Theory of computation.

week 2

m, n \in \mathbb{N}

\text{mult} = 0

\text{for } (i = 0; i < m; i++)
\quad \text{mult} = \text{mult} + n
\text{mul} + (m, m)
\begin{cases}
\text{mult} + (m, 0) = 0 \\
\text{mult} + (m, n+1) = \text{mult} + (n, m) + n
\end{cases}

\begin{cases}
\text{f}(n_1, n_k, 0) = g(n_1, \ldots, n_k) \\
\text{f}(n_1, n_k, m+1) = h(n_1, \ldots, n_k, m, f(n_1, n_k, m+1))
\end{cases}

k = 1

\begin{align*}
\text{f}(m, 0) &= g(m) \\
\text{f}(m, m+1) &= h(m, m, f(m, m))
\end{align*}

\begin{cases}
\text{f}(n_1, 0) = 0 \\
\text{f}(n_1, m+1) = \text{f}(n_1, m) + n_1
\end{cases}

\begin{align*}
g(n_1) &= 0 \\
h(n_1, m_1, f) &= f + m_1 = \frac{n_1^3}{m_1} + \frac{n_1^3}{m_1}
\end{align*}

\begin{align*}
\text{mult} &= \text{PR}((0, \text{add}(\frac{n_1^3}{m_1}, \frac{n_1^3}{m_1})))
\end{align*}

\text{square}

\begin{align*}
0^2 &= 0 \\
(m+1)^2 &= m^2 + 2m + 1
\end{align*}

\begin{align*}
\text{square}(0) &= 0 \\
\text{square}(m+1) &= \text{square}(m^2) + 2m + 1
\end{align*}
\[ f(0) = 0 \quad g = 0 \]
\[ f(m+1) = f(m) + m - m + 1 \]

\[ n = \]

prove factorial is descriptive recursive
\[ \text{fact}(n) = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \]
\[ f = 1; \]
\[ \text{factorial}(n) \]
\[ \text{for } (i=1; i \leq n; i++) \{
\]
\[ \quad f(i) = f(i) \times i; \]
\[ \quad \]  \]
\[ \}

1. Write java code -> loop // General Description
2. 
3. Match with definition
\[ f(m, k) : f(m, n, 0) = g(n, \ldots, n) \]
\[ f(m, k) : f(m, n, m+1) = g(n, \ldots, n, m) \]

\[ \square \text{ Before Thursday} \]

1) \( a^n \) is PR
2) \( n^2 + n^2 + 5 \) is PR
3) \( n^3 - n^2 + 5 \) is PR
4) exclusive OR is PR
Prove that
\[ a - b = \begin{cases} 
  a - b & \text{if } a > b \\
  0 & \text{if } a \leq b 
\end{cases} \]

\[
\begin{align*}
\text{prev}(3) &= 2 \\
\text{prev}(2) &= 1 \\
\text{prev}(1) &= 0 \\
\text{prev}(0) &= 0
\end{align*}
\]

\[
\begin{align*}
\text{prev}(0) &= 0 \\
\text{prev}(m+1) &= 1 \\
&& \text{if } m + 1 \\
\end{align*}
\]

\[ \text{prev}(0, \pi_2^2) \]

\[ \forall \text{prev}(0, \pi_2^2) \Rightarrow a = 0 \]

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

\[ + \text{ minus } = a \]
\[ \text{for } (i = 1; i < b; i++) \{
\]
\[ \text{minus } = \text{ minus } - 1; \]
\[ \}
\]

\[ \text{minus}(0) = a \]
\[ \text{minus}(a, m+1) = h(f(a, m)) \]

\[ f(n, 0) = n \]
\[ f(n_1, m+1) = 0 \text{ if prev}(f(n_1, m+1)) \]

\[ h(n, m, f) \]

Prove
\[ q = n = n^3 \]

Conditions: if then

\[ \begin{cases} <, >, =, \leq, \geq \\ + \quad > 0 \quad \text{and not} \end{cases} \]

\[ \text{true} = 1 \quad \text{pos}(0) = 0 - q \]
\[ \text{false} = 0 \quad \text{pos}(m+1) = 1 - h \]

\[ \Rightarrow \text{PR}(q, h) = \text{PR}(0, 1) \]

\[ > 0 \quad \text{is PR} \]

\[ + \quad a \geq b \]
\[ \Rightarrow a - b > 0 \quad a - b \equiv \text{PR}(\text{pos}(a-b)) \]
\[ \begin{cases} \text{not}(0) = 1 \\ \text{not}(m+1) = 0 \end{cases} \]
\[ a \leq b \quad \text{not} \quad \text{pos}(a \geq b) \]
\[ \text{not} \quad \text{is PR} \]

\[ \begin{aligned} n > 0 & \quad \text{is PR} \\ a \geq b & \equiv \text{pos}(a-b) \\ \text{not} & \quad \text{is PR} \\ a > b & \equiv \text{not} \quad (b > a) \end{aligned} \]

\[ a = b \quad \Rightarrow \quad \text{not} \quad (a > b) \quad \text{and} \quad \text{not} \quad (a < b) \]
\[ a \land b \quad \Rightarrow \quad \text{not} \quad (\text{not}(a \land \text{not} b)) \]
Three classes of basic functions:

- 2 ways of building new functions from existing ones:
  - composition
  - projections

Projection function: \( p^k : \mathbb{N}^k \rightarrow \mathbb{N} \)

\[ p(x_1, x_2, \ldots, x_k) = x_i \]

- it goes from \( k \)-dimensional "space" into 1-dimensional "space"
- basic building blocks for PR functions.

If \( g_1, g_2, \ldots, g_m \) are functions \( \mathbb{N}^k \rightarrow \mathbb{N} \), and \( f \) is a function \( \mathbb{N}^m \rightarrow \mathbb{N} \),

\[ h : \mathbb{N}^k \rightarrow \mathbb{N} \text{ given by} \]

\[ h(x_1, x_2, \ldots, x_k) = f(g(x_1, x_2), \ldots, g(x_1, x_m)) \]

- \( h \) is formed from:

\[ h = \text{comp} [f, g_1, \ldots, g_m] \]

\[ h(x_1, x_2, x_3) = f(g(x_1, x_2), x_3) \]

Primitive recursion:

\( f : \mathbb{N}^k \rightarrow \mathbb{N} \)
\( g : \mathbb{N}^{k+2} \rightarrow \mathbb{N} \)

Then \( h : \mathbb{N}^{k+1} \rightarrow \mathbb{N} \) is said to be defined by PR form if \( f \) and \( g \) is:

\[ h(x, 0) = f(x) \]

\[ h(x, s(y)) = g(x, y, h(x, y)) \]

\[ h = \text{PR}(f, g) \]
Def:
- Any PR function can be obtained from 0, S, P by composition and PR.
- 0 is PR, S, P are so.

A polynomial is a function can be obtained from const or variable by using multiplication / addition.

\[ \text{Ex: } n^3 + n^2 + 5 \Rightarrow n^5: \text{result of } \text{mult} \quad \Rightarrow \text{PR} \quad \Rightarrow \]
\[ n^3 + n^2 + 5: \text{result of } + \quad \Rightarrow \text{PR} \]

Ans:
- Exclusive OR:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A XOR B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

+ Two ways:

1. \[ A \oplus B = (A \land \overline{B}) \lor (!A \land B) \]

2. \[ AB + \overline{AB} \]

\[ (!A \land \overline{B}) \lor (A \land B) \]

Logical notations:
- \(<, \leq, >, \geq, =, \neq, \land, \lor, \neg, \)
- if then statement

\[ (a > 0) \Rightarrow a = -a \]

\[ \text{if } (a > 0) \text{ then } a \text{ else } -a; \]

\[ h(n) = \text{if } p(n) \text{ then } s(n) \text{ else } g(n) \]

PR | PR | PR

We want to prove \( h(n) \) is PR.
let's make it natural.

either \( p(n) \) and \( f(n) \) is the value.
or \( \overline{p(n)} \) and \( g(n) \) is the value.

\[
\text{not } p = 1 - p
\]
\[
A \oplus \overline{B} = A \cdot \overline{B}
\]
\[
A \oplus B = \overline{A} \cdot B + A \cdot \overline{B}
\]

\[
\left\lfloor p(n) \cdot f(n) + (1 - p(n)) \cdot g(n) \right\rfloor
\]

\[
\text{if } p(n) \text{ is true } \implies p(n) = 1 : 1 \cdot f(n) + 0 \cdot g(n) = f(n)
\]
\[
\text{false } \implies p(n) = 0 : 0 \cdot f(n) + 1 \cdot g(n) = g(n)
\]

By Tuesday

1) Prove that this is PR.

if \( \overline{p(n)} \) then \( \overline{f(n)} \)
else if \( \overline{g(n)} \) then \( \overline{g(n)} \)
else \( h(n) \)

4) Remainder:

\( \text{rem}(n) = 0 \)

\( \text{rem}(n) = \begin{cases} 
(m \text{ rem } n + 1) & \text{if } (m \text{ rem } n + 1) < n \\
0 & \text{otherwise}
\end{cases} \)

\[ 0 \% 5 = 0 \rightarrow \text{prev. remainder} \]
\[ 0 + 1 \% 5 = 0 + 1 \]
\[ 2 \quad 4 + 1 \]
\[ 3 \quad 2 + 1 \]
\[ \text{rem}(n, m+1) = \begin{cases} \text{rem}(n, m) + 1 & \text{if } (\text{rem}(n, m) + 1 < n) \\ \text{rem}(n, m) + 1 & \text{else} \quad 0 \end{cases} \]

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

Proof:

\[ \text{if } (f+1 < n) \text{ then } f+1 \]
\[ \text{else } 0 \]

\[ d \]

Division:

\[ 0 / n = 0 \]

\[ m + 1 / n = \begin{cases} \text{if } (m + 1) / n = 0 \text{ then } (m + 1) / n + 1 \\ \text{else } m / n \end{cases} \]

\[ 13 / 3 = 4 \]
\[ 15 / 3 = 5 \]
\[ 17 / 3 = 5 \]
\[ 19 / 3 = 4 \]
\[ 16 / 3 = 5 \]
\[ 18 / 3 = 6 \]

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

\[ f(n, m+1) = \begin{cases} \text{if } (m + 1) / n = 0 \text{ then } f(n, m) + 1 \\ \text{else } f(n, m) \end{cases} \]

\[ h(n, m, f(n, m)) \]

\[ 2 \]

Is every computable function P recursive? 2

2 proofs:

< more detailed, easier proof
< less detailed, more natural

Theorem: There exists a computable function which is not P.
Way 1: \( \text{PR \& \& (n)} \Rightarrow \text{PR (0, 0, \sigma, \Pi)} \)

\[ \text{LaTeX: PR (0, \lor \text{some} \lor \text{ore, } \lor \Pi \rightarrow \Pi)} \]

\[ 0 \rightarrow 1 \]

\[ C = \text{code of a PR } \#(n) \]

+ Computable but not PR.

\[ f(n) = \begin{cases} 1 & \text{if } \#(n) \text{ is a valid code} \\ f(n) + 1 & \text{else} \end{cases} \]

Statement: \( f \) is not PR.

Proof by contradiction:

Let \( f \) be PR.

Then it has a code \( e \).

\[ f(n) = \#(n) \Rightarrow f(e) = f_{e}(e) \]

by def. since \( e \) is a valid code.

Then apply \( f_{c}(c) = f_{e}(c) + 1 \)

\[ \Rightarrow f(c) = f_{e}(c) + 1 \]

\[ \Rightarrow 0 = 1 \]

contra.
Theorem: Not every computable function is PR.

We will construct a function \( f(n) \) which is
- computable
- not PR.

1st part: Codes of PR \( f \)s (PR functions).

Goal: assign to every PR \( f \) a number.

We start with an expression \( PR(\Pi^0_1 \Delta \sigma^0_1 \circ 0) \).

Strings:

\( PR(\Pi^0_1 \Delta \sigma^0_1 \circ 0) \) → \( ASCI \)

The strings of binary 0, 1: 0110...111

Lucky put 1 in front.

\( 10110 ... 1 ... 1 \) ...

(in order to differentiate the code)

\( 101 = 3 \)
\( 1001 = 5 \)
\( 11001 = 9 \)

\( \vdots \) code of a PR \( f \).

Statement: There is an algorithm that gives a natural number \( c \) that checks whether \( c \) is a valid code of a PR \( f \).

If it's produces a valid code \( f \) for computing the corresponding PR \( f \).
Idea:
1. Strip off the \( \oplus \) in front of every piece of code.
2. Check if all combinations are \( \setminus \) symbols.
3. Let \( Y \) compute checks if the syntax is correct.
4. Check syntax.

\[
\text{for } i = 0, i < n, i++ \text{ do } h = g_i
\]

\[
\text{for } i = 0, i < n, i++ \text{ do } h = f_i
\]

* **n**:

```
if \( n \) is a valid code of a LR fn.

return 0
```

```
apply \( f_n \) to \( n \)

return \( f_n(n) + 1 \)
```

If this shows \( f \) is computable.
\[ f(n) = \begin{cases} fn(n) + 1 & \text{if } n \text{ is a valid code of a P.E.} \\ 0 & \text{otherwise} \end{cases} \]

We assume that \( f \) is P.E.

Then \( f \) is represented by some exp. that exp has a code \( c \).

\[ f_c(x) = f(x) \]

Since \( c \) is valid code, by definition of \( f \), we have

\[ f(c) = f_c(c) + 1 \]

In particular for \( x = c \)

\[ f(c) = f_c(c) \]

\[ \Rightarrow f_c(c) + 1 = f(c) \]

\( \Rightarrow 1 = 0 \)

(incorrect)

\[ \Rightarrow \text{by contradiction we have shown that } f \text{ is not P.E.} \]

---

Georg Cantor
1845 - 1918

Kantor's proof: we only care about the existence of P.E. proof: we also care about computability.

**Diagonalization**

Result: \( f \) is computable for which is not P.E.

1st proof: \( f \) is meaningless.

2nd proof: \( f \) is meaningful but the proof is more difficult.
Reminder:
we started with 0
\[ a + b = a + 1 + 1 + 1 + 1 \]
\[ \text{b times} \]
\[ a + a + a + a \]
\[ \text{b times} \]
\[ a \times a \times a \times a \]
\[ \text{b times} \]
level 0: \[ f_0(n, m) = n + 1 \]
level 1: \[ f_0(n, f_0(n, m)) \]

\[ f_1(n, 0) = 0 \]
\[ f_1(n, m + 1) = f_0(f_1(n, m), f_1(n, m)) \]

\[ f_2(n, m + 1) = f_1(f_2(n, m), n) \]

\[ f_3(n, m + 1) = f_2(f_3(n, m), n) \]

\[ \vdots \]

\[ f_{k+1}(n, m + 1) = f_k(f_{k+1}(n, m), n) \]

\[ \Rightarrow \text{Archimedes.} \]

\[ a \times b \times \text{b times} = a \]

\[ A(n) = f_n(n, n) \]

Ackermann's function.
Ackermann

\[ A(0) = \delta_0(0,0) = 1 \]
\[ A(1) = \delta_1(1,1) = 1 + 1 = 2 \]
\[ A(2) = \delta_2(2,2) = 2 \times 2 = 4 \]
\[ A(3) = \delta_3(3,3) = 3^3 = 27 \]
\[ A(4) = \delta_4(4,4) = 4^{4^4} = 4 \uparrow\uparrow\uparrow \uparrow 4 \approx 10^{10^{10^{1.3}}} \]

\[ \text{used: } 6.7 \times 10^{360} \]

\[ \text{By Tuesday: Ask Dr. VK -> theory project topic} \]

Prove that \( A(n) \) is not PR

\[ A(n) = \delta_n(n,n) \]

\[ \text{if } \leq k \text{ for loops } \Rightarrow f \leq A_k(n,n) \]

\[ \Rightarrow \text{ Don't use for loops to prove.} \]
\[ 0 \rightarrow \delta(0), \delta(\delta(0)) \]
\[ \rightarrow \delta(\delta(\delta(0))), \delta(\delta(\delta(\delta(0)))) \]
\[ n, n+1, n+2 \]

\[ \exists \text{ list: } \]
\[ f(n) \leq n+c \]
\[ f(n_1, n_2, \ldots, n_k) \leq n_1 + c \]

\[ f(n, 0) = g(n) \]
\[ f(n, m+1) = h(n, m, f(n, m)) \]

\[ f(0) = g \]
\[ f(m+1) = \max (m, f(m)) \leq \max (m, f(m)) + e \]
\[ (1, c) \text{ constant} \]
\[ f(0) = 0 \]

\[ f(n+1) = h(n, f(n)) + c \leq \max(\bar{g}, c) + c \leq \bar{g} + c \]

\[ f(1+1) = h(n, f(1)) + c \leq \max(\bar{g}, c) \leq \bar{g} + 2c \]

For no for-loops:

\[ f(n) \leq n + 1 + \ldots + 1 \]

const times

For one for-loop:

\[ f(n) \leq n + \ldots + n + \bar{g} \]

const times

\[ i^{\text{th}} \]

\[ n+1 \]

\[ n+2n \]

\[ n^2 \]

\[ n^n \]

1. Turn in project title proposal \( \rightarrow \) PR

2. Prove that not every computable fn is \( \rightarrow \) PR

First hypothesis: Every computable fn is \( \rightarrow \) PR.

Fact: There exists a computable fn is NOT \( \rightarrow \) PR.

\( \exists \) : Ackermann fn, \( \#(n) \).

Natural idea:

Add \( \#(n) \) to basic funs, \( 0, 1, \overline{\text{c}}, \text{ and } \#(n) \).
If then \( \forall \) using while loop.
For \( \forall \) we need to formulate while loop.

Boolean \( \forall \) Big diff: \# of iterations is not explicitly known, determined by a condition.

\[
\text{while} \left( \, \text{??} \, \right) \quad \{ \ldots \} \quad \text{\# of iterations = smallest } \mu \text{ for which } f(n, m) \text{ is true}
\]

\[
\mu - \text{reconstruction:} \quad \mu(n, m) = \mu(m, f(n_1, n_2, \ldots n_k, m))
\]

\( \mu = "\mu\mu". \)

\( \rightarrow \) A \( \mu \)-recursive fn is any fn which can be obtained from \( 0, \sigma, \Pi_k, \) by \( \sigma, \Pi_k \) and \( \mu \)-reconstruction:

\[
f(n, m) = \mu m. \ f(n, m) \quad m = 0 \quad \text{while} \left( \, \text{??} \, \right)
\]

\[
a - b = c \quad \text{st.} \quad (\text{such that})
\]

\[
c + b = a
\]

\[
m + s
\]

\[
\mu(c + b = a)
\]

\[
\text{smallest } c = \mu c
\]

\[
\Rightarrow \mu c (c + b > a) : a \leq b \quad \mu c = 0
\]

\[
\mu c (c + b > a) : a > b
\]

3. Describe 
\( a / b. \)

4. Describe \( \lim. \) 
\( f(n) = \begin{cases} 1 & \text{if } n = 2 \\ 2 & \text{if } n = 1 \\ \text{undef, otherwise} \end{cases} \)
\( f(n) = \begin{cases} 
\mu m ((0 \land m \neq 0) \lor (n = 0 \land m = 0)) \\
\text{while } (!((0 = 2)) \\
\text{if } m = 1 \text{ then } r \text{ if } \text{undefined otherwise} \\
\text{if } n = 0 \text{ then return } 0 \\
\text{if } n > 0 \text{ then undefined} 
\end{cases} \)

This fn. is never defined.

\( f(n) = \begin{cases} 
0 & \text{if } n = 0 \\
\text{otherwise} 
\end{cases} \)

\( f(n) = \begin{cases} 
1 & \text{if } n = 2 \\
2 & \text{if } n = 3 \\
\text{undefined otherwise} 
\end{cases} \)

\( \mu m \left[ (n = 2) \land (m = 1) \lor (n = 3) \land (m = 2) \right] \)

\( m \): smallest number makes fn. satisfiable.

\( \mu \)-recursive functions may be undefined.

Totally recursive = \( \mu \)-recursive and everywhere defined.

every Java program can be described as a \( \mu \)-recursive fn.

Important:

In 1930s, 2 different def's appeared:
- Recursion (church) based on a prog. lang.
Reproduce = write detailed proof.

$$f_n(n) + 1$$

+ proof has to be completely clear.

Computable = $\mu$-recursive.

big problems with while-loops: go into infinite loops

// halt, queue < stop, stand in line

Theory: no algorithm is possible that given a problem program $p$ and data $d$ would to check whether $p$ halts on $d$ (data) or not.

$\text{halte}\text{checker}(p, d) = \begin{cases} \text{yes} & \text{if } p \text{ halts on } d. \\ \text{no} & \text{otherwise.} \end{cases}$

Reduction to a contradiction: let's assume that haltechecker exists.

$$f_n(n) = \begin{cases} f_n(n) + 1 & \text{if } n \text{ is a valid C code program} \\ 0 & \text{otherwise} \end{cases}$$
- with \( x = c \) \( \rightarrow f_c(x) = f(x) \rightarrow f(x) = f(c) \) \( \text{(2)} \)

- By definition of \( f \), since \( c \) is a valid code, \( f_c \)
  always halts
  \[ f_c(c) = f(c) + 1 \] \( \text{(1)} \)

\( \text{(1)}, \text{(2)} \rightarrow \text{contradiction} \)

- Assume: If \( f \) is computable, then there is a code \( c \) for (by some Java program, let's denote by \( \text{\( f(c) \)}} \)\)
  the code of [Java program, particular \( x = c \)]\n  \[ f_c(x) = f(x) \]

1) Redo proof.
2) Reproduce halting proof.

A program

\begin{verbatim}
1) does it halt?
2) if it halts, does it produce correct results?

we want \( \forall x (f(x) = 0) \).

public static int zero (int x) {
    return 0;
}
\end{verbatim}

Def: A zero checker is a program \( p \) that given \( p \) that always halt, checks whether \( \forall x (p(x) = 0) \).

\[ \text{zero checker}(p) = \begin{cases} 
    \text{yes} & \text{if} \; \forall x (p(x) = 0) \\
    \text{no} & \text{otherwise} 
\end{cases} \]

Theorem: Zero checkers do not exist.
1. Assume zero checkers exist.
2. Build halt checkers on top of zero checkers $\rightarrow$ haltcheckers exist!
3. However, halt checkers don't exist $\rightarrow$ so neither do zero checkers!

We assume that zero checkers exist, and we will conclude that a halt checker exists.

$$\text{halt checker } (p, d) = \begin{cases} 1 & \text{if } p \text{ halts on } d \\ 0 & \text{if } p \text{ does not halt on } d \end{cases}$$

we can design:

$$\text{halt } (p, d, t) = \begin{cases} 1 & \text{if } p \text{ halts on } d \text{ by time } (t) \\ 0 & \text{otherwise} \end{cases}$$

$p$ halts on $d$ $\iff$ there exists $a +$ such that $p$ halts on $d$ and $p$ does not $= \forall t \ p$ does not halt on $d$.

we define:

$$f_{p,d}(t) = \text{halt}(p, d, t)$$

Zero checker: $f_{p,d} = \emptyset$
- Apply zero checker to $f_{p,d}$.

$$\text{halt checker } = \neg \text{ Zero checker } \left( f_{p,d} \right)$$

3. Reproduce zero checker proof.

Square checker $(p) = \begin{cases} 1 & \text{if } \forall x \ p(x) = x^2 \\ 0 & \text{otherwise} \end{cases}$

Theorem: square checker is uncomputable.
1. Prove that a square checker exists and we will build a zero checker.

\[ q \rightarrow 1. \forall x. q(x) = x^2 \]

Zero checker: 
\[
\text{zero checker}(p) \equiv \text{square checker}(p(x^2))
\]
\[
\forall x. p(x) + x^2 = x^2
\]
\[
\forall x. (p(x)) = 0
\]

4. Prove no \(\mu\) checker does not exist.

\(\mu\) reason.

Open problem:

- if \(n\) is even \(n = n/2\)
- if \(n\) is odd \(n = 3n + 1\)

\(a/b = \mu c (c(b+1) > a)\)

Church \(\mu\)-recursive:
- \(\varepsilon, \xi, \Pi^k\), \(\varepsilon, \xi\) \(\mu\)-recursive

Turing Machines:

- can we compute every \(\mu\)-recursive fun on a Turing machine?
Binary code: 0 1 1
2 1
3 1

Start: # 1 ... 1 #

Result: # 1 ... 1 #

f(n)

Function: return 0;

Function: Add 1:

(start, #) → (R, reading)
(reading, 1) → (R, reading)
(reading, #) → (L, erasing)
(erasing, 1) → (L, #, erasing) replace 1 by #
(erasing, #) → halt.

Function: Add 1:

(start, #) → (R, reading)
(reading, 1) → (R, reading)
(reading, #) → (L, 1, back)
(back, 1) → (L, back)
(back, #) → halt.
1. Design a TM that computes $f(n) = n + 2$.

   // Use the 'state' wisely.

   function projection $\Pi^2_3((n_1, n_2)) \rightarrow n_1$

   

   

   

   

   

   

   (start, #) $\rightarrow$ (R, in1 right)

   (in1 right, #) $\rightarrow$ (R, in1 right)

   (in1 right, 1) $\rightarrow$ (R, in2 right)

   (in2 right, #) $\rightarrow$ (L, #, erase2no)

   replace

   (erase2no, 1) $\rightarrow$ (L, erase2no)

   (erase2no, #) $\rightarrow$ (L, in1 left)

   (in1 left, 1) $\rightarrow$ (L, in1 left)

   (in1 left, #) $\rightarrow$ inhalt

2. TM: $\Pi^3_4$  $f(n_1, n_2, n_3) = n_1$

3. Extra credit: $\Pi^2_2$  $f(n_1, n_2) = n_2$