text{mult}(\pi^3_1, \pi^3_2), G(\pi^5_5, \pi^5_1, \pi^5_3))$,

where, $G \equiv PR(\pi^2_1, \text{mult}(\pi^4_4, \pi^4_2))$.

But we are looking for a function of three variables, so

$$F(a, b, c) = z(a, b, c, b).$$

Value: $a = 1, b = 2, c = 2$

$z_0 = 1 * 2 = 2$

$z'(2, 1, 0) = 2$

$z'(2, 1, 1) = 2 * 1 = 2$

$z'(2, 1, 2) = 2 * 1 = 2$

$z(1, 2, 2, 0) = 1 * 2 = 2$

$z(1, 2, 2, 1) = 2 * 1 = 2$

$z(1, 2, 2, 2) = 2 * 1 = 2$

$z = 1 * 2 = 2$

$i = 1,$

$j = 1, \; z = 2 * 1 = 2$

$j = 2, \; z = 2 * 1 = 2$

$i = 2,$

$j = 1, \; z = 2 * 1 = 2$

$j = 2, \; z = 2 * 1 = 2$

So, the value is 2

10/10

3. Translate, step-by-step, the following primitive recursive function into a for-loop:

$$f = \sigma(PR(mult(\pi^2_1, \pi^2_2), mult(\pi^4_1, \pi^4_2, \pi^4_4)).$$

For this function f, what is the value f(2, 3, 1)? Provide an explicit formula for the corresponding function.

let,

$f = \sigma \cdot F$ where

$F = PR(mult(\pi^2_1, \pi^2_2), mult(\pi^4_1, \pi^4_2, \pi^4_4))$

$\therefore F(n_1, n_2, 0) = n_1 * n_2$

$F(n_1, n_2, m+1) = n_1 * n_2 * F(n_1, n_2, m)$

$f(n_1, n_2, m) = F(n_1, n_2, m) + 1$

```
∴ int F = n1*n2
for (int i=1; i<=m; i++)
  { F = n1*n2*F; }
f = F+1;
```

$F(2,3,0) = 2*3 = 6$

$F(2,3,1) = 2*3*F(2,3,0) = 6*6 = 36.$

$f(2,3,1) = 1 + F(2,3,1) = 1+36 = 37$

Formula:

$f(x, y, z) = (xy)^{z+1} + 1$

Ans:

20/20

4-5. Prove, from scratch, that integer division a / b is primitive recursive. Start with the definitions of a primitive recursive function, and use only this definition in your proof -- do not simply mention results that we proved in class, prove them.

Definition : A function is called primitive recursive (p.r.) if can be obtained from 0, σ and π^k_i by using composition and primitive recursion.

Integer division can be written as

$$div(a, 0) = 0$$

$$div(a, m+1) = \text{if } rem(a, m+1) = 0 \text{ then } div(a,m)+1 \text{ else } 0$$

To prove that integer division is p.r. we have to prove that rem (remainder) is p.r. Now, let's describe rem as

$$rem(a,0) = 0$$

$$rem(a, m+1) = \text{if } (rem(a,m)+1 < a) \text{ then } rem(a,m)+1 \text{ else } 0$$

To prove that rem is p.r, it is sufficient to prove that

(i) if-~~then~~-then-else is p.r.

(ii) < is p.r.

(i) if-then-else is p.r. :

we know that if p, g, h are p.r. then if p then g else h is also p.r. Let's prove it.

proof: Here, the result is either p and g or ¬p and h.

So, $p * g + (1-p) * h$

If p is true (p=1): $1 * g + 0 * h = g$

If p is false (p=0): $0 * g + 1 * h = h$

To prove this, we need to prove that addition, multiplication and subtraction are also p.r.

<u>Addition is p.r. :</u>

By definition, $a+b$ means $a+1+\ldots+1$. Thus

$add(a,0) = a$

$add(a,m+1) = add(a,m)+1$

In general for a p.r. function of 2 variables,

$$f(n_1, 0) = g(n_1)$$

$$f(n_1, m+1) = h(n_1, m, f(n_1, m))$$

In this case,

$$f(n_1, 0) = n_1$$

$$f(n_1, m+1) = f(n_1, m) + 1$$

Thus, here, $g = \pi^1{}_1$, $h = \sigma \circ \pi^3{}_3$

So, $add \equiv PR(\pi^1{}_1, \sigma \circ \pi^3{}_3)$.

<u>Multiplication is p.r. :</u>

By definition, $a * b$ means $\underbrace{a+a+\ldots+a}_{b \text{ times}}$. Thus,

$mult(a,0) = a$

$mult(a,m+1) = mult(a,m) + a$

Renaming mult to f and a to n_1,

$$f(n_1, 0) = n_1 = g(n_1)$$

$$f(n_1, m+1) = \underbrace{f(n_1, m) + n_1}_{= h(n_1, m, f(n_1, m))}$$

So, $g = \pi^1{}_1$, $h = add(\pi^3{}_3, \pi^3{}_1)$

$mult \equiv PR(\pi^1{}_1, add(\pi^3{}_3, \pi^3{}_1))$

prev is p.r.:

$$prev(n) = \begin{cases} n-1, & \text{if } n \geq 1 \\ 0 & \text{otherwise} \end{cases}$$

$prev(0) = 0$
$prev(m+1) = m$

so, $g = 0$, $h = \pi^2_1$

so, $prev = PR(0, \pi^2_1)$

subtraction is p.r.:

$$a \dot{-} b = a \dot{-} \underbrace{1 \ldots \dot{-} 1}_{b \text{ times}}$$

$sub(a, 0) = a$
$sub(a, m+1) = prev(sub(a, m))$

So, $g = \pi^1_1$, $h = prev(\pi^3_3)$

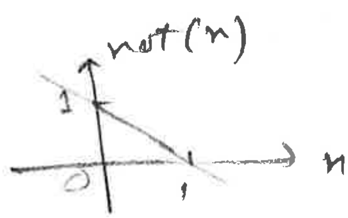
so, $sub = PR(\pi^1_1, pre(\pi^3_3))$

(ii) $<$ is p.r.:

To prove this, we need to prove that and(&&), not(!), or(||), eq to 0, $=$, $\leq$ are p.r.

not is p.r.:

$not(0) = 1$
$not(1) = 0$
$not(n) = 1 \dot{-} n$
$= \sigma(0) \dot{-} n$



So by definition, not (!) is p.r.

and is p.r.:

b \ a	0	1
0	0	0
1	0	1

&& is multiplication.
So is p.r.

or is p.r.:

b \ a	0	1
0	0	1
1	1	1

By De Morgan's theory,

$a \,||\, b = \text{not}(\text{and}(\text{not}(a), \text{not}(b)))$

and, not are p.r. So $||$ is p.r.

eq to 0 is p.r.

$$\text{eq to } 0(n) = \begin{cases} \text{true} & \text{if } n = 0 \\ \text{false} & \text{otherwise} \end{cases}$$

$\text{eq to } 0(0) = 1$

$\text{eq to } 0(m+1) = 1$

$g = 1 = \sigma \circ 0$

$h = 0$

$\text{eq to } 0 = PR(\sigma \circ 0, 0)$

~~$=$: $a = b \Leftrightarrow a \dot{-} b = 0 \Leftrightarrow a \le b$~~

~~because $a \dot{-} b = \begin{cases} a-b, & \text{if } a > b \\ 0 & \text{otherwise} \end{cases}$~~

~~so $=$ is p.r.~~

$\le$ is p.r.: $a \le b \Leftrightarrow \text{eq to } 0(a \dot{-} b)$

eq to 0, $\dot{-}$ are p.r. So $\le$ is p.r.

$=$ is p.r.: $a = b \Leftrightarrow a \le b \;\&\&\; b \le a$

$\le$, && are p.r. So $=$ is p.r.

$\ne$ is p.r.: $!(a = b)$

!, = are p.r. So $\ne$ is p.r.

$<$ is p.r.: $a \le b \;\&\&\; !(a=b)$

$\le$, &&, ! are p.r. So $<$ is p.r.

Hence, we are done.

6. Prove that the following function f(a, b) is μ-recursive: f(a, b) = a !! b when a and b are both equal to either 0 or 1, and f(a, b) is undefined for other pairs (a, b).

$$f(a,b) = \begin{cases} a \,!!\, b, & \text{if } a \in \{0,1\} \text{ and } b \in \{0,1\} \\ \text{undefined} & \text{otherwise} \end{cases} = \begin{cases} 0, & \text{if } a=0 \,\&\, b=0 \\ 1, & \text{if } a=0 \,\&\, b=1 \\ 1, & \text{if } a=1 \,\&\, b=0 \\ 1, & \text{if } a=1 \,\&\, b=1 \end{cases}$$

we have four cases.

Case - 1: $a = 0$ and $b = 0$, then $m = 0$

Case - 2: $a = 0$ and $b = 1$, then $m = 1$

Case - 3: $a = 1$ and $b = 0$ then $m = 1$

Case - 4: $a = 1$ and $b = 1$, then $m = 1$

To express the given function using μ-recursion, we need to find a relationship among a, b and m such that m is the smallest value that satisfies the properties of the given function.

Let's describe the given function as μ-recursion,

$$\mu m.(\,(a==0 \;\&\&\; b==0 \;\&\&\; m==0) \,||$$
$$(a==0 \;\&\&\; b==1 \;\&\&\; m==1) \,||$$
$$(a==1 \;\&\&\; b==0 \;\&\&\; m==1) \,||$$
$$(a==1 \;\&\&\; b==1 \;\&\&\; m==1))$$

7. Translate the following μ-recursive expression into a while-loop:

$$f(a, b) = \mu m.(m * a > b).$$

For this function f, what is the value of f(2, 5)?

10/10

While-loop:

A while loop is like a for loop but instead of fixed number of iteration, the number of iteration is the smallest number m at which some condition is satisfied.

$P(n_1, \ldots, n_k, m)$ - condition (true or false, 1 or 0)

Smallest natural number m for which this condition is true

$\mu m. P(\bar{n}, m)$.

Let's translate the given μ-recursive expression into a while-loop:

```
int m = 0
while (! ((m * a) > b))
{
    m++;
}
```

Value:

$f(2, 5) = 3$

$\boxed{2}$ a $\boxed{5}$ b $\boxed{\cancel{0}}$ $\cancel{1}$ $\cancel{2}$ 3 m

20/20

8-9. In class, we proved that not every computable function is primitive recursive, by using Cantor's diagonal construction to define an auxiliary function f(n) which is computable but not primitive recursive. What if, in addition to 0, π^k_i, and σ, we also allow this auxiliary function f(n) in our constructions? Let us call functions that can be obtained from 0, π^k_i, σ, and f by using composition and primitive recursion *f-primitive recursive* functions. Will then every computable function be f-primitive recursive? Prove that your answer is correct.

Definition:

An f-primitive recursive function (f-p.r.f) is a function that can be obtained from 0, π^k_i, σ and f by using composition and primitive recursion.

In addition to 0, π^k_i and σ, if we also allow f(n) in our constructions, then every computable function will not be f-p.r.

To prove this, at first we need to know what is the code of a f-p.r.f.

f-p.r.f can be expressed using symbols generated via application such as LaTeX. Therefore, these symbols are represented via a sequence of bits (0's and 1's). In that case, we have to append 1 in front of this sequence of 0's and 1's, because we want to interpret a sequence as a number. If we don't append, different sequences will correspond to the same number, for example, 001, 01, 1 are not same binary sequence but represents same number 1. So we will not be able to reconstruct the sequence from number. After appending, we interpret the resulting sequence as a natural number. This number is called a code of a f-p.r.f.

Now, we are going to present a lemma which is useful to the proof.

<u>Lemma:</u>

There exists an algorithm that given a natural number c, checks whether c is a code of a f-p.r.f. and if yes, returns the executable file, f_c, computing the value of this f-p.r.f.

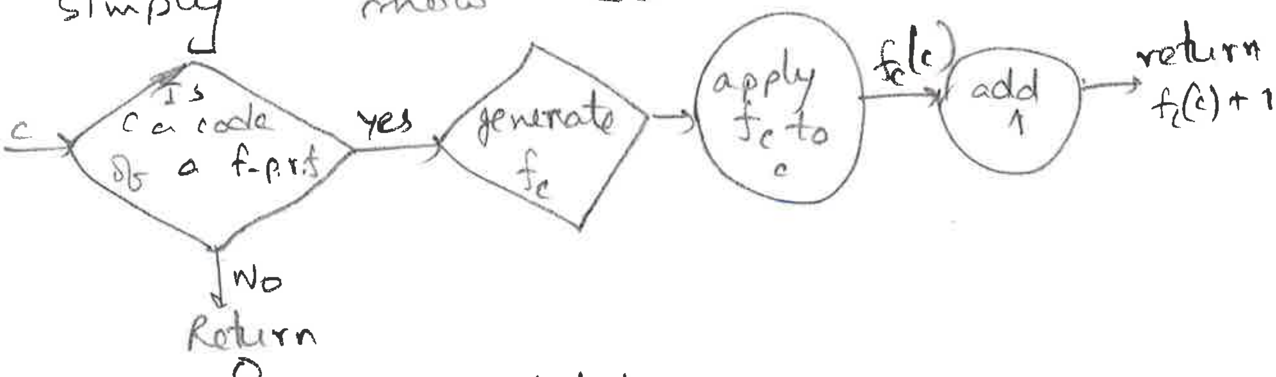
Let's define a function,

Same letter??

$$f(c) = \begin{cases} f_c(c)+1, & \text{if } c \text{ is a code of a f-p.r.f} \\ 0, & \text{otherwise} \end{cases}$$

We will prove that f is computable and that f is not p.r.

To prove that f is computable, we simply show how to compute it.



So f is computable

Let's now prove that f is not f-p.r.:

We prove by contradiction. Let's assume that f is f-p.r. and let's show that this assumption leads to a contradiction.

Since f is a f-p.r, it has a code, c_0. So by applying the algorithm to c_0, we get an executable file, f_{c_0}, that computes $f(c)$.

This means that for every input n

$$f_{c_0}(n) = f(n)$$

In particular, it is true for $n = c_0$

$$f_{c_0}(c_0) = f(c_0) \quad \text{--- } (1)$$

By definition of $f(c)$,

$$f(c_0) = f_{c_0}(c_0) + 1 \quad \text{--- } (2)$$

From (1) and (2), we can write

$$f_{c_0}(c_0) = f_{c_0}(c_0) + 1$$

$\Rightarrow \quad 0 = 1$, which is a contradiction

So, f is not f-p.r.

Therefore, f is computable but not f-p.r.