Theory of Computations,
Test 1 for the course
CS 5315, Spring 2018

Name:

Up to 5 handwritten pages are allowed.

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

```c
int x = a + b;
for (int i = 1; i <= a; i++)
    x = x * b;
```

You can use \text{add}(\ldots) (sum) and \text{mult}(\ldots) (product) in this expression.

What is the value of this function when \(a = 2\) and \(b = 1\)?

The function, \(x\), relies on \(a, b, 0\), and the previous iteration of \(x\):

\[
x(a, b, 0) = a + b
\]

\[
x(a, b, i) = x(a, b, i-1) \times b
\]

To look at the general function of \(k = 2\) variables:

\[
f(n, n_0, 0) = g(n, n_0)
\]

\[
f(n, n_0, m+1) = h(n, n_0, m, f(n, n_0, m))
\]

Change \(x \rightarrow f\), \(a \rightarrow n\), \(b \rightarrow n_0\), \(i \rightarrow m+1\) (so \(i-1 = m\)):

\[
f(n, n_0, 0) = n + n_0
\]

\[
f(n, n_0, m+1) = f(n, n_0, m) \times n_0
\]

\[
h(\ldots) = f(n, n_0, m) \times n_0
\]

To represent as a P. R. function, \(x = \text{PR}(g, h)\):

\[
x = \text{PR}(\text{add}(\Pi^0_1, \Pi^0_2), \text{mult}(\Pi^0_1, \Pi^0_2))
\]
1. Translate, step-by-step, the following for-loop into a primitive recursive expression:

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

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

\[ x(a,b,0) = a \cdot b \]
\[ x(a,b,m+1) = x(a,b,m) \cdot b \]

\[ f(n_1,n_2,0) = n_1 \cdot n_2 \]
\[ f(n_1,n_2,m+1) = f(n_1,n_2,m) \cdot n_2 \]

\[ g = \text{add}(n_1^2, n_2^2) \equiv \text{PE}(\text{add}(\pi_1^2, \pi_2^2), \text{mult}(\pi_4^4, \pi_5^4)) \]
\[ h = \text{mult}(\pi_4^4, \pi_5^4) \]

\[ + (a,b) \]
\[ \begin{array}{ccc}
    2 & 1 & 3 \\
    a & b & c \\
\end{array} \]

\[ x = 3 \]
\[ x = 3 \cdot 1 = 3 \]
\[ x = 3 \cdot 1 = \boxed{3} \]
The theory of computations,
Test 1 for the course
CS 5315, Spring 2018

Name:

Up to 5 handwritten pages are allowed.

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

```c
int x = a + b;
for (int i = 1; i <= a; i++)
    x = x + b;
```

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

Mathematical notation of the loop:

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

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

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

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

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

General function of \( f \) is with \( 3 \) var

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

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

So, comparing with general expression:

\[ g = n_1^2 + n_2^2 = \text{add}\left( n_1^2, n_2^2 \right) \]

\[ h = \text{mult}\left( n_1^d, n_2^d \right) \]
So, $z(a, b, m) = PR \left( add \left( \pi_1^2, \pi_2^4 \right), mult \left( \pi_1^a, \pi_2^b \right) \right)$

Value:

$a = 2$, $b = 1$

$z(2, 1, 0) = 2 + 1 = 3$

$z(2, 1, 1) = 2 \times 1 = 3$

$z(2, 1, 2) = 3 \times 1 = 3 \text{ (not 3)}$
2. Translate, step-by-step, the following for-loop into a primitive recursive expression:

```java
int x = a + b;
for (int i = 1; i <= a; i++)
    (for (int j = 1; j <= c; j++)
        t = (x, b, c)
    x = x * b;
)
```

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

```
x = x0
for (int j = 1; j <= c; j++)
x = x * b;
```

```
x(0, b, 0) = x0
x(0, b, m+1) = x(0, b, m) * b
```

```
t(n1, n2, 0) = n1
```

```
t(n1, n2, m+1) = t(n1, n2, m) + n2
```

```
g(n1, n2) = \text{add}(n1^3, n2^3)
```

```
h(n1, n2, m, t(n1, n2, m)) = \text{mult}(n1^3, n2^3)
```

```
= PR(\nu^3, \text{mult}(\nu^3, \nu^3))
```

```
x(a, b, c, 0) = a + b
x(a, b, c, m+1) = t(x(a, b, c, m), b, c)
```

```
t(n1, n2, n3, 0) = n1 + n2
```

```
t(n1, n2, n3, m+1) = t(t(n1, n2, n3, m), n2, n3)
```

```
g = \text{add}(\nu^3, \nu^3)
```

```
h = t(\nu^5, \nu^5, \nu^5)
```

```
= PR(\text{add}(\nu^3, \nu^3), t(\nu^5, \nu^5, \nu^5))
```

---

x = 3
x = 3 * 2 = 6
x = 6 * 2 = 12
2. Translate, step-by-step, the following for-loop into a primitive recursive expression:

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

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

Let's consider the inner loop first:

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

Mathematical eqn:

\[ x' (b, x, 0) = x_0 \]  where \( x_0 \) initial val

\[ x' (b, x, m+1) = x' (a, 2, m) \times b \]

Relating:

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

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

General equation

\[ f(n, n, 0) = g(n, n) \]

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

Comparing both:

\[ g(n, n) = n \times n = \pi_2^2 \]

\[ h = f(n, n, m) \times n = \pi_4^2 \times \pi_1^a \]

\[ \pi_1^a = \text{mult} (\pi_4^a, \pi_4^a) \]

Considering:

\[ C(b, x, c) = x' (b, x, m) \]

\[ = pr (\pi_2^2, \text{mult} (\pi_4^a, \pi_4^a)) \]
Now consider the outer loop

\[
\text{int } x = a + b \\
\text{for } (\text{int } i = 1; i < a, i++) \\
\quad x = G \left( b, 2, c \right) \]

\[
x(a, b, c, 0) = a + b \]

\[
x(a, b, c, m+1) = G \left( b, x(a, b, c, m), c \right) \]

\[
f(n_1, n_2, n_3, 0) = n_1 + n_2 = g(n_1, n_2, n_3) \]

\[
f(n_1, n_2, n_3, m+1) = G \left( n_2, f(n_1, n_2, n_3, m), n_3 \right) = h(n, n_2, n_3, m, f(n_1, n_2, n_3)) \]

Comparing with general

\[
g = \text{add} \left( \pi_1, \pi_2 \right) \]

\[
h = G \left( \pi_2 \frac{5}{2}, \pi_2 \frac{5}{2}, \pi_2 \frac{5}{2} \right) \]

So \( x(a, b, c, m) = PR \left( \text{add} \left( \pi_2 \frac{5}{2}, \pi_2 \frac{5}{2} \right), G \left( \pi_2 \frac{5}{2}, \pi_2 \frac{5}{2}, \pi_2 \frac{5}{2} \right) \right) \)

\[
\text{Value: } a = 1, \ b = 2, \ c = 2 \]

\[
\chi(1, 2, 2, 0) = 3 \]

\[
\chi(1, 2, 2, 1) = G \left( 1, 2, 2 \right) \]

\[
\Leftrightarrow \chi' (1, 2, 0) = 3 \]

\[
\Leftrightarrow \chi' (1, 2, 1) = 6 \]

\[
\Leftrightarrow \chi' (1, 2, 2) = 12 \]

\[
= 12 \left( f(n) \right) \]
3. Translate, step-by-step, the following primitive recursive function into a for-loop:

\[ f = \sigma(\text{PR}(\text{add}(\pi_1, \pi_2), \text{add}(\pi_3, \pi_4))). \]

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

\[
\begin{align*}
\text{Left} & \quad f = \sigma \circ F \\
F & = \text{PR}(\text{add}(\pi_1, \pi_2), \text{add}(\pi_3, \pi_4)) \\
F & = \text{PR}(g, h) \\
\text{base} \quad f(n, 0, m) & = g(n, m) \\
f(n_1, \ldots, n, m+1) & = h(n_1, \ldots, n, m, f(n_1, \ldots, n, m)) \\
\text{So,} \quad F(n_1, n_2, 0) & = g(n_1, n_2) = n_1 + n_2 \\
F(n_1, n, m+1) & = h(n_1, n, m, f(n_1, n, m)) \\
& = n_1 + m + f(n_1, n, m)
\end{align*}
\]

\[ \text{Code:} \]

\[
\begin{align*}
\text{int } F & = n_1 + n_2 \\
\text{for } (\text{int } i = 1; i \leq m; i++) & \\
& \quad F = n_1 + (i - 1) \cdot F \\
& \quad f = F + 1 \\
F(2, 3, 0) & = 5 \quad 0 \\
F(2, 3, 1) & = 2 + y + 5 = 8 \\
f(2, 3, 0) & = 7 \quad 8 \\
\text{Formula:} \quad f(x, y, z) & = x + y + (x^2) + 1
\end{align*}
\]
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 it can be obtained from $0, \omega, \eta$ by using composition and primitive recursion.

Integer division $a/b$ can be written as

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

To prove $a/b$ is P.R. we have to prove that remainder is also P.R.

Let's describe remainder as

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

To prove that $\text{rem}$ is P.R. it is sufficient to prove:

1. if then else is P.R.
2. $\ell < a$ is P.R.

If then else:

We know if $p \text{ \& \& } q$ is P.R. then

if $p$ then $q$ is also P.R.
Here result is either \( p \) and \( g \) or \( \neg p \) and \( h \)

So,

\[
P \cdot g + (1 - p) h
\]

if \( p \) is true \((p = 1)\): \(1 \cdot g + (1 - 1) h = g\)

\( p \) is false \((p = 0)\): \(0 \cdot g + (1 - 0) h = h\)

To prove that we need to prove add, subtract, mult, is also \( p \cdot r \)

**Addition** is \( p \cdot r \):

By def \( a + b = a + 1 + 1 + \ldots + 1 \)

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

Renaming \( f, m, n \),

\[
f (n, 0) = n, \quad f (n, 1) = g (n)
\]

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

Compare with general case of \( p \cdot r \)

\[
g = n^1
\]
\[
h = n^0 \cdot (n^2)
\]

\[
\text{add} = p \cdot r \cdot (n^1, n^0 \cdot (n^2)) \quad \varnothing \cdot \varnothing
\]
prev is P.R.

\[ \text{prev}(n) = \begin{cases} n - 1 & \text{if } n > 1 \\ 0 & \text{otherwise} \end{cases} \]

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

So, \( g = 0 \), \( h = \pi^2_1 \)

So, \( \text{prev} = PR(0, \pi^2_1) \)

Subtraction is P.R.: \( a - b = a_1 \ldots - 1 \)

\[ \text{sub}(a_0, 0) = a_0 \]
\[ \text{sub}(a_0, m+1) = \text{prev}(\text{sub}(a_0, m)) \]

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

\[ \text{sub} = PR(\pi^1_1, \text{prev}(\pi^3_3)) \]

Multiplication is P.R.

\[ a \times b = a + a + a \ldots + a \]

\[ \text{mult}(a_0, 0) = a_0 \]
\[ \text{mult}(a_0, m+1) = \text{mult}(a_0, m) + a \]

Renaming: \( \text{mult} = f \), \( a = n_1 \)
\[ f(n_1, 0) = n_1 = g(n_1) \]
\[ f(n_1, m+1) = f(n_1, m) + n_1 = h(n_1, m, f(n_1, m)) \]

General equ of P.R. compare

\[ g = \pi^1_1 \]
\[ h = \text{add}(\pi^2_3, \pi^3_3) \]
So \( \text{mult} = PR(\mathbb{R}, \text{add}(\mathbb{R}, \mathbb{R})) \)

\( \leq \) is \( PR \)

To prove this we need to prove
and \((\mathbb{R}, +)\), \( \text{not}(\cdot)\), \( =\), \( \leq\), \( \text{pos}(\cdot)\) all are \( PR \)

\( \text{not}(\cdot) \) is \( PR \):
\[
\begin{align*}
\text{not}(0) &= 1 \\
\text{not}(1) &= 0 \\
\text{not} \left( \frac{1}{n} \right) &= 1 - n = \omega(n) - n = \omega(\text{not}(n)) - n = \omega(0) - n = \omega(0) - 0 = \omega(0) = 1 \\
\end{align*}
\]
so \( \text{not} \) is \( PR \)

and is \( PR \)

\[
\begin{array}{ccc}
\frac{a}{0} & 1 & 1 \\
0 & 0 & 0 \\
1 & 0 & 1 \\
\end{array}
\]
and is multiplication

\( \text{or} \) is \( PR \)

\[
\begin{array}{ccc}
\frac{a}{1} & 1 & 1 \\
0 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]
on is addition

\( \text{pos}(0) \):
\[
\begin{align*}
\text{pos}(0) &= 0 \\
\text{pos}(m+1) &= 1 \\
\end{align*}
\]
so, \( q = 0 \)

\( \text{h} = v^0(0) \)

So \( \leq \) is \( PR \), hence we have proved all the symbol that we need to prove are \( PR \).

\( 0/b \) is \( PR \)
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.

A primitive recursive function is called primitive recursive if it can be obtained from \( 0, \text{or}, \text{and}, \text{result}, \text{and} \text{primitive recursion.}

**Integer division** can be written as:

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

To prove that integer division is PR, we must also prove that remainder is PR.

**Remainder** can be written as:

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

To prove that remainder is PR we have to prove that if-then-else \(< a\) are PR.

**If-then-else**: If \( p, q, h \) are PR, then if \( p \) then \( q \) else \( h \) is also PR.

Proof: Result: either \( p \) is true \( (p = 1) \), then \( q + 0 + h = q \)

If \( p \) is false \( (p = 0) \), then \( q + 1 + h = h \)

To prove if-then-else is PR, we have to prove that addition, multiplication, and subtraction are also PR.

**Addition**: \( a + b = a + b \times \) \( b \times \) times so \( \text{add}(a, 0) = a \\
\text{add}(a, m+1) = \text{add}(a, m) + 1\)

\( f(n, 0) = n \) \( + \) \( f(n, m+1) = f(n, m) + 1 \)

\( 0, q = 1 \), \( n = 0 \times 3 \)

\( \equiv \text{PR}(n, 1, 0 = 3) \)

**Multiplication**: \( a \times b = a + a \times b \times \) \( b \times \) times so \( \text{mult}(a, 0) = a \\
\text{mult}(a, m+1) = \text{mult}(a, m) + a \) \( \times \) \( b \times \) \( f(n, 0) = n \) \( + \) \( f(n, m+1) = f(n, m) + n \)

\( 0, q = 1 \), \( n = \text{add}(3, 3) \)

\( \equiv \text{PR}(n, 1, \text{add}(3, 3)) \)

file:///Q:/cs5315.18/test1.html

1/28/2018
\[
\text{prev: } \text{prev}(n) = \begin{cases} 
  n-1 & \text{if } n \geq 1 \\
  0 & \text{otherwise}
\end{cases}
\]
\[
\text{so, prev}(0) = 0 \quad \text{so, } g = 0 \quad h = \pi_1 \Rightarrow PR(0, 0, \pi_1)
\]

Subtraction: \( a \div b = a \div \underbrace{\ldots}_{\text{b times}} - 1 \)
\[
\text{so, } \text{sub}(a, b) = a.
\]
\[
\Rightarrow g = \pi_1, h = \text{prev}(\pi_3)
\]

\[
\text{sub}(a, m) = \text{prev}(\text{sub}(a, m)) \equiv PR(\pi_1, \pi_1, \text{prev}(\pi_3))
\]

To prove this we need to prove \& (and), \mid (not), \mid (or), equal to zero, =, \leq, \geq, etc.

\[
\text{not}(1) : \text{not}(0) = 1
\]
\[
\text{not}(1) = 0
\]
\[
\text{not}(n) = 1 \div n
\]
\[
= 0 \div n
\]

So, by definition, \text{not}(1) is P.R.

\[
\text{and (\&): } a \text{ or } b \text{ or } c
\]

\[
\Delta \& \text{ is multiplication, therefore P.R.}
\]

\[
\text{or (\mid): } a \text{ or } b \text{ or } c
\]

\[
\text{All } \mid B = \mid (\mid A \Delta \& \mid B), \text{ since } \Delta \& \text{ and } \mid \text{ are P.R., then } \mid \text{ is P.R.}
\]

\[
\text{equal to zero: } a \text{ equal to zero } = \begin{cases} 
  \text{true if } n = 0 \\
  \text{false otherwise}
\end{cases}
\]
\[
\text{eqt}0(0) = 1 \quad \text{eqt}0(1) = 1 \quad h = 0
\]
\[
\Rightarrow PR(0, 0, 0)
\]

\[
\leq: a \leq b \iff \text{eqt}0(a \div b)
\]
\[
= : \text{a} = \text{b} \iff a \leq b \Delta \& b \leq a
\]
\[
\geq: a \geq b \iff \text{not}(a \div b)
\]
\[
\leq: a \leq b \Delta \& \text{not}(a \div b)
\]

Therefore, they're all P.R., which means that integer division is P.R. as well.
A function is primitive recursive if it can be obtained from \( \emptyset, \delta, \text{ or } \Pi_i^n \), or composition:

- \( \emptyset \) is p.r.:
  \[
  \text{equalTo}\emptyset(n) = \begin{cases} \text{true, if } n = \emptyset \\ \emptyset, \text{ else} \end{cases}
  \]
  \[
  \text{equalTo}\emptyset(\emptyset) = 1 \\
  \text{equalTo}\emptyset(m+1) = \emptyset \\
  q = 6 \circ \emptyset \\
  h = \emptyset \\
  \end{align}
  \]
  \[
  \text{PR}(6 \circ \emptyset, \emptyset)
  \]

- \( \delta \) is the function:
  \[
  \delta(n) = n+1
  \]

- \( \Pi_i^n \) is:
  \[
  \Pi_i^n (n_1, \ldots, n_k) = n_i
  \]

- composition, or \( \circ \) is: \( f \circ g = f(g(n)) \)

The general form:

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

Def. see left

To prove division is p.r., we need to prove \( \text{div}(a, 0) = 0 \)
\( \text{div}(a, m) = \)
if \( \text{rem}(a, m) > \text{div}(a, m) + 1 \)
\( \text{div}(a, m) + 1 \)
else
\( \text{div}(a, m) \)

To prove remainder is p.r., we need to prove \( \text{rem}(a, 0) = 0 \)
\( \text{rem}(a, m+1) = \)
if \( \text{rem}(a, m+1) + 1 < a \)
\( \text{rem}(a, m) + 1 \)
else
\( \text{rem}(a, m) \)

0) prove if-else is p.r.

if \( p \), \( q \), \( g \) are p.r., then if \( p \) then \( f \) else \( q \) is p.r.

\( \text{do}: p * f + (1-p) * g \)
if \( p \) is true \( (p = 1) \): \( p * f + 0 * g = f \)
if \( p \) is false \( (p = 0) \): \( 0 * f + 1 * g = g \)

a) we need to prove: addition \( + \) is p.r.
\( a + b = a + \underbrace{1 + 1 + \ldots + 1}_{n} \)
\( \text{add}(a, 0) = a \)
\( \text{add}(a, m+1) = \text{add}(a, m) + 1 \)
\( \text{add} = \text{PR}(\Pi^{1}_{1}, \text{add}^{1+n}_{3}) \)

b) we need to prove: multiplication \( \times \) is p.r.
\( a \times b = a + a + \ldots + a \)
\( \text{mul}(a, 0) = a \)
\( \text{mul}(a, m+1) = \text{mul}(a, m) + a \)
\( \text{mul} = \text{PR}(\Pi^{1}_{1}, \text{add}(\Pi^{3}_{1}, \Pi^{3}_{3})) \)
c) to prove $\vdash$ is p.r.:  
   i. we need to prove $\text{prev is p.r.}$:  
      \[
      \text{prev}(n) = \begin{cases} \varnothing, & n = 1 \\ \varnothing, & \text{otherwise} \end{cases}
      \]
      \[
      \text{prev}(\varnothing) = \varnothing \quad q = \varnothing \\
      \text{prev}(m+1) = m \\
      h = \Pi_3^1
      \]
      \[
      \text{sub(a, 0)} = a \\
      \text{sub(a, m+1)} = \text{prev(sub(a, m))} \\
      q = \Pi_1 \\
      h = \text{prev}(\Pi_3^3)
      \]
      \[
      \text{sub} = \text{PR(\Pi_1, \text{prev}(\Pi_3^3))} 
      \]

2. $\vdash$ is p.r.:  
    a) $\&$ (and) is p.r.:  
      $\&$ is multiplication, $\vdash$ p.r., $\&$ and $\&$ are p.r.  

    b) $\neg$ (not) is p.r.:  
      \[
      \neg (\varnothing) = \varnothing \\
      \neg (1) = \varnothing \\
      \neg (n) = 1 - n \\
      = \Pi(n) - n, \quad \& \text{and } \& \text{ are p.r.} 
      \]

c) $\lor$ (or) is p.r.:  
      by de Morgan's law:  
      \[
      a \lor b = \neg (\neg a \land \neg b), \quad \text{all p.r.} 
      \]

d) equal-to $\varnothing$ is p.r. $\rightarrow$ see first page.  

e) $\leq$ (less than/equal-to) is p.r.:  
      \[
      a \leq b \iff \text{equalTo}(a - b), \quad \text{equalTo is p.r.} 
      \]

f) $=$ (equal) is p.r.:  
      \[
      a = b \iff a \leq b \land b \leq a, \quad \leq \text{ and } \& \text{ are p.r.} 
      \]

g) $\neq$ (not equal) is p.r.:  
      \[
      \neg (a = b), \quad \text{both are p.r.} 
      \]

h) $<$ (less-than) is p.r.:  
      \[
      a \leq b \land \neg \neg \neg (a = b) \rightarrow \leq, \quad \neg \text{(not), and } = \text{ are p.r.} 
      \]
Therefore, with these proofs, it is sufficient proof that \( \text{div}(a, b) \) is p.c.
6. Prove that the following function \( f(a, b) \) is \( \mu \)-recursive: \( f(a, b) = a \parallel b \) when \( a \) and \( b \) are both equal to either 0 or 1, and \( f(a, b) \) is undefined for other pairs \( (a, b) \).

\[
f(a, b) = \begin{cases} 
\text{all } !b, (a = \emptyset \parallel a = 1) \land (b = 0 \parallel b = 1) & \text{defined, else}\end{cases}
\]

`all !b` is recursive

we have 4 cases:

1. \( a = 0, b = 0, m = 1 \)
2. \( a = 0, b = 1, m = \emptyset \)
3. \( a = 1, b = 1, m = 1 \)
4. \( a = 1, b = \emptyset, m = 1 \)

\[
\mu m. (a = \emptyset \land b = \emptyset \land m = 1) \parallel (a = \emptyset \land b = 1 \land m = \emptyset) \parallel (a = 1 \land b = 1 \land m = 1) \parallel (a = 1 \land b = \emptyset \land m = 1))
\]
6. Prove that the following function \( f(a, b) \) is \( \mu \)-recursive: \( f(a, b) = a \parallel lb \) when \( a \) and \( b \) are both equal to either 0 or 1, and \( f(a, b) \) is undefined for other pairs \( (a, b) \).

\[
\begin{align*}
f(a, b) &= \begin{cases} 
1 & \text{if } a = 0 \land b = 0 \\
0 & \text{if } a = 0 \land b = 1 \\
1 & \text{if } a = 1 \land b = 0 \\
1 & \text{if } a = 1 \land b = 1 \\
\text{undefined, otherwise}
\end{cases}
\end{align*}
\]

\[
\mu n. \ (((a = 0) \land (b = 0)) \land (m = 1), \\
(a = 0) \land (b = 1) \land (m = 0), \\
(a = 1) \land (b = 0) \land (m = 1), \\
(a = 1) \land (b = 1) \land (m = 1))
\]
6. Prove that the following function \( f(a, b) \) is \( \mu \)-recursive: \( f(a, b) = a \parallel (b) \) when \( a \) and \( b \) are both equal to either 0 or 1, and \( f(a, b) \) is undefined for other pairs \( (a, b) \).

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

\[
f(0, 0) = 1 \quad \text{if } a = 0 \text{ and } b = 0
\]

\[
= \begin{cases} 1 & \text{if } a = 0 \text{ and } b = 0 \\ 0 & \text{if } a = 0 \text{ and } b = 1 \\ 1 & \text{if } a = 1 \text{ and } b = 0 \\ 1 & \text{if } a = 1 \text{ and } b = 1 \\ \end{cases}
\]

To express the given function using \( \mu \)-recursive, we need to find a relationship among \( \mu \) and \( m \) such that \( m \) is the smallest value that satisfies the properties of the given function.

Let's describe the given function

\[
\mu m \left( \begin{array}{c}
(a = 0, b = 0, m = 0) \\
(a = 0, b = 1, m = 0) \\
(a = 1, b = 0, m = 0) \\
(a = 1, b = 1, m = 0)
\end{array} \right)
\]
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 \( f(2, 5) \)?

\[
\begin{align*}
\text{condition: } & P(n, \ldots, n_k, m) < T: 1 \\
\therefore & \exists m. P(n, \ldots, n_k, m) \quad \text{smallest } m
\end{align*}
\]

```c
int m = 0;
while (! (m + a > b))
    m++;

f(a, b) = 4
```

- \( a = 2 \)
- \( b = 5 \)
- \( m = 3 + 2 = 4 \)
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 \(f(2, 5)\)?

\[ \text{While loop: This is like for loop but instead of fixed number of iteration the num of iteration is the smallest number at which some condition is satisfied} \]

\[ p(n, \ldots, n_k, m) \rightarrow \text{condition } (\text{true/false}) \]

\[ \text{smallest natural number } m \text{ for which the condition is true} \]

\[ \text{elm} = e(n, \ldots, n_k, m) \]

\[ \text{Smallest } m \text{ such that } \]

\[ e(n, \ldots, n_k, m) \text{ is satisfied} \]

μ-recursive: A μ-recursive function is any function that can be obtained from \(0, \omega, \omega_i\) by using \(\mu\), \(\rho\), \(\beta\), and \(\epsilon\) recursion.

\[ \text{While loop of ques.:} \]

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

\[ \text{Value: } f(2, 5) = 4 \]

\([2, 5, \emptyset, \emptyset, \emptyset, 9] \]

\(m\)
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 $0, \pi^k$, and $\sigma$, we also allow this auxiliary function $f(n)$ in our constructions? Let us call functions that can be obtained from $0, \pi^k, \sigma,$ 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.

Thm. not every computable function is $f$-primitive recursive ($f$-p.r.)

Proof. Let's introduce an auxiliary notion of a code of an $f$-p.r. function. This notion is defined as follows:

- By definition, our $f$-p.r. function is obtained from $\varnothing$, $\pi^k$, and $\varnothing(n)$ by using composition ($\circ$) and primitive recursion. Therefore, every function can be described by an expression containing $(\cdot), \varnothing, \pi^k$, and $\varnothing(n)$.
- We can use LaTeX to translate this expression into ASCII symbols
  - we replace $\varnothing$ with $\sigma$
  - we replace $\pi^k$ with $\pi_k$
  - we replace $\varnothing$ with $\sigma$

Then, we use ASCII code to replace each ASCII symbol with a sequence of $\circ$s and $1$s.

So the resulting binary sequence, we append a 1 in front, it represents the resulting binary sequence as a natural number. This number is called a code of an $f$-p.r. function.

Lemma. There exists an algorithm $A$, that, given a natural number $c$, checks whether $c$ is a code of an $f$-p.r. function. If yes, it returns an executable file $f_c$ for computing this function.
Let's define a function $f(c)$:

$$f(c) = \begin{cases} \sum \phi(c) + 1 & \text{if } c \text{ is a code of an f-p.r. function} \\ \emptyset & \text{otherwise} \end{cases}$$

Let's prove that $f$ is computable but not f-p.r.

1. We will show that $f$ is computable by showing how to compute $f(c)$
   a) given $c$, check whether $c$ is a code of an f-p.r. function. For this, we use algorithm A (whose existence is proved in Theorem).
      If $c$ is not a code, we return $\emptyset$.
   b) If $c$ is a code, we produce $\phi_c$ by using A.
   c) we apply $\phi_c$ to $c$
   d) we add $1 + \text{return the result}$

2. Now, let's prove $f$ is not f-p.r. by contradiction.
   So, let's assume that $f$ is f-p.r. This means that it has a code, $c_0$.
   a) this means that for every $n$, $\phi_{c_0}(n) = f(n)$
      particularly, for $n = c_0$, we get $\phi_{c_0}(c_0) = f(c_0)$
   b) On the other hand, by definition of $\phi(c)$,
      since $c_0$ is a code, $\phi_{c_0}(c_0) = \phi_{c_0}(c_0) + 1$
      $\phi_{c_0}(c_0) = \phi_{c_0}(c_0) + 1$
      $\emptyset \neq 1$
      $\Rightarrow$ a contradiction.

   Therefore, our assumption is false and not every computable function is
   f-p.r.
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, \( \pi^k \), and \( \sigma \), we also allow this auxiliary function \( f(n) \) in our constructions? Let us call functions that can be obtained from 0, \( \pi^k \), \( \sigma \), 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: A \( f \)-primitive recursive function (f.p.r.f.)
is a function that can be obtained from 0, \( \pi^k \), \( \sigma \), and \( f \) by using composition and primitive recursion.

Thus every f.p.r.f. can be described by an expression involving \( f \), \( \pi^k \), \( \sigma \), and \( f \).

We can use later to translate this expression into ASCII symbols:

```
expr = with \sigma
  \pi^k will \pi^k
  0 with \sigma
```

Then we can use this rule to replace each ASCII Symbols with 0s and 1s:

```
\sigma \pi^k
```

To the resulting long binary sequence we append:

```
0 0 0 1 0 1 0 1 1 1 0 1 0 0 1 0 1 1 1 0 1 1 0 0 1
```

in the form and interpret the resulting binary as a natural sum.

This number is called a code of the original f.p.r. function.
Now we are going to present a lemma which is useful to prove.

**Lemma:** There exist an algorithm $A$ that, given a natural number $c$, checks whether $c$ is a code of a f.p.r. function, and if yes, then returns a executable file $f_c$ for computing this function.

Let's describe a function $f(c) = \begin{cases} f_c(c) & \text{if } c \text{ is a code of a f.p.r. function} \\ 0 & \text{otherwise} \end{cases}$

Let's prove $f$ is computable but not f.p.r.

We will prove by contradiction that we will first prove $f$ is computable. If $f$ is computable, we will see how to compute it.
So, \( f \) is computable

Let us now prove \( f \) is not \( f \). By contradiction:

**Assuming** \( f \) is \( f \). We will show contradiction.

Since \( f \) is \( f \), it has a code, \( \text{co} \).

So by applying the algo \( A \) to \( \text{co} \),

we get an executable file \( f_{\text{co}} \) that computes \( f(\*) \).

So for input

\[ f_{\text{co}}(n) = f(n) \]

In particular, it is true for

\[ n = \text{co} \]

\[ f_{\text{co}}(\text{co}) = f(\text{co}) \]

By definition of \( f(\*) \)

\[ f(\text{co}) = f_{\text{co}}(\text{co}) + 1 \]

\[ 0 = 1 \] (contradiction)

So \( f \) is not a \( f \).

Therefore, \( f \) is computable but not \( f \). proved.