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

Name: 

Up to 5 handwritten pages are allowed.

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

```java
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:

\[
z(a, b, 0) = a \times b
\]

\[
z(a, b, m+1) = z(a, b, m) \times a
\]

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

\[
f(n_1, n_2, 0) = n_1 \times n_2
\]

\[
f(n_1, n_2, m+1) = f(n_1, n_2, m) \times n_1
\]

In general, for a primitive recursive function \( f \) of 3 variables:

\[
f(n_1, n_2, 0) = q(n_1, n_2)
\]

\[
f(n_1, n_2, m+1) = h(n_1, n_2, m, f(n_1, n_2, m))
\]

In this case:

\[
q(n_1, n_2) = \text{mult}(x_1^2, x_2^2)
\]

\[
h(n_1, n_2, m, f(n_1, n_2, m)) = \text{mult}(x_4^2, x_1^2)
\]

So, \( z(a, b, m) \equiv \text{PR}(\text{mult}(x_1^2, x_2^2), \text{mult}(x_4^2, x_1^2)) \)

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, \( a = x_1^2 \), \( b = x_2^2 \).

So, \( f \equiv \text{PR}(\text{mult}(x_1^2, x_2^2), \text{mult}(x_4^2, x_1^2)) \)

(file://Q:/cs5315.17/test1.html)

1/31/2017
Value:
If \( a = 2 \) and \( b = 1 \), then
\[
\begin{align*}
2(2, 1, 0) &= 2 \times 1 \\
&= 2 \\
2(2, 1, 1) &= 2 \times 2 \\
&= 4
\end{align*}
\]
2. Translate, step-by-step, the following for-loop into a primitive recursive expression:

```c
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.

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

Using mathematical notation, we get:

\[
\varepsilon(z_0, a, 0) = z_0
\]

where \( z_0 \) is the initial value.

\[
\varepsilon'(z_0, a, m+1) = \varepsilon'(z_0, a, m) \ast a
\]

Renaming the function \( \varepsilon' \) to \( f \) and the parameters \( z_0, a \) to \( n_1, n_2 \) respectively, we get:

\[
f(n_1, n_2, 0) = n_1
\]

\[
f(n_1, n_2, m+1) = f(n_1, n_2, m) \ast n_2
\]

In general, for a primitive recursive function \( f \) in 3 variables:

\[
f(n_1, n_2, 0) = f(n_1, n_2)
\]

\[
f(n_1, n_2, m+1) = h(n_1, n_2, m, f(n_1, n_2, m))
\]

In this case,

\[
h(n_1, n_2, m, f(n_1, n_2, m)) = \text{mult}(x_1^4, x_2^4)
\]

So,

\[
G_1(z, a, c) \equiv \varepsilon(z_0, a, m) \equiv \text{PR}(x_1^2, \text{mult}(x_1^4, x_2^4))
\]

Now, let’s consider the outer for loop:

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

Using mathematical notation, we get:

\[
z(a, b, c, 0) = a \ast b
\]

\[
z(a, b, c, m+1) = G(z(a, b, c, m), a, c)
\]
Renaming the function $\varepsilon$ to $f$ and the parameters $a, b, c$ to $n_1, n_2, n_3$ respectively, we get,

$$f(n_1, n_2, n_3, 0) = n_1 \ast n_2$$
$$f(n_1, n_2, n_3, m+1) = \delta(f(n_1, n_2, n_3, m), n_1, n_2)$$

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

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

In this case,

$$q(n_1, n_2, n_3) = \text{mult}(n_1^3, n_2^3)$$
$$h(n_1, n_2, n_3, m, f(n_1, n_2, n_3, m)) = \delta_1(n_5^5, n_1^5, n_3^5)$$

So,

$$\varepsilon(a, b, c, m) \equiv \text{PR}(\text{mult}(n_1^3, n_2^3), \delta_1(n_5^5, n_1^5, n_3^5))$$

where, $\delta_1 \equiv \text{PR}(n_1^2, \text{mult}(n_1^4, n_2^4))$.

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

$$F(a, b, c) = \varepsilon(a, b, c, b)$$

---

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

$$\varepsilon = 1 \ast 2 = 2$$
$$\varepsilon(2, 1, 0) = 2$$
$$\varepsilon(2, 1, 1) = 2 \ast 1 = 2$$
$$\varepsilon(2, 1, 2) = 2 \ast 1 = 2$$
$$\varepsilon(1, 2, 2, 0) = 1 \ast 2 = 2$$
$$\varepsilon(1, 2, 2, 1) = 2 \ast 1 = 2$$
$$\varepsilon(1, 2, 2, 2) = 2 \ast 1 = 2$$

So, the value is 2
3. Translate, step-by-step, the following primitive recursive function into a for-loop:

\[ f = \sigma(\text{PR}(\text{mult}(n_1^2, n_2^2), \text{mult}(n_1^4, n_2^4))). \]

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

\[
\begin{align*}
\text{let,} \\
\quad f &= \sigma \cdot F \\
\quad F &= \text{PR} \left( \text{mult} \left( n_1^2, n_2^2 \right), \text{mult} \left( n_1^4, n_2^4 \right) \right) \\
F(n_1, n_2, 0) &= n_1 \times n_2 \\
F(n_1, n_2, m+1) &= n_1 \times n_2 \times F(n_1, n_2, m) \\
f(n_1, n_2, m) &= F(n_1, n_2, m) + 1
\end{align*}
\]

\[
\begin{align*}
\text{int} & \quad F = n_1 \times n_2 \\
\text{for} (\text{int} \ i = 1; i \leq m; i++) \\
& \quad \{ F = n_1 \times n_2 \times F; \} \\
F &= F + 1 \\
F(2, 3, 0) &= 2 \times 3 = 6 \\
F(2, 3, 1) &= 2 \times 3 \times F(2, 3, 0) = 6 \times 6 = 36. \\
f(2, 3, 1) &= 1 + F(2, 3, 1) = 1 + 36 = 37
\end{align*}
\]

Formula:
\[ f(x, y, 2) = (x \times y)^{2+1} \]

Ans:
4.5. Prove, from scratch, that integer division \( a \div b \) 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 if it can be obtained from 0, \( \times \), and \( \min \) by using composition and primitive recursion.

Integer division can be written as

\[
\text{div}(a, 0) = 0 \\
\text{div}(a, m+1) = \begin{cases} 
\text{rem}(a, m+1) = 0 & \text{if } \text{div}(a, m) = 0 \\
\text{div}(a, m+1) = \text{rem}(a, m+1) & \text{else}
\end{cases}
\]

To prove that integer division is P.R., we have to prove that \( \text{rem} \) (remainder) is P.R. Now, let's describe \( \text{rem} \) as

\[
\text{rem}(a, 0) = 0 \\
\text{rem}(a, m+1) = \begin{cases} 
\text{rem}(a, m) & \text{if } \text{rem}(a, m+1) < a \\
\text{rem}(a, m+1) & \text{else}
\end{cases}
\]

To prove that \( \text{rem} \) is P.R., it is sufficient to prove that

(i) if \( a \leq b \) then else is P.R.

(ii) \( \exists \) is P.R.

(iii) if \( \text{then else} \) is P.R.: we know that if \( p, q, h \) are P.R. then if \( p \) then \( q \) else \( h \) is also P.R. Let's prove it.

**proof:** Here, the result is either \( p \) and \( h \) or \( \neg p \) and \( h \).

\[
P \times q + (1-P) \times h
\]

If \( p \) is true \( (p=1) \):

\[
1 \times q + 0 \times h = q
\]

If \( p \) is false \( (p=0) \):

\[
0 \times q + 1 \times h = h
\]
To prove this, we need to prove that addition, multiplication and subtraction are also p.r.

**Addition is p.r.:**

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

\[
\begin{align*}
\text{add} \ (a, 0) &= a \\
\text{add} \ (a, m+1) &= \text{add} \ (a, m) + 1
\end{align*}
\]

In general, for a p.r. function \( f \) of 2 variables,

\[
\begin{align*}
f (n_1, 0) &= \varphi (n_1) \\
f (n_1, m+1) &= h (n_1, m, f (n_1, m))
\end{align*}
\]

In this case,

\[
\begin{align*}
f (n_1, 0) &= n_1 \\
f (n_1, m+1) &= f (n_1, m) + 1
\end{align*}
\]

Thus, here, \( \varphi = \pi_1 \), \( h = \sigma \circ \pi_3 \).

So, \( \text{add} = \text{PR} (\pi_1, \sigma \circ \pi_3) \).

**Multiplication is p.r.:**

By definition, \( a * b \) means \( a + a + \ldots + a \). Thus, \( a \) times

\[
\begin{align*}
\text{mult} \ (a, 0) &= a \\
\text{mult} \ (a, m+1) &= \text{mult} \ (a, m) + a
\end{align*}
\]

Renaming mult to \( f \) and \( a \) to \( n_1 \),

\[
\begin{align*}
f (n_1, 0) &= n_1 = \varphi (n_1) \\
f (n_1, m+1) &= f (n_1, m) + n_1 = h (n_1, m, f (n_1, m))
\end{align*}
\]

So, \( \varphi = \pi_1 \), \( h = \text{add} (\pi_3 \circ \pi_3, \pi_3) \).

Thus, \( \text{mult} = \text{PR} (\pi_1, \text{add} (\pi_3 \circ \pi_3, \pi_3)) \).
\[ \text{prev is p.r.} \]
\[ \text{prev}(n) = \begin{cases} n-1, & \text{if } n \geq 1 \\ 0 & \text{otherwise} \end{cases} \]

\[ \text{prev}(0) = 0 \]
\[ \text{prev}(m+1) = m \]

So, \[ q = 0, \quad h = \pi_{1}^2 \]

So, \[ \text{prev} = \text{PR}(0, \pi_{1}^2) \]

\underline{Subtraction is p.r. :}

\[ a - b = a \times \frac{1}{b} \text{ times} \]

\[ \text{sub}(a, 0) = a \]
\[ \text{sub}(a, m+1) = \text{prev}(\text{sub}(a, m)) \]

So, \[ q = \pi_{1}^1, \quad h = \text{prev}(\pi_{3}^3) \]

So, \[ \text{sub} = \text{PR}(\pi_{1}^1, \text{pre}(\pi_{3}^3)) \]

\[(\text{i}) \mid < \text{ is p.r. :}\]

To prove this, we need to prove that and (\&\&), not (\text{!}) \equiv 0, =, \leq are p.r.

\[ \text{not is p.r.:} \]
\[ \text{not}(0) = 1 \]
\[ \text{not}(1) = 0 \]
\[ \text{not}(n) = 1 - n \]
\[ = \pi(0) = n \]

So by definition, not (\text{!}) is p.r.

\[ \text{and is p.r.:} \]
\[ \begin{array}{ccc}
0 \times 0 & = & 0 \\
0 \times 0 & = & 0 \\
1 \times 0 & = & 1
\end{array} \]

\[ \text{\& is multiplication.} \]

So is p.r.
By De Morgan's theory,
all \( b \) is not (and \( \neg \text{not}(a) \), \( \neg \text{not}(b) \))
and, not are p.r. So \( 1 \) is p.r.

\[ \begin{align*}
\text{eq to } 0 \text{ is p.r.} \\
\text{eq to } 0(0) &= 1 \\
\text{eq to } 0(n) &= 0 \\
q &= 1 = 0, 0, 0 \\
h &= 0 \\
\text{eq to } 0 &= \text{PR}(0, 0, 0, 0)
\end{align*} \]

\[ a = b \iff a \cdot b = 0 \iff a \leq b \]
because \( a \cdot b = 0 \) if \( a > b \) otherwise

So \( = \) is p.r.

\[ \leq \text{ is p.r.} : a \leq b \implies \text{eq to } 0(a = b) \]

\[ \text{eq to } 0, = \text{ are p.r. So } \leq \text{ is p.r.} \]

\[ \text{is p.r.: } a = b \implies a \leq b \ \&\ \& \ b \leq a \]

\[ \leq, \&\& \text{ are p.r. So } = \text{ is p.r.} \]

\[ \neq \text{ is p.r.: } ! (a = b) \]

\[ =, ! \text{ are p.r. So } \neq \text{ is p.r.} \]

\[ \leq \text{ is p.r.: } a \leq b \ \&\& ! (a > b) \]

\[ \leq, \&\& ! \text{ are p.r. So } < \text{ is p.r.} \]

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

\[
f(a, b) = \begin{cases} 
    a \oplus b, & \text{if } a \neq 0, 1 \text{ and } b \neq 0, 1 \\
    \text{undefined otherwise}
\end{cases}
\]

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 \( \mu \)-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 \( \mu \)-recursion:

\[
\mu m. ( (a = 0 \land b = 0 \land m = 0) \lor (a = 0 \land b = 1 \land m = 1) \lor (a = 1 \land b = 0 \land m = 1) \lor (a = 1 \land b = 1 \land m = 1) )
\]
7. Translate the following μ-recursive expression into a while-loop:
   \[ f(a, b) = \mu m. (m \cdot a \geq b) \]
For this function \( f \), what is the value of \( f(2, 5) \)?

**While-loop:**

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

\[ P(n_1, \ldots, n_k, m) - \text{condition} \]

(true or false, 1 or 0)

Smallest natural number \( m \) for which this condition is true:

\[ \mu m. P(m, m) \]

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

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

**Value:**

\[ f(2, 5) = 3 \]
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, \( x_k \), and \( \sigma \), we also allow this auxiliary function \( f(n) \) in our constructions? Let us call functions that can be obtained from 0, \( x_k \), \( \sigma \), 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, \( x_k \), \( \sigma \), and \( f \) by using composition and primitive recursion.

In addition to 0, \( x_k \), and \( \sigma \), 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 recognize 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.

**Lemma:**
There exists an algorithm that given a natural number $e$, checks whether $e$ is a code of a $f$-p.r.f. and if yes, returns the executable file, $f_e$, computing the value of this $f$-p.r.f.

Let's define a function $f_e : \mathbb{N} \rightarrow \{0, 1\}$, if $e$ is a code of a $f$-p.r.f. \[ f_e(0) = 0, \quad \text{otherwise} \]

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.s.

we prove by contradiction. Let's assume that $f$ is $f$-p.r.s. 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 $n = c_0$

$$f_{c_0}(c_0) = f(c_0)$$  \[1\]

By definition of $f(c)$,

$$f_{c_0}(c_0) = f_{c_0}(c_0) + 1$$  \[2\]

From $1$ and $2$, we can write

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

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

So, $f$ is not $f$-p.r.

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