**Primitive Recursive Functions**

**Base Functions are PRFs**

- \( c_a (\overline{x}) = a \)  
  - **Constants**

- \( I^n_L (x_1, \ldots, x_n) = x_i \)  
  - **Projections (Identity)**

- \( S(x) = x + 1 \)  
  - **Successor (Increment)**

**Build More Via**

\[ F(\overline{x}) = H(G_1(\overline{x}), \ldots, G_k(\overline{x})) \]

- **Composition**

- \( F(\overline{x}, 0) = c_{\overline{x}} (\overline{x}) \)

- \( F(\overline{x}, y+1) = H(\overline{x}, y, F(\overline{x}, y)) \)

- **Induction (Primitive Recursion)**
Building New PRFs

**Addition: Formal**

\[ + (x, 0) = I_1'(x) \]

\[ + (x, y+1) = S \left( \Gamma_3 \left( x, y, \Gamma_3 \left( x, y \right) \right) \right) \]

**Addition: Less Formal**

\[ x + 0 = x \]

\[ x + (y+1) = (x+y) + 1 \]

**Multiplication: Formal**

\[ \star (x, 0) = C_0(x) \]

\[ \star (x, y+1) = H \left( x, y, \star (x, y) \right) \]

\[ H(x, y, z) = + \left( I_1(x, y, 3) \Gamma_3(x, y, z) \right) \]

**Multiplication: Less Formal**

\[ x \star 0 = 0 \]

\[ x \star (y+1) = x \star y + x \]
More Basic Arithmetic

Predecessor: (Limited)

\[ 0 - 1 = 0 \]
\[ (x+1) - 1 = x \]

Subtraction: (Limited)

\[ x \div 0 = x \]
\[ x \div (y+1) = (x-y) - 1 \]

Factorial:

\[ 0! = 1 \]
\[ (x+1)! = x! \times (x+1) \]
RELATIONS

IS ZERO:

\[ 0 \equiv 0 = 1 \]
\[ (x+1) \equiv 0 = 0 \]

EQUALITY AND ONE OTHER:

\[ x \equiv y = ((x-y) + (y-x)) \equiv 0 \]
\[ x \equiv y = (x-y) \equiv 0 \]

BOOLEANS

NEGATION

\[ \neg x \equiv x \equiv 0 \]

AND

\[ x \& y = (x \cdot y) \]

OR

\[ x \| y = \neg((x \equiv 0) \& (y \equiv 0)) \]
**Bounded Minimization**

\[ f(x) = \mu z (z \leq x) \left\lceil P(z) \right\rceil \text{ if } \exists \text{ such } z \]

\[ = x + 1 \text{ otherwise} \]

\[ f(0) = 1 - P(0) \]

\[ f(x+1) = \left[ f(x) \ast (f(x) \leq x) \right] \]

\[ + ((x+2-P(x+1)) \ast \sim (f(x) \leq x)) \]

\[ f(x) = \mu z (z < x) \left\lceil P(z) \right\rceil \text{ if } \exists \text{ such } z \]

\[ = x \text{ otherwise} \]

\[ f(0) = 0 \]

\[ f(x+1) = \mu z (z \leq x) \left\lceil P(z) \right\rceil \]
**Division & Divisibility**

**Division:**
\[ \frac{x}{0} = 0 \quad \text{need a value} \]
\[ \frac{x}{(y+1)} = \mu z [(z < x) [z + 1] (y + 1) > x] \]

**Divisibility**
\[ x \mid y = (\exists y/x) x = y \]

**Exponents**
\[ x^0 = 1 \]
\[ x^{y+1} = x \cdot (x^y) \]

**Primality**
\[ \text{FirstFactor}(x) = \mu z [(2 \leq z \leq x) [z | x]] \]
\[ \text{FirstFactor}(x) = 0 \quad \text{if none} \]
\[ \text{IsPrime}(x) = \text{FirstFactor}(x) = x \in \mathbb{N} (x > 1) \]
\[ \text{Prime}(0) = 2 \]
\[ \text{Prime}(x+1) = \mu z [\text{Prime}(x) \times z \leq \text{Prime}(x)! + 1][\text{IsPrime}(z)] \]

Abbreviate \( \text{Prime}(2) \) as \( \pi \)
(note: stack analogy) \[
\langle z', y' \rangle = \langle x', y' \rangle
\]
encode n-tuples

- These are very useful and can be extended to

\[
\zeta \prod_1 (1 - 1) (z^2) = \zeta \prod_1 (1 + z)
\]

(0, \exp(z) + 1, 0)

with inverses

- \[
\pair(x, y) = \langle x', y' \rangle = \langle z^2, x \rangle
\]

Pairing Functions
the problem of mapping the pairing function, \( \langle x, y \rangle \), which implies the modification of the pairing function, \( \langle x, y \rangle + 1 \), which implies the following.

We will look at two cases, where we use the following:

**Approach 1:**

is \( 1 \cdot 1 \) onto the natural numbers.

Prove that the pairing function \( \langle x, y \rangle = 2^x (2y + 1) - 1 \):

**Pairing Function** is \( 1 \cdot 1 \) onto.
the odd natural numbers. produced by $2^{x}(2^y+1)$ when $x > 0$. Thus, $x < 0$, $y < 0$. $T$ hence, $x = 1$. $y > 1$ onto each such odd number and no odd number is moreover, a particular value of $y$ is uniquely associated number is by definition one of the form $2^y + 1$, where $y \geq 0$.

For $x = 0$, $y > 0$, $2^0(2^y+1) = 2^y+1$. But every odd number is

\[ \text{Case } 1: \quad (x=0) \]
\[ \text{as was desired.} \]

The above shows that \( \forall y \in \mathbb{Z}^+, \exists x \in \mathbb{Z} \) such that \( x + y = 1 \) if \( y > 0 \) is odd and \( x - y = 1 \) if \( y > 0 \) is even.

\[ \exists y \in \mathbb{Z}^+, \forall x \in \mathbb{Z} \] when \( x > 0 \).

\[ \downarrow \text{onto \ the \ even \ natural \ numbers, \ when } x > 0. \]

Thus, \( x + y + 1 \) is \( 1 \) if \( y > 0 \) is odd and \( x - y + 1 \) is \( 1 \) if \( y > 0 \) is even. Moreover, \( x \) must be even, since it has a factor of 2, and hence \( \exists x \in \mathbb{Z}^+, \exists y \in \mathbb{Z}^+ \) such that \( x + y = 1 \) if \( y > 0 \) is odd and \( x - y = 1 \) if \( y > 0 \) is even.

\[ \text{Case 2: } \]

\[ (x > 0) \]
μ Recursive

4th Model

A Simple Extension to Primitive Recursive
least value such that:

iterate the minimization operator (\(\mu\)), read "the

The class of recursive functions adds one more

primitive recursive functions.

primitive recursive functions.

that cannot be represented by the class of

There are algorithms like Ackermann's function

limitation.

since the only iterator is bounded. That's a clear

All primitive recursive functions are algorithms

\(\mu\) Recursive Concepts
of atoms in our universe.

For any value of n that can be written using the number
exponentiation, \( a(n) = A_{-1}(n) \) grows so slowly that it is less
a super exponentional number involving six levels of
union/find algorithm. Note: A(4,4) is

The inverse of Ackermann's function is important to analyze
implementation.

Ackermann's function grows too fast to have a for-loop
primitive recursive function.

Witness an Ackermann observed in 1928 that this is not a

\[
\begin{align*}
\text{A}(1, i) &= \text{A}(i - 1, 1) \text{ for } i > 1, \\
\text{A}(1, 1) &= \text{A}(0, 2) \text{ for } i > 2, \\
\text{A}(1, 0) &= 2 \text{ for } i > 1.
\end{align*}
\]

Ackermann's Function
You should have learned that in CS2
- How do we represent the classes?
  - Can see if \( x \equiv y \) by seeing if \( \text{Find}(x) = \text{Find}(y) \)
- \( \text{Find}(x) \) returns the canonical element of \([x]\) containing \( x \) \( (\forall y) ([x] \subseteq [y] \) with that containing \( y \)
- \( \text{Union}(x, y) \) merges the class containing equivalence classes

\text{Union}/\text{Find}
condition.

is recursive can be used as the stopping
testing for one. In fact any predicate that
We also allow other predicates besides

\( F(x_1, \ldots, x_n) = \forall y \, (G(y, x_1, \ldots, x_n) \Rightarrow \neg y) \)

so is \( F \), where

If \( G \) is already known to be recursive, then

Minimization:

The \( n \) Operator
Primitive Recursive Functions

Base Functions are PRFs

\[ C_a(x) = a \]
\[ \pi^n(x_1, \ldots, x_n) = x_i \]
\[ S(x) = x + 1 \]

Build more via

\[ F(x) = H(G_1(x), \ldots, G_k(x)) \]

Composition

\[ F(x, 0) = C_x(x) \]
\[ F(x, y + 1) = H(x, y, F(x, y)) \]

Induction (Primitive Recursion)
Building New PRFs

Addition: Formal

\[ + (x, 0) = I'_1(x) \]
\[ + (x, y+1) = S \left( \underbrace{I^3_3(x, y, 3)}_{\text{Composition}} + (x, y) \right) \]

Addition: Less Formal

\[ x + 0 = x \]
\[ x + (y+1) = (x + y) + 1 \]

Multiplication: Formal

\[ \ast (x, 0) = C_0(x) \]
\[ \ast (x, y+1) = H(x, y, \ast (x, y)) \]
\[ H(x, y, 3) = + \left( I'_1(x, y, 3) I^3_3(x, y, 3) \right) \]

Multiplication: Less Formal

\[ x \ast 0 = 0 \]
\[ x \ast (y+1) = x \ast y + x \]
More Basic Arithmetic

Predecessor: (Limited)

\[ 0 - 1 = 0 \]
\[ (x+1) - 1 = x \]

Subtraction: (Limited)

\[ x \div 0 = x \]
\[ x \div (y+1) = (x-y) - 1 \]

Factorial:

\[ 0! = 1 \]
\[ (x+1)! = x! \times (x+1) \]
RELATIONS

ZERO:

\[ 0 \oplus 0 = 1 \]
\[ (x+1) \oplus 0 = 0 \]

EQUALITY AND ONE OTHER:

\[ x \oplus y = (x-y) + (y-x) = 0 \]
\[ x \Delta y = (x-y) \oplus 0 \]

BOOLEANS

NEGATION

\[ \neg x = x \oplus 0 \]

AND

\[ x \& y = (x \oplus y) \]

OR

\[ x \| y = \neg ((x = 0) \& \neg (y = 0)) \]
Bounded Minimization

\[ f(x) = \begin{cases} 
\mu z (z \leq x) \left[ P(z) \right] & \text{if } \exists \text{ such } z \\
= x + 1 & \text{otherwise}
\end{cases} \]

\[ f(0) = 1 - P(0) \]
\[ f(x+1) = \left( f(x) \ast (f(x) \leq x) \right) \\
+ ((x + 2 - P(x+1)) \ast \sim (f(x) \leq x)) \]

\[ f(x) = \begin{cases} 
\mu z (z < x) \left[ P(z) \right] & \text{if } \exists \text{ such } z \\
= x & \text{otherwise}
\end{cases} \]

\[ f(0) = 0 \]
\[ f(x+1) = \mu z (z \leq x) \left[ P(z) \right] \]
DIVISION & DIVISIBILITY

\[ x \div y = \mu z \ (z < x) \ [z + 1 \cdot y > x] \]

DIVISIBILITY
\[ x \mid y \iff ((x \div y) \cdot x) = y \]

EXPORENTS

\[ x^{\phi} = 1 \]
\[ x^{a + b} = x^a \cdot x^b \]

PRIMALITY

\[ \text{FIRSTFACTOR}(x) = \mu z \ (2 \leq z \leq x) \ [z \mid x] \]
\[ x \mid y \iff x \epsilon \mathbb{R}(x \geq 1) \]

PRIME \( (0) = 2 \)

PRIME \( (x + 1) = \mu z \ (\text{PRIME}(x) \cdot x \leq \text{PRIME}(x)! + 1) [z \mid \text{PRIME}(z)] \]

ABBREVIATE PRIME \( (2) \) AS \( P \).
\( \langle x', y' \rangle = \langle z', y' \rangle = \langle z, y' \rangle \)

encode n-tuples

These are very useful and can be extended to

\[ z \parallel (1 - 1) + z \parallel (1 + z) )) = z <z> \]

With inverses

\[ (0 + z)(x + 1, 0) = 1 <z> \]

Pairing Functions

(pair(x, y)) = (2x (2y + 1) - 1)
the problem of mapping the pairing function \( \langle x, y \rangle + 1 \), which implies modification of the pairing function \( \langle x, y \rangle \), which we use the following.

We will look at two cases, where we use the following.

**Approach 1:**

Prove that the pairing function \( \langle x, y \rangle = 2^x (2y + 1) - 1 \) is \( 1 - 1 \) onto the natural numbers.

**Pairing Function is 1-1 Onto**
the odd natural numbers. 
produced by $2^x(2y+1)$ when $x > 0$. Thus, $y > 0$ if $x > 0$. 
Moreover, a particular value of $y$ is uniquely associated with each such odd number and no odd number is 

For $x = 0$, $y > 0$, $2^0(2y+1) = 2y+1$. But every odd number is by definition one of the form $2y+1$, where $y \geq 0$.

Case 1: $x = 0$.
The above shows that \( x > 0 \) is \(-1\) onto \( \mathbb{Z} \), but then \( x' < y \) \(+1\) is \(-1\) onto \( \mathbb{Z} \). Hence, \( x', y + 1 \) is \(-1\) onto \( \mathbb{Z} \). Thus, \( x', y + 1 \) is \(-1\) onto \( \mathbb{Z} \). However, every even number except zero is of the form \( 2 \times 2 \), where \( x < 0 \), \( z \) is an odd number and this pair \( x, z \) is unique. Therefore, \( x', y + 1 \) is \(-1\) onto \( \mathbb{Z} \). Moreover, from elementary number theory, we know that every even number except zero is of the form \( 2 \times 2 \), where \( x ) + 1 \) is also even. Moreover, from elementary number theory, we know that in case \( 1 \), \( z \) must be even, since \( x' \) has a factor of \( 2 \) and hence is uniquely associated with one based on the value of \( y \) (we saw for \( x < 0 \), \( x', y + 1 = 2 \times (2y + 1) \), where \( 2y + 1 \) ranges over all odd numbers.

Case 2:

\[ x > 0 \]
Recursive
A Simple Extension to Primitive
4th Model

Recursive
least value such that

The class of recursive functions adds one more

primitive recursive functions.

primitive recursive functions.

The class cannot be represented by the class of

There are algorithms like Ackermann's function

primitive recursive functions are algorithms

that cannot be bounded. That's a clear

since the only iterator is bounded. That's a clear

Recurseive Concepts
of atoms in our universe. For any value of \( n \) that can be written using the number smaller than 5, \( a(n) = \Theta(n) \) grows so slowly that it is less

is a super exponential number involving six levels of

Union/Find algorithm. Note: \( a(4,4) \) is

The inverse of Ackermann's function is important to analyze

Implementation.

Ackermann's function grows too fast to have a for-loop

Primitive recursive function.

With Helm Ackermann observed in 1928 that this is not a

\[
\begin{align*}
A(1, i) &= A(1, i-1) \text{ for } i \geq 2 \\
A(i, 1) &= A(i-1, 2) \text{ for } i \geq 2 \\
A(1, 1) &= 2 \text{ for } i = 1
\end{align*}
\]

Ackermann's Function
You should have learned that in CS2

How do we represent the classes?

Can see if $x \equiv y$, by seeing if $\text{Find}(x) \equiv \text{Find}(y)$

$\text{Find}(x)$ returns the canonical element of $\lfloor x \rfloor$

$\text{Union}(x, y)$'s $x$ and $y$ are in $\mathcal{S}$, merges the class

Singleton equivalence classes

Start with a collection $\mathcal{S}$ of unrelated elements —

Union/Find
condition. It is recursive can be used as the stopping test for one. In fact any predicate that we also allow other predicates besides

\[ F(x_1, \ldots, x_n) = \forall y (G(y, x_1, \ldots, x_n) \Rightarrow \neg \neg F) \]

so is \( F \), where \( F \) is already known to be recursive, then

Mimimization:

The \( \eta \) Operator
Equivalence

$$TM \leq_R TM \leq_P FRS \leq_P REC \leq_T TM$$
Unary Alphabet with 0 as Blank

Representing words over larger alphabets $\Sigma = \{a, b, c\}$

Word $= aca$ $b$

001011110101100

00 separates words

Thus, we can focus on tape alphabet or $\{?\}$ with blank as 0.
Encoding in Instantaneous Description

**String Approach**

\[ \ldots 0010100111 \underline{01} 0 \ldots \]

\[ \underline{01} \]

\[ 10100111 \underline{01} \]

Record shortest string on right that includes scanned square as rightmost non-blank.

Record shortest string on left that includes leftmost non-blank.

Place state to left of scanned square.

**Integer Approach**

\((2, 83, 7)\) for \(10100111 01\)

\[ \uparrow \quad \uparrow \quad \uparrow \]

Right Read R to L

State: Indef

Left Read L to R

**Note:**

If first number is even, scanned square is 0; if odd, then 1.

Same for rightmost symbol on left.
TM & REGISTER MACHINE

CAN STORE TM ID IN JUST THREE REGISTERS

CAN SHIFT LEFT VIA MULTIPLY BY 2
ASSUME $r_2 = 0$, $r_3 = 0$

\[
\begin{align*}
&x, \text{ DEC}_r (x+1, x+4) \quad r_2 = r_1 \times 2 \\
&x+1, \text{ INC } r_2 (x+2) \\
&x+2, \text{ INC } r_2 (x+3) \\
&x+3, \text{ INC } r_3 (x) \\
&x+4, \text{ DEC}_r (x+5, x+6) \\
&x+5, \text{ INC } r_1 (x+4) \\
&x+6.
\end{align*}
\]

CAN SHIFT RIGHT VIA DIVIDE BY 2

DETAILS OF TM & RM

IN SUPPLEMENTAL NOTES
RM ≤ FR \[ S \]

ID FOR RM IS

\[ P^r_1, P^r_2, \ldots, P^n_n, P^{n+i} \]

WHERE \( V^r_k \) IS CONTENTS OF REGISTER \( k \)

AND WE ARE ABOUT TO EXECUTE INSTR. \( i \) CAN SIMULATE BY

\[ J. \text{ INCR}_p[i] \]

\[ P_{n+j} X \rightarrow P_{n+i} P_r X \]

\[ J. \text{ DEC}_r[s, f] \]

\[ P_{n+d} P_r X \rightarrow P_{n+s} X \]

\[ P_{n+j} X \rightarrow P_{n+f} X \]

ALSO

\[ P_{n+m+1} X \rightarrow X \]

FOR HALTING CONDITION

DETAILS IN SUPPLEMENTAL NOTES
The Universal Machine on the selected input.

- The Universal Machine will then simulate argument to this factoring scheme.
  - The first specific factor is the factor argument (encoded) and the second the factor.
  - This is a single recursive function with two

Universal Machine
Encodings

Encode this machine by the number F,

\[ d^r \cdot d_{i+1}^{2n+1} \cdot d_i^{2n+2} \cdot d_{i-1}^{2n} \cdot d_{i+2}^{2n+1} \text{ for } i \leq q_i \]

\[ \exists [q_i \leq 1 \leq q_{i+1}] \]

\[ 2^n \leq q_i \leq 2^{n+1} \]

\[ \exists q_i \leq 1 \leq q_{i+1} \]

- Let \((a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)\) be some factor replacement system, where

\[ a_1, \ldots, a_n \in \{1, 2, \ldots, \}

\[ b_1, \ldots, b_n \in \{1, 2, \ldots, \}

\[ a_i, b_i \in \{1, 2, \ldots, \} \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ b_i = 1 \]

\[ a_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]

\[ a_1, \ldots, a_n \]

\[ b_1, \ldots, b_n \]

\[ (a_1, b_1), \ldots, (a_n, b_n) \]

\[ a_i, b_i \]
\[ \text{NEXT}(F, x) = (x \text{ \textbackslash\textbar\ } \text{exp}(F', \text{RULE}(F', x) - 1)) \times \text{exp}(F', \text{RULE}(F', x)) \]

Given the function \text{RULE}(F, x), we can determine \text{NEXT}(F, x). The number that follows \( x \), when using \( F \), by

\[ \text{RULE}(F, x) \]

returns \( n + 1 \). That's why we added \( p_{2n+1} \).

If \( x \) is not divisible by any \( a_i \) and \( x \) is divisible by \( 1 \) and any \( a_i \) the "involving \( p_{2n-1} \). Thus, \text{RULE}(F', x) = 1."

Note: If \( x \) is divisible by \( a_i \) and \( x \) is the least integer for which this is true, then \( \text{exp}(F', x) = 1 \).

\[ \text{RULE}(F, x) = \text{true} \]

We can determine the rule of \( F \) that applies to \( x \) by

Simulation by Recursive # 1
This assumes we converge to a fixed point only if we stop

\[ \text{HALT}(F, x) = \text{HALT}(F, x', y) = \text{CONF}(F, x', y) \]

\( F \) halts is the number of the configuration on which

\( \text{CONF}(F, x', y+1) = \text{NEXT}(F, \text{CONF}(F, x', y)) \)

\( \text{CONF}(F, x', 0) = x \)

started on \( x \) are

\( \text{CONF} \) configurations listed by \( F \), when

Simulation by Recursive # 2
Factor System that we wish to simulate.

2. We can fix $F$ for any given prime $p$. The exponent of the only even returned as the exponent will be

This assures that the answer will be

$\text{univ}(F, x) = \exp(\text{CONF}(F, x, \text{HALT}(F, x)))) \neq 0$

then be defined by


A Universal Machine that simulates an

Simulation by Recursive #3
Example of Universal Machine in Action

\[ 3 \times 5 \times 6 \rightarrow 2 \times 1 - 61 \]

\[ 3 \times 5 \times x \rightarrow x \]

\[ 3 \times x \rightarrow 2x \]

\[ 5 \times x \rightarrow 2x \]

\[ F = \begin{pmatrix} 2^3 & 5^4 \end{pmatrix} \]

Do for \( x = 3^2 \times 5^4 \)

Rule \( (F, x) = M \times z \) \( (1 \leq z \leq 4) \) \( [P_{2z-1} \times 1] \)

Rule \( (F, 3^2 \times 5^4) = 1 \)

Next \( (F, 3^2 \times 5^4) = (3^2 \times 5^4) \times 1 = 3^2 \times 5^4 \)

Rule \( (F, 3^2 \times 5^4) = 1 \)

Next \( (F, 3^2 \times 5^4) = (3^2 \times 5^4) \times 2 = 2 \times 3^2 \times 5^4 \)

Rule \( (F, 2^3) = 4 \)

Next \( (F, 2^3) = (2^3) \times 1 = 2^3 \)

Config \( (F, 3^2 \times 5^4, 0) = 3^2 \times 5^4 \)

Config \( (F, 3^2 \times 5^4, 1) = 3^2 \times 5^4 \)

Config \( (F, 3^2 \times 5^4, 2) = 5^2 \times 2^2 \times 3^2 \)

Config \( (F, 3^2 \times 5^4, 3) = 2^2 \times 3^2 \times 5^4 \)

Config \( (F, 3^2 \times 5^4, 4) = 2^2 \times 5^4 \)

Config \( (F, 3^2 \times 5^4, 5) = 2^2 \times 5^4 \)

\[ \text{HALT}(F, 3^2 \times 5^4) = 4 \]
RESULT FOR EXAMPLE

Again,
\[ \text{HALT}(F, 3^{3^{3^4}}) = 4 \]

So,
\[ \text{UNIN}(F, 3^{3^{3^4}}) = \exp(\text{CONAG}(F, 3^{3^{3^4}}, 4), 0) \]
\[ = \exp(2^{2^0}) \]
\[ = 2 \]

Note: F and x were arbitrary except that F was a FRS encoding and x was legit input we could write recursive functions that syntactically check F and x, or even just checking F works
**Recursive ≤ Turing**

Show base functions are Turing computable

\[ C^n_a(x_1, \ldots, x_n) = \alpha \]
\[ (R \mid 1)^a \mid R \]

\[ I_i^n(x_1, \ldots, x_n) = x_i \]
\[ C_{n-i+1} \]

\[ S(x) = x + 1 \]
\[ C_1 \mid 1 \mid R \]

Now show Turing computable closed under composition, induction and minimization

Details in supplemental notes
Universal Machine

Really an interpreter for programs in some model of computation, written in that model.

\[ \text{Univ}(x,y) = \varphi_x(y) \]

where \( \varphi_x \) is \( x \)-th program in some way of ordering programs, e.g., lexically.

\[ \varphi(x, y) = \text{Univ}(x, y) = \varphi_x(y) \]
Halting Problem

Assume algorithmic predicate Halt
\[ \text{Halt}(f, x) \Leftrightarrow \phi_f(x) \downarrow \]

Define
\[ \text{Disagree}(x, y) = \mu y \lceil \text{Halt}(x, x) \downarrow \text{NOT} \rceil \]

Clearly
- If \( \neg \text{Halt}(x, x) \) then \( \text{Disagree}(x) = 0 \)
- If \( \not\exists \text{Halt}(x, x) \) then \( \text{Disagree}(x) \uparrow \)

Or
\[ \text{Halt}(x, x) \Leftrightarrow \text{Disagree}(x) \uparrow \]

Or
\[ \phi_x(x) \downarrow \Leftrightarrow \text{Disagree}(x) \uparrow \]

Since Halt is an algorithm, Disagree is an effective procedure and so, for some \( q \),
\[ \phi_q = \text{Disagree} \]

But then
\[ \phi_q(q) \downarrow \Leftrightarrow \text{Disagree}(q) \uparrow \Leftrightarrow \phi_q(q) \uparrow \]

If so Halt cannot exist.
Semi-Decide-Halting() 

{ 
  Print "yes";
  p(x);
  Read p, x;
}

Halting Problem. Here is a procedural description:

Run the procedure P on input x until it stops. If it stops, say "yes". If P does not stop, we will provide no answer. This semi-decides the

To see this, consider the following semi-decision procedure. Let P

While the Halting Problem is not solvable, it is re-recognizible or

Halting ( ATM ) is re-recognizable
This gives us a way to enumerate the recursively enumerable (semi-decidable) sets.

**Proof:** Follows from definition of $\phi(x,u)^\#(u,x)$.

**Theorem:** A set $B$ is recursive if there exists an $u$ such that $B = \text{W}_u^n$.

**Theorem:** $\{ \uparrow(x,u)^\# \mid x \in \text{W}_u^n \}$

Define

**Enumeration Theorem**
where $F_0$, $F_1$, $F_2$, ... is a list of indexes of all and only the algorithms

\[ F(x) = F^x \]

enumerated, and that $F$ accomplishes this. Then assume that the set of algorithms (TOTALL) can be effective procedure $P$, whether or not $P$ is an algorithm.

Problem, that is, the problem to decide of an arbitrary Halting

The classic non-re problem is the Uniform Halting

unsolvable -- they're not even semi-decidable.

There are even "practical" problems that are worse than

Non-Re Problems
(partial) recursive functions.

undefined. In fact, we already have shown how to enumerate the
enumerable, since the above is not a contradiction when $G(9)$ is
This cannot be used to show that the effective procedures are non-
an algorithm.

But then $G$ contradicts its own existence since $G$ would need to be

\[ G(9) = 1 + 1 = 2 \]

\[ F(9) = F^9 G \]

Then

But then $G$ is itself an algorithm. Assume it is the 9-th one

\[ 1 + (x)^x F = (x)^{(x)F} \phi = 1 + (x) (x) \text{ Univ} \]

Define

The Contradiction
Proof: Shown earlier.

Theorem: \( \text{TOTAL} \) is not re.

\[ \text{Domain of } \Phi \]

\( \{ \mathcal{N} \mid \mathcal{W} \subseteq \mathcal{N} \} = \text{TOTAL} \)

We can also note that

\[ \{ \uparrow (x) \mid \mathcal{N} \in \text{TOTAL} \} = \text{TOTAL} \]

as

The listing of all algorithms can be viewed

The set \( \text{TOTAL} \)
algorithmic acceptor of such programs. Procedures can be generated in fact, we can build an algorithmic acceptor of all programs. Of course, if you buy Church’s Theorem, the set of all divergent (can accept all and only algorithms) No parsing system (even one that rejects by No generative system (e.g., grammar) can produce descriptions of all and only algorithms)
Many unbounded ways
- Factor Replacement Systems, Petti Nets
- It sometimes hinders

- Push Down Automata
- It sometimes helps

- Linear Bounded Automata
- Turing Machines, Finite-State Automata
- It sometimes doesn't matter

Non-determinism
How hard is it to analyze Petri nets?

To determine if some marking can eventually arise is in EXPSPACE(N).

Solvable, but takes exponential space.

Time is actually $2^{2^N}$.

If priority added to transitions, Petri nets are complete models of computation.

Can recast as FRS w/o ordering $\equiv$ Petri Net w/ ordering $\equiv$ Petri Net with priorities.
some open problem in which we are interested. Then shows that this problem is no harder than starts with some known unsolvable problem and technique commonly used is called reduction. The other, open problems are unsolvable. The proofs by contradiction are tedious after you've seen a few. We really would like proofs that...
Problem Categories

Recursive (Solvable)
Lots of Examples

RE, Non-recursive (Undec but Semi-Dec)
HALT = \{ (\Sigma, \phi) | \phi(\Sigma) \downarrow \}

Shown by Diagonalization

Non-RE (Cannot even Semi-Decide)
TOTAL = \{ (\Sigma, \phi) | \phi(\Sigma) \downarrow \}

Problem Categories

Recursive (Solvable)
Lots of Examples

RE, Non-Recursive (Undec but Semi-Dec)

\[
\text{HALT} = \{ \langle \phi, x \rangle \mid \phi(x) \downarrow \}
\]

Shown by Diagonalization

Non-RE (Cannot Even Semi-DECIDE)

\[
\text{Total} = \{ \phi \mid \forall x \phi(x) \downarrow \}
\]
Intro to Reduction

A \leq_m B \text{ if there exists some computable algorithm } f \Rightarrow \forall x \in A \Leftrightarrow f(x) \in B

If B is easy to solve then so is A if f does not add to computational complexity

However, if A is known to be hard (or even unsolvable) and f does not change the complexity landscape, then B must be hard at least within the order of f and A's complexity, if A is unsolvable then so is B.
Showing Complexity of New Problems

First Technique is Reduction

Let \( B \) be some set of unknown complexity

Let \( A \) be some set of known complexity

Let \( f \) be a computable 1-1 function (total)

\[ A \leq_m B \text{ or just } A \leq B \]

via \( f \) if

\[ x \in A \text{ iff } f(x) \in B \]

If \( A \) is RE, non-re. then \( B \)

is non-re., but not nec. RE

If \( A \) is non-re-then \( B \)

is non-re and, of course, non-rec.
Reduction Example #1

Show \( \text{HALT} \leq \text{TOTAL} \)

Let \( s, x \) be arbitrary nat. numbers \( \langle s, x \rangle \in \text{HALT} \iff \Phi_s(x) \downarrow \)

Define \( F_x \) by
\[
\forall y \forall x \Phi_x(y) = \Phi_s(x) \quad \text{//ignores input}
\]

Now \( \langle s, x \rangle \in \text{HALT} \iff F_x \in \text{TOTAL} \)

Thus, \( \text{HALT} \leq \text{MTOTAL} \)

By this, \( \text{TOTAL} \) is non-rec. But we do not know if it's re (well, we do, and it's not)

Note: We can leave out \( \Phi \)

And just say \( \forall y \forall F_x(y) = s(x) \quad \text{//overload} \)

For convenience
Example #2

$$\text{HAS}_\text{ZERO} = \{S | \exists x \ f(x) = 0 \}$$ //skip φ

Show $\text{HAS}_\text{ZERO}$ is non-rec.

Let $S, x$ be arb.

Define $F_x$ by

$$\forall y \ F_x(y) = -f(x) - f(x)$$

Clearly, $\forall y F_x(y) = 0$ if $f(x) \downarrow$

else $\forall y F_x(y) \uparrow$

So

$$\langle S, x \rangle \in \text{HALT} \iff F_x \text{ HAS}_\text{ZERO}$$

Thus, $\text{HAS}_\text{ZERO}$ is non-rec, since

$$\text{HALT} \leq_m \text{HAS}_\text{ZERO}$$

But is $\text{HAS}_\text{ZERO}$ re?

Well it is and we will show that later.
Example 3

\[ \text{Zero} = \{ f \mid \forall x f(x) = 0 \} \]

Show zero is non-re.

Note prior example showed non-rec ! !.

Let \( f \) be arb.

Define

\[ \forall x h(x) = f(x) - f(x) \]

Now

\( f \in \text{Total} \iff \forall x f(x) \downarrow \)

\( \iff \forall x h(x) = 0 \)

\( \iff h \in \text{Zero} \)

Thus,

\( \text{Total} \leq_m \text{Zero} \)

And so zero is non-re.
Example #4

IDENTITY = \exists f | \forall x f(x) = x? \\

Let f be an arbitrary index

Define

\forall x \ g_f(x) = f(x) - f(x) + x

Now

f is total iff \forall x \ f(x) \neq x

iff \forall x \ g_f(x) = x

iff g_f \subseteq \text{identity}

Thus,

TOTAL \ \subseteq \text{identity}

And so identity is not even REL...
TYPES OF REDUCTION

\[ M - 1 \leq m \]
\[ 1 - 1 \leq 1 \]

TURING (AKA ORACLE)
\[ \leq t \]

DEGREES ARE EQUIV. CLASSES

\[ \equiv 1 \]
\[ \equiv m \]
\[ \equiv t \]

ONE CLASS WE CARE ABOUT
IS COMPLETE DEGREE (HIGHEST)
AMONG RE SETS
RE COMPLETE

S IS RE COMPLETE IFF
(1) S IS RE
(2) IF T IS RE THEN T \subseteq S

HALT (AKA K₀) IS RE COMPLETE

LET A BE ARB RE SET THEN
A = Dom..Qₐ FOR SOME INDEX a
Here A = Wₐ (ENUMERATION THEOREM)

x \in A \iff x \in Dom(Qₐ) \iff Qₐ(x) \iff (a, x) \in K₀

Thus
A \leq K₀ (REALLY A \leq₁ K₀)
$K$ is also RE-complete

$K = \{ f \mid Q_f(s) \downarrow \}$

Let $s, x$ be arb.

Define $\forall y. F_x(y) = f(x)$

$\langle s, x \rangle \in \text{HALT}(K_0) \iff$

$\quad \forall y. F_x(y) \downarrow$

$\Rightarrow F_x \in K$

$\langle s, x \rangle \not\in \text{HALT}(K_0) \iff$

$\quad \forall y. F_x(y) \uparrow$

$\Rightarrow F_x \not\in K$

Thus,

$K_0 \leq K$ (actually $K \equiv K_0$)

But $K$ is obviously RE

and so $K$ is RE-complete