PRIMITIVE RECURSIVE FUNCTIONS

BASE FUNCTIONS ARE PRFs

$$C_a(\vec{x}) = a$$ CONSTANTS

$$I^n_i(x_1, \ldots, x_n) = x_i$$ PROJECTIONS (IDENTITY)

$$S(x) = x+1$$ SUCCESSOR (INCREMENT)

BUILD MORE VIA

$$F(\vec{x}) = H(G_1(\vec{x}), \ldots, G_k(\vec{x}))$$

PRFs

COMPOSITION

$$F(\vec{x}, 0) = G(\vec{x})$$

$$F(\vec{x}, y+1) = H(\vec{x}, y, F(x, y))$$

INDUCTION (PRIMITIVE RECURSION)

PRFs

BUILDING NEW PRFs

ADDITION: FORMAL

$$+(x,0) = I^1_1(x)$$
$$+(x,y+1) = \underbrace{S(I^3_3}_{\text{COMPOSITION}}(x,y,+(x,y)))$$

ADDITION: LESS FORMAL

$$x+0 = x$$
$$x+(y+1) = (x+y)+1$$

MULTIPLICATION: FORMAL

$$*(x,0) = C_0(x)$$
$$*(x,y+1) = H(x,y,*(x,y))$$
$$H(x,y,z) = +(I^3_1(x,y,z), I^3_3(x,y,z))$$

MULTIPLICATION: LESS FORMAL

$$x*0 = 0$$
$$x*(y+1) = x*y + x$$

MORE BASIC ARITHMETIC

PREDECESSOR: (LIMITED)

$$0 - 1 = 0$$

$$(x+1) - 1 = x$$

SUBTRACTION: (LIMITED)

$$x \dot{-} 0 = x$$

$$x \dot{-} (y+1) = (x \dot{-} y) - 1$$

FACTORIAL:

$$0! = 1$$

$$(x+1)! = x! * (x+1)$$

RELATIONS

ISZERO:

$0 == 0 = 1$

$(x+1) == 0 = 0$

EQUALITY AND ONE OTHER:

$x == y = ((x-y) + (y-x)) == 0$

$x \le y = (x-y) == 0$

BOOLEANS

NEGATION

$\sim x = x == 0$

AND

$x \&\& y = (x * y)$

OR

$x || y = \sim((x == 0) \&\& (y == 0))$

BOUNDED MINIMIZATION

$$f(x) = \mu z (z \leq x) [P(z)] \quad \text{IF } \exists \text{ SUCH } z$$
$$= x+1 \quad \text{OTHERWISE}$$

$$f(0) = 1 - P(0)$$
$$f(x+1) = [f(x) * (f(x) \leq x))$$
$$+ ((x+2-P(x+1)) * \sim (f(x) \leq x))$$

$$f(x) = \mu z (z < x) [P(z)] \quad \text{IF } \exists \text{ SUCH } z$$
$$= x \quad \text{OTHERWISE}$$

$$f(0) = 0$$
$$f(x+1) = \mu z (z \leq x) [P(z)]$$

DIVISION & DIVISIBILITY

DIVISION:

NEED A VALUE

$$x // 0 = 0$$

$$x // (y+1) = \mu z (z < x) [(z+1) * (y+1) > x]$$

DIVISIBILITY

$$x \mid y = ((y // x) * x) == y$$

EXPONENTS

POWER

$$\left.\begin{array}{l} x \wedge 0 = 1 \\ x \wedge (y+1) = x * (x \wedge y) \end{array}\right\} \text{ABBREVIATE } x^y$$

PRIMALITY

$$\text{FIRSTFACTOR}(x) = \mu z \, (2 \le z \le x) [z \mid x]$$
$$0 \quad \text{IF NONE}$$

$$\text{ISPRIME}(x) = \text{FIRSTFACTOR}(x) = x \;\&\&\; (x > 1)$$

$$\text{PRIME}(0) = 2$$
$$\text{PRIME}(x+1) = \mu z (\text{PRIME}(x) < z \le \text{PRIME}(x)! + 1) [\text{ISPRIME}(z)]$$

ABBREVIATE PRIME(i) AS p_i

Pairing Functions

- $\mathbf{pair(x,y) = \langle x,y\rangle = 2^x\ (2y + 1) - 1}$
- with inverses

  $\langle z\rangle_1 = \mathrm{exp}(z+1,0)$

  $\langle z\rangle_2 = (((\, z + 1\,) \,//\, 2^{\langle z\rangle_1}\,) - 1\,) \,//\, 2$
- These are very useful and can be extended to encode **n**-tuples

  $\langle x,y,z\rangle = \langle x, \langle y,z\rangle \rangle$ (note: stack analogy)

Pairing Function is 1-1 Onto

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

Approach 1:

We will look at two cases, where we use the following modification of the pairing function, $\langle x,y \rangle+1$, which implies the problem of mapping the pairing function to Z^+.

Case 1 ($x=0$)

Case 1:

For $x = 0$, $\langle 0,y\rangle+1 = 2^0(2y+1) = 2y+1$. But every odd number is by definition one of the form $2y+1$, where $y\geq 0$; moreover, a particular value of y is uniquely associated with each such odd number and no odd number is produced by $2^x(2y+1)$ when $x>0$. Thus, $\langle 0,y\rangle+1$ is 1-1 onto the odd natural numbers.

Case 2 ($x > 0$)

Case 2:

For $x > 0$, $\langle x,y\rangle+1 = 2^x(2y+1)$, where $2y+1$ ranges over all odd number and is uniquely associated with one based on the value of y (we saw that in case 1). 2^x must be even, since it has a factor of 2 and hence $2^x(2y+1)$ is also even. Moreover, from elementary number theory, we know that every even number except zero is of the form $2^x z$, where $x>0$, z is an odd number and this pair x,z is unique. Thus, $\langle x,y\rangle+1$ is 1-1 onto the even natural numbers, when $x>0$.

The above shows that $\langle x,y\rangle+1$ is 1-1 onto Z^+, but then $\langle x,y\rangle$ is 1-1 onto $\aleph$, as was desired.

μ Recursive

4th Model

A Simple Extension to Primitive Recursive

μ Recursive Concepts

- All primitive recursive functions are algorithms since the only iterator is bounded. That's a clear limitation.
- There are algorithms like Ackerman's function that cannot be represented by the class of primitive recursive functions.
- The class of recursive functions adds one more iterator, the minimization operator (μ), read "the least value such that."

Ackermann's Function

- **$A(1, j)=2j$ for $j \geq 1$**
- **$A(i, 1)=A(i-1, 2)$** for **$i \geq 2$**
- **$A(i, j)=A(i-1, A(i, j-1))$** for **$i, j \geq 2$**
- Wilhelm Ackermann observed in 1928 that this is not a primitive recursive function.
- Ackermann's function grows too fast to have a for-loop implementation.
- The inverse of Ackermann's function is important to analyze Union/Find algorithm. Note: $A(4,4)$ is a super exponential number involving six levels of exponentiation. $\alpha(n) = A^{-1}(n, n)$ grows so slowly that it is less than 5 for any value of n that can be written using the number of atoms in our universe.

Union/Find

- Start with a collection **S** of unrelated elements – singleton equivalence classes
- **Union(x,y)**, **x** and **y** are in **S**, merges the class containing **x** (**[x]**) with that containing **y** (**[y]**)
- **Find(x)** returns the canonical element of **[x]**
- Can see if **x≡y**, by seeing if **Find(x)==Find(y)**
- How do we represent the classes?
- You should have learned that in CS2

The μ Operator

- Minimization:

  If **G** is already known to be recursive, then so is **F**, where

  $$\mathbf{F(x1,\ldots,xn) = \mu y\ (G(y,x1,\ldots,xn) \cancel{==} \cancel{1})}$$

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

PREDICATE

PRIMITIVE RECURSIVE FUNCTIONS

BASE FUNCTIONS ARE PRFs

$C_a(\vec{x}) = a$ CONSTANTS

$I_i^n(x_1, \ldots, x_n) = x_i$ PROJECTIONS (IDENTITY)

$S(x) = x+1$ SUCCESSOR (INCREMENT)

BUILD MORE VIA

$$F(\vec{x}) = H(G_1(\vec{x}), \ldots, G_k(\vec{x}))$$

PRFs

COMPOSITION

$$F(\vec{x}, 0) = G(\vec{x})$$
$$F(\vec{x}, y+1) = H(\vec{x}, y, F(x, y))$$

INDUCTION (PRIMITIVE RECURSION)

PRFs

BUILDING NEW PRFs

ADDITION: FORMAL

$$+(x,0) = I_1^1(x)$$
$$+(x,y+1) = \underbrace{S(I_3^3}_{\text{COMPOSITION}}(x,y,+(x,y)))$$

ADDITION: LESS FORMAL

$$x + 0 = x$$
$$x + (y+1) = (x+y)+1$$

MULTIPLICATION: FORMAL

$$*(x,0) = C_0(x)$$
$$*(x,y+1) = H(x,y,*(x,y))$$
$$H(x,y,z) = +(I_1^3(x,y,z), I_3^3(x,y,z))$$

MULTIPLICATION: LESS FORMAL

$$x * 0 = 0$$
$$x * (y+1) = x * y + x$$

MORE BASIC ARITHMETIC

PREDECESSOR: (LIMITED)

$$0 - 1 = 0$$

$$(x+1) - 1 = x$$

SUBTRACTION: (LIMITED)

$$x \dot{-} 0 = x$$

$$x \dot{-} (y+1) = (x - y) - 1$$

FACTORIAL:

$$0! = 1$$

$$(x+1)! = x! * (x+1)$$

RELATIONS

ISZERO:

$$0 == 0 = 1$$

$$(x+1) == 0 = 0$$

EQUALITY AND ONE OTHER:

$$x == y = ((x-y) + (y-x)) == 0$$

$$x \leq y = (x-y) == 0$$

BOOLEANS

NEGATION

$$\sim x = x == 0$$

AND

$$x \&\& y = (x * y)$$

OR

$$x \| y = \sim((x==0) \&\& (y==0))$$

BOUNDED MINIMIZATION

$$f(x) = \mu z (z \leq x) [P(z)] \quad \text{IF } \exists \text{ SUCH } z$$
$$= x+1 \quad \text{OTHERWISE}$$

$$f(0) = 1 - P(0)$$
$$f(x+1) = [f(x) * (f(x) \leq x))$$
$$+ ((x+2-P(x+1)) * \sim (f(x) \leq x))$$

$$f(x) = \mu z (z < x) [P(z)] \quad \text{IF } \exists \text{ SUCH } z$$
$$= x \quad \text{OTHERWISE}$$

$$f(0) = 0$$
$$f(x+1) = \mu z (z \leq x) [P(z)]$$

DIVISION & DIVISIBILITY

DIVISION: NEED A VALUE

$$X // 0 = 0$$

$$X // (y+1) = \mu z\,(z < x)\,[(z+1) * (y+1) > X]$$

DIVISIBILITY

$$x \mid y = ((y // x) * x) == y$$

EXPONENTS

POWER

$$\left.\begin{aligned} X \wedge 0 &= 1 \\ X \wedge (y+1) &= X * (X \wedge y) \end{aligned}\right\} \text{ABBREVIATE } X^y$$

PRIMALITY

$$\text{FIRSTFACTOR}(X) = \mu z\,(2 \le z \le x)\,[z \mid x]$$

0 IF NONE

$$\text{ISPRIME}(X) = \text{FIRSTFACTOR}(X) = X \,\&\&\, (X > 1)$$

$$\text{PRIME}(0) = 2$$

$$\text{PRIME}(x+1) = \mu z\,(\text{PRIME}(X) < z \le \text{PRIME}(X)! + 1)\,[\text{ISPRIME}(z)]$$

ABBREVIATE PRIME (i) AS Pi

Pairing Functions

- **pair(x,y)** $= \langle x,y \rangle = 2^x (2y + 1) - 1$
- with inverses

  $\langle z \rangle_1 = \mathrm{exp}(z+1,0)$

  $\langle z \rangle_2 = (((z + 1) // 2^{\langle z \rangle_1}) - 1) // 2$
- These are very useful and can be extended to encode **n**-tuples

  $\langle x,y,z \rangle = \langle x, \langle y,z \rangle \rangle$ (note: stack analogy)

Pairing Function is 1-1 Onto

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

Approach 1:

We will look at two cases, where we use the following modification of the pairing function, $\langle x,y \rangle + 1$, which implies the problem of mapping the pairing function to Z^+.

Case 1 ($x=0$)

Case 1:

For $x = 0$, $<0,y>+1 = 2^0(2y+1) = 2y+1$. But every odd number is by definition one of the form $2y+1$, where $y \geq 0$; moreover, a particular value of y is uniquely associated with each such odd number and no odd number is produced by $2^x(2y+1)$ when $x>0$. Thus, $<0,y>+1$ is 1-1 onto the odd natural numbers.

Case 2 $(x > 0)$

Case 2:

For $x > 0$, $\langle x,y\rangle+1 = 2^x(2y+1)$, where $2y+1$ ranges over all odd number and is uniquely associated with one based on the value of y (we saw that in case 1). 2^x must be even, since it has a factor of 2 and hence $2^x(2y+1)$ is also even. Moreover, from elementary number theory, we know that every even number except zero is of the form 2^xz, where $x>0$, z is an odd number and this pair x,z is unique. Thus, $\langle x,y\rangle+1$ is 1-1 onto the even natural numbers, when $x>0$.

The above shows that $\langle x,y\rangle+1$ is 1-1 onto Z^+, but then $\langle x,y\rangle$ is 1-1 onto $\aleph$, as was desired.

μ Recursive

4^{th} Model

A Simple Extension to Primitive Recursive

μ Recursive Concepts

- All primitive recursive functions are algorithms since the only iterator is bounded. That's a clear limitation.
- There are algorithms like Ackerman's function that cannot be represented by the class of primitive recursive functions.
- The class of recursive functions adds one more iterator, the minimization operator (μ), read "the least value such that."

Ackermann's Function

- **$A(1, j)=2j$ for $j \geq 1$**
- **$A(i, 1)=A(i-1, 2)$** for **$i \geq 2$**
- **$A(i, j)=A(i-1, A(i, j-1))$** for **$i, j \geq 2$**
- Wilhelm Ackermann observed in 1928 that this is not a primitive recursive function.
- Ackermann's function grows too fast to have a for-loop implementation.
- The inverse of Ackermann's function is important to analyze Union/Find algorithm. Note: A(4,4) is a super exponential number involving six levels of exponentiation. $\alpha(n) = A^{-1}(n, n)$ grows so slowly that it is less than 5 for any value of n that can be written using the number of atoms in our universe.

Union/Find

- Start with a collection **S** of unrelated elements – singleton equivalence classes
- **Union(x,y)**, **x** and **y** are in **S**, merges the class containing **x** (**[x]**) with that containing **y** (**[y]**)
- **Find(x)** returns the canonical element of **[x]**
- Can see if **x≡y**, by seeing if **Find(x)==Find(y)**
- How do we represent the classes?
- You should have learned that in CS2

The μ Operator

- Minimization:

  If **G** is already known to be recursive, then so is **F**, where

  $$\mathbf{F(x1,\ldots,xn) = \mu y\ (G(y,x1,\ldots,xn)\ \cancel{==\ 1})}$$

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

PREDICATE

EQUIVALENCE

$$TM \le RM \le FRS \le REC \le TM$$

UNARY ALPHABET WITH 0 AS BLANK

REPRESENTING WORDS OVER LARGER ALPHABETS

$\Sigma = \{a, b, c\}$

WORD = acab

$0\underbrace{010111010110}\,0$

00 SEPARATES WORDS

THUS, WE CAN FOCUS ON TAPE ALPHABET OR $\{1\}$ WITH BLANK AS 0.

ENCODING TM INSTANTANEOUS DESCRIPTION

STRING APPROACH

$$\ldots 00\underbrace{1010011}\, q_7 \underbrace{\underline{0}1}\, 00 \ldots$$

$$1010011\, q_7\, 01$$

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 $1010011\, q_7\, 01$

RIGHT READ R TO L

LEFT READ L TO R

STATE INDE

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 $r_2 = 0$; $r_3 = 0$

X. DEC_{r_1} (X+1, X+4)
X+1. INC r_2 (X+2)
X+2. INC r_2 (X+3)
X+3. INC r_3 (X)
X+4. DEC_{r_3} (X+5, X+6)
X+5. INC_{r_1} (X+4)
X+6.

$r_2 = r_1 * 2$
$r_3 = r_1$
$r_1 = 0$

$r_1 = r_3$
$r_3 = 0$

CAN SHIFT RIGHT VIA DIVIDE BY 2

DETAILS OF TM ≤ RM
IN SUPPLEMENTAL NOTES

RM $\leq$ FRS

ID FOR RM IS

$$P_1^{r_1} P_2^{r_2} \cdots P_n^{r_n} P_{n+j}$$

WHERE r_k IS CONTENTS OF REGISTER k
AND WE ARE ABOUT TO EXECUTE INSTR. j.

CAN SIMULATE BY

j. $INCR_r[i]$

$$P_{n+j} X \rightarrow P_{n+i} P_r X$$

j. $DEC_r[s,f]$

$$P_{n+j} P_r X \rightarrow P_{n+s} X$$

$$P_{n+j} X \rightarrow P_{n+f} X$$

ALSO

$$P_{n+m+1} X \rightarrow X$$

FOR HALTING CONDITION

DETAILS IN SUPPLEMENTAL NOTES

Universal Machine

- In the process of doing this reduction, we will build a Universal Machine.
- This is a single recursive function with two arguments. The first specifies the factor system (encoded) and the second the argument to this factor system.
- The Universal Machine will then simulate the given machine on the selected input.

Encoding FRS

- Let $(n, ((a_1,b_1), (a_2,b_2), \ldots ,(a_n,b_n))$ be some factor replacement system, where (a_i,b_i) means that the **i**-th rule is

  $$a_i x \rightarrow b_i x$$

- Encode this machine by the number **F**,

  $$2^n 3^{a_1} 5^{b_1} 7^{a_2} 11^{b_2} \cdots p_{2n-1}^{a_n} p_{2n}^{b_n} p_{2n+1} p_{2n+2}$$

Simulation by Recursive # 1

- We can determine the rule of **F** that applies to **x** by

  $$\mathbf{RULE(F, x) = \mu\, z\, (1 \le z \le exp(F, 0)+1)\ [\ exp(F, 2*z\text{-}1) \mid x\ ]}$$

- Note: if **x** is divisible by $\mathbf{a_i}$, and **i** is the least integer for which this is true, then $\mathbf{exp(F,2*i\text{-}1) = a_i}$ where $\mathbf{a_i}$ is the number of prime factors of **F** involving $\mathbf{p_{2i-1}}$. Thus, **RULE(F,x) = i**.

  If x is not divisible by any $\mathbf{a_i}$, $\mathbf{1 \le i \le n}$, then **x** is divisible by **1**, and **RULE(F,x)** returns **n+1**. That's why we added $\mathbf{p_{2n+1}\ p_{2n+2}}$.

- Given the function **RULE(F,x)**, we can determine **NEXT(F,x)**, the number that follows **x**, when using **F**, by

  NEXT(F, x) = (x // exp(F, 2*RULE(F, x)-1)) * exp(F, 2*RULE(F, x))

Simulation by Recursive # 2

- The configurations listed by **F**, when started on **x**, are

CONFIG(F, x, 0) = x

CONFIG(F, x, y+1) = NEXT(F, CONFIG(F, x, y))

- The number of the configuration on which **F** halts is

HALT(F, x) = μ y [CONFIG(F, x, y) == CONFIG(F, x, y+1)]

This assumes we converge to a fixed point only if we stop

Simulation by Recursive # 3

- A Universal Machine that simulates an arbitrary Factor System, Turing Machine, Register Machine, Recursive Function can then be defined by
  Univ(F, x) = exp (CONFIG (F, x, HALT (F, x)), 0)
- This assumes that the answer will be returned as the exponent of the only even prime, **2**. We can fix **F** for any given Factor System that we wish to simulate.

EXAMPLE OF UNIVERSAL MACHINE IN ACTION

$3^a 5^b \overset{*}{\Rightarrow} 2^{|a-b|}$

$3\cdot 5x \to x$
$3x \to 2x$
$5x \to 2x$

$F = 2^3 \overbrace{3^{15} 5^1}^{3.5x\to x} \overbrace{7^3 11^2}^{3x\to 2x} \overbrace{13^5 19^2}^{5x\to 2x} \underbrace{19^1 23^1}_{\text{IMPLICIT } x\to x}$

DO FOR $X = 3^2 5^4$

$\text{RULE}(F,x) = \mu z\,(1 \le z \le 4)\,[\,p_{2z-1} \mid x\,]$

$\text{RULE}(F, 3^2 5^4) = 1$

$\text{NEXT}(F, 3^2 5^4) = (3^2 5^4 // 15) * 1 = 3^1 5^3$

$\text{RULE}(F, 3^1 5^3) = 1$

$\text{NEXT}(F, 3^1 5^3) = (3 5^3 // 15) * 1 = 5^2$

$\text{RULE}(F, 5^2) = 3$

$\text{NEXT}(F, 5^2) = (5^2 // 5) * 2 = 2^1 5^1$

$\text{NEXT}(F, 2^1 5^1) = (2^1 5^1 // 5) * 2 = 2^2$

$\text{RULE}(F, 2^1 5^1) = 3$

$\text{RULE}(F, 2^2) = 4$

$\text{NEXT}(F, 2^2) = (2^2 // 1) * 1 = 2^2$

$\text{CONFIG}(F, 3^2 5^4, 0) = 3^2 5^4$

$\text{CONFIG}(F, 3^2 5^4, 1) = 3^1 5^3$

$\text{CONFIG}(F, 3^2 5^4, 2) = 5^2$

$\text{CONFIG}(F, 3^2 5^4, 3) = 2^1 5^1$

$\left.\begin{array}{l} \text{CONFIG}(F, 3^2 5^4, 4) = 2^2 \\ \text{CONFIG}(F, 3^2 5^4, 5) = 2^2 \end{array}\right\} \text{HALT}(F, 3^2 5^4) = 4$

RESULT FOR EXAMPLE

AGAIN,

$$\text{HALT}(F, 3^2 5^4) = 4$$

SO,

$$\begin{aligned} \text{UNIV}(F, 3^2 5^4) &= \text{EXP}(\text{CONFIG}(F, 3^2 5^4, 4), 0) \\ &= \text{EXP}(2^2, 0) \\ &= 2 \end{aligned}$$

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

$C_a^n(x_1, \ldots, x_n) = a$

$(R\,1)^a R$

$I_i^n(x_1, \ldots, x_n) = x_i$

C_{n-i+1}

$S(x) = x+1$

$C_1\,1\,R$

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

$$\mathrm{Univ}(x,y) = \varphi_x(y)$$

WHERE φ_x IS x-TH PROGRAM IN SOME WAY OF ORDERING PROGRAMS, E.G., LEXICALLY.

$$\varphi(x,y) = \mathrm{Univ}(x,y)$$
$$= \varphi_x(y)$$

HALTING PROBLEM

ASSUME ALGORITHMIC PREDICATE HALT

$$\text{HALT}(f, x) \Leftrightarrow \varphi_f(x)\downarrow$$

DEFINE

$$\text{DISAGREE}(x) = \mu y\, [\sim \text{HALT}(x,x)]$$

NOT

CLEARLY

IF $\sim$HALT(X,X) THEN DISAGREE(X)=0

IF HALT(X,X) THEN DISAGREE(X)$\uparrow$

OR

$$\text{HALT}(x,x) \Leftrightarrow \text{DISAGREE}(x)\uparrow$$

OR

$$\varphi_x(x)\downarrow \Leftrightarrow \text{DISAGREE}(x)\uparrow$$

SINCE HALT IS AN ALGORITHM, DISAGREE IS AN EFFECTIVE PROCEDURE AND SO, FOR SOME d,

$$\varphi_d \equiv \text{DISAGREE}$$

BUT THEN

$$\varphi_d(d)\downarrow \Leftrightarrow \text{DISAGREE}(d)\uparrow \Leftrightarrow \varphi_d(d)\uparrow$$

$\times$ SO HALT CANNOT EXIST

Halting (A_{TM}) is recognizable

While the Halting Problem is not solvable, it is re, recognizable or semi-decidable.

To see this, consider the following semi-decision procedure. Let *P* be an arbitrary procedure and let *x* be an arbitrary natural number. 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 Halting Problem. Here is a procedural description.

```
Semi_Decide_Halting() {
    Read P, x;
    P(x);
    Print "yes";
}
```

Enumeration Theorem

- Define
  $$\mathbf{W_n} = \{\ \mathbf{x} \in \mathcal{N} \mid \varphi(\mathbf{n},\mathbf{x})\downarrow\ \}$$
- Theorem: A set **B** is re iff there exists an **n** such that $\mathbf{B} = \mathbf{W_n}$.
  Proof: Follows from definition of $\varphi(\mathbf{n},\mathbf{x})$.
- This gives us a way to enumerate the recursively enumerable (semi-decidable) sets.

Non-re Problems

- There are even "practical" problems that are worse than unsolvable -- they're not even semi-decidable.
- The classic non-re problem is the Uniform Halting Problem, that is, the problem to decide of an arbitrary effective procedure P, whether or not P is an algorithm.
- Assume that the set of algorithms (TOTAL) can be enumerated, and that F accomplishes this. Then

  $F(x) = F_x$

  where F_0, F_1, F_2, … is a list of indexes of all and only the algorithms

The Contradiction

- Define $G(x) = Univ(F(x), x) + 1 = \varphi_{F(x)}(x) = F_x(x) + 1$
- But then G is itself an algorithm. Assume it is the g-th one

  $F(g) = F_g = G$

  Then, $G(g) = F_g(g) + 1 = G(g) + 1$
- But then G contradicts its own existence since G would need to be an algorithm.
- This cannot be used to show that the effective procedures are non-enumerable, since the above is not a contradiction when G(g) is undefined. In fact, we already have shown how to enumerate the (partial) recursive functions.

The Set TOTAL

- The listing of all algorithms can be viewed as
  TOTAL = $\{ f \in \boldsymbol{N} \mid \forall x\ \varphi_f(x)\downarrow \}$
- We can also note that
  TOTAL = $\{ f \in \boldsymbol{N} \mid W_f = \boldsymbol{N} \}$, where W_f is the domain of φ_f
- Theorem: TOTAL is not re.
  Proof: Shown earlier.

Insights

Non-re nature of algorithms

- No generative system (e.g., grammar) can produce descriptions of all and only algorithms
- No parsing system (even one that rejects by divergence) can accept all and only algorithms

- Of course, if you buy Church's Theorem, the set of all procedures can be generated. In fact, we can build an algorithmic acceptor of such programs.

Many unbounded ways

- How do you achieve divergence, i.e., what are the various means of unbounded computation in each of our models?
- GOTO: Turing Machines and Register Machines
- Minimization: Recursive Functions
  - Why not primitive recursion/iteration?
- Fixed Point: (Ordered) Factor Replacement Systems

Non-determinism

- It sometimes doesn't matter
  - Turing Machines, Finite-State Automata, Linear Bounded Automata
- It sometimes helps
  - Push Down Automata
- It sometimes hinders
  - Factor Replacement Systems, Petri Nets

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

Reduction Concepts

- Proofs by contradiction are tedious after you've seen a few. We really would like proofs that build on known unsolvable problems to show other, open problems are unsolvable. The technique commonly used is called reduction. It starts with some known unsolvable problem and then shows that this problem is no harder than some open problem in which we are interested.

PROBLEM CATEGORIES

RECURSIVE (SOLVABLE)

LOTS OF EXAMPLES

RE, NON-RECURSIVE (UNDEC BUT SEMI DEC)

$HALT = \{ \langle f, x \rangle \mid \varphi_f(x) \downarrow \}$

SHOWN BY DIAGONALIZATION

NON-RE (CANNOT EVEN SEMI-DECIDE)

$TOTAL = \{ f \mid \forall x \; \varphi_f(x) \downarrow \}$

PROBLEM CATEGORIES

RECURSIVE (SOLVABLE)

LOTS OF EXAMPLES

RE, NON-RECURSIVE (UNDEC BUT SEMI DEC)

$\text{HALT} = \{\langle f, x \rangle \mid \varphi_f(x)\downarrow\}$

SHOWN BY DIAGONALIZATION

NON-RE (CANNOT EVEN SEMI-DECIDE)

$\text{TOTAL} = \{f \mid \forall x \ \varphi_f(x)\downarrow\}$

INTRO TO REDUCTION

$A \leq_m B$ IF THERE EXISTS SOME COMPUTABLE ALGORITHM $f \ni$

$x \in A \Leftrightarrow f(x) \in B$

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 OR f's 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 UNKOWN COMPLEXITY

LET A BE SOME SET OF KNOWN COMPLEXITY

LET f BE A COMPUTABLE M-1 FUNCTION (TOTAL)

$$A \leq_m B \text{ OR JUST } A \leq B$$

VIA f IF

$x \in A$ IFF $f(x) \in B$

IF A IS RE, NON-REC. THEN B IS NON-REC., BUT NOT NEC. RE

IF A IS NON-RE THEN B IS NON-RE AND, OF COURSE, NON-REC.

REDUCTION EXAMPLE #1

SHOW HALT $\leq$ TOTAL

LET f, x BE ARBITRARY NAT. NUMBERS

$\langle f,x \rangle \in$ HALT IFF $\Phi_f(x)\downarrow$

DEFINE F_x BY

$\forall y \, F_x(y) = \Phi_f(x)$ // IGNORES INPUT

NOW

$\langle f,x \rangle \in$ HALT IFF $F_x \in$ TOTAL

THUS, HALT $\leq_M$ TOTAL

BY THIS, 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 Φ AND JUST SAY

$\forall y \, F_x(y) = f(x)$ // OVERLOAD

FOR CONVENIENCE

EXAMPLE #2

HASZERO $= \{ f \mid \exists x\ f(x) = 0 \}$ //SKIP φ

SHOW HASZERO IS NON-REC.

LET f, 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

$\langle f, x \rangle \in$ HALT $\Leftrightarrow F_x \in$ HASZERO

THUS, HASZERO IS NON-REC. SINCE

HALT $\leq_m$ HASZERO

BUT IS HASZERO 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

$$
\begin{aligned}
f \in \text{TOTAL} \quad & \text{IFF } \forall x\, f(x)\downarrow \\
& \text{IFF } \forall x\; h(x) = 0 \\
& \text{IFF } h \in \text{ZERO}
\end{aligned}
$$

THUS,

$$\text{TOTAL} \leq_m \text{ZERO}$$

AND SO ZERO IS NON-RE

EXAMPLE #4

$\text{IDENTITY} = \{ f \mid \forall x\, f(x) = x \}$

LET f BE AN ARBITRARY INDEX

DEFINE

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

NOW

$$f \in \text{TOTAL} \text{ IFF } \forall x\, f(x)\downarrow$$
$$\text{IFF } \forall x\, g_f(x) = x$$
$$\text{IFF } g_f \in \text{IDENTITY}$$

THUS,

$$\text{TOTAL} \le_m \text{IDENTITY}$$

AND SO IDENTITY IS NOT EVEN RE

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 $T \leq S$

HALT (AKA K_0) IS RE-COMPLETE

LET A BE ARB RE SET THEN

$A = \mathrm{Dom}\,\varphi_a$ FOR SOME INDEX a

HERE $A = W_a$ (ENUMERATION THEOREM)

$$x \in A \iff x \in \mathrm{Dom}(\varphi_a) \iff \varphi_a(x)\downarrow \iff \langle a, x\rangle \in K_0$$

THUS

$A \leq K_0$ (REALLY $A \leq_1 K_0$)

K IS ALSO RE-COMPLETE

$K = \{f \mid \varphi_f(f)\downarrow\}$

LET f, x BE ARB.

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

$\langle f, x\rangle \in HALT(K_0) \Leftrightarrow$

$x \in DOM(f) \Leftrightarrow$

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

$\Rightarrow F_x \in K$

$\langle f, x\rangle \notin HALT(K_0) \Leftrightarrow$

$x \notin DOM(f) \Leftrightarrow$

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

$\Rightarrow F_x \notin K$

THUS,

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

BUT K IS OBVIOUSLY RE

AND SO K IS RE-COMPLETE