Algorithms and Procedures

Procedures are just programs or processes that are clearly computable.

Procedures need not halt for all input.

Algorithms are procedures that halt on all input.

Predicates are procedures that produce answers (output) true/false (yes/no, 1/0).

Decision problems are problems where each instance has a true/false answer.

Notation:

$f(x)\downarrow$ means procedure $f$ halts (gives an output) when evaluated at $x$.

$f(x)\uparrow$ means procedure $f$ diverges when evaluated at $x$.

If $f$ is an algorithm then $\forall x f(x)\downarrow$. 
SOLVED

UNSOLVED

TEMPORAL

SOLVABLE
(RECURSIVE)
(DECIDABLE)

UNSOLVABLE
(NON-RECURSIVE)
(UNDECIDABLE)

INHERENT

SEMI-DECIDABLE
(ENUMERABLE)
(GENERABLE)
(RECURSIVELY ENUMERABLE)
(RE)

NOT SEMI-DECIDABLE
(NON-ENUMERABLE)
(NON-GENERABLE)
(NON-RE)
Examples

Fermat's Last Theorem is solved and always was solvable

\[ \exists n \geq 2, a, b, c \left[ a^n + b^n = c^n \right] \]
where \( n, a, b, c \) are positive natural numbers

Decision for Polynomial Time Problems versus Non-Det Polynomial Time Ones

\( P \neq NP \)

is unsolved but solvable

Property to determine, for arbitrary program, \( P \), and input, \( x \), whether or not \( P(x) \) eventually halts is unsolvable but semi-decidable

Note: \( P(x) \downarrow \) means \( P(x) \) converges
More Examples

Property to determine, for arbitrary program $P$, whether or not
$$\forall x \ P(x) \Downarrow$$ is unsolvable and not even semi-decidable.

Property to determine, for arbitrary polynomial $P(x_1, \ldots, x_n)$ whether or not
$$\exists x_1, \ldots, x_n \ [P(x_1, \ldots, x_n) = 0]$$
is unsolvable but RE (semi-decidable) can do a highly structured search (breadth first) for a solution.

Limit on algorithms to a countable set means there must be some that cannot be solved or even enumerated.
HALTING PROBLEM

Assume we can decide for arbitrary procedure, \( P \), and input \( x \), whether or not \( P(x) \uparrow \).

Define \( D(x) = \begin{cases} \text{while } P_x(x); \text{return}(1); \end{cases} \) if there is a predicate \( P_x(x) \) that halts (i.e., \( P_x(x) \uparrow \)) for every input \( x \).

Since \( \text{HALT} \) is an algorithm, it is a procedure and hence so is \( D \). Assume \( D = P_d \) then:

- \( D(d) \uparrow \iff P_d(d) \downarrow \iff \text{HALT}(d,d) = \text{true} \iff P_d(d) \uparrow \iff D(d) \uparrow \)

Lazy Evaluation
ASSUME HALTING PROBLEM IS SOLVABLE

\[ H(p,x) = \begin{cases} 
\text{YES} & \text{if } p(x) \downarrow \\
\text{NO} & \text{if } p(x) \uparrow 
\end{cases} \]

\[ D(x) = \begin{cases} 
\text{IF } H(x,x) = \text{NO} \\
\text{IF } H(x,x) = \text{YES} 
\end{cases} \]

\[ D(x) : \quad \text{while } \neg H(x,x) \]

D is a program then \( D = P_d \) for some \( d \)

\[ D(d) \downarrow \iff H(d,d) = \text{NO} \iff P_d(d) \uparrow \iff D(d) \uparrow \]
WHAT CONTRADICTION REQUIRED
TO TALK ABOUT \( P_0, P_1, \ldots \)
WE NEED THAT WE CAN EFFECTIVELY
MAP NATURAL NUMBERS TO PROGRAMS
IN CASE OF PROGRAMMING LANGUAGES
THAT IS JUST SORTING THEM, OR
REALLY SORTING SYNTACTICALLY CORRECT
PROGRAMS.

To use notation \( \mu y \[ y = y + 1 \] \)
we just need a control construct
that can run forever.

Could define \( D \) by

\[
D (i) \{ \\
\quad \text{while } (\text{HALT}(i, i)) ;\\
\quad \text{return } 0 ;
\}
\]

Assuming predicate \( \text{HALT EXISTS} \)
where \( \text{HALT}(i, i) \) IFF \( P_i (i) \downarrow \)
TURING MACHINE

Can read character in scanned cell (say $x$) and based on current state (say $q$) and $x$ ($q_x$ is called the discriminant) can either
(a) rewrite $x$ as some other symbol,
(b) move right; or
(c) move left
and then change state.

Note: This is really Post's notation $q \times R \rightarrow \ldots \rightarrow q \Delta \Rightarrow \ldots \Rightarrow s \Delta$
TM TAPE

Unmarked parts of tape are blank.
Tape starts with input (finite)
at each stage it might move to
parts of tape never visited before
and may write new values.

Based on this, tape is always
finitely marked and only a finite
number of cells can be visited
in any finite period of time.

For our purposes, we will use
tape alphabet of \{0, 1\} and

\[ M = (Q, \Sigma, \Gamma, \delta, q_0, F, \text{blank}) \]

\[ \text{TABLE OF QUAD MAPPING} \]

\[ (Q \times \Sigma, 1) \rightarrow (Q \times \{0, 1\}, R, L) \]

Halting is just entering a state
\( q \) with input \( x \), where \( q \times x \) has no
entry in \( T \). So, really \( q \times x_0, 1 \rightarrow q \times \overline{x_0}, R, L, y \)
STANDARD TURING COMPUTING

\[ 01^x00 \ldots 01^x00 \]

\[ \Rightarrow a^x00 \ldots 01^x01^y00 \]

WHERE \( y = f(x_1, \ldots, x_n) \)

Composition is easy with this approach to computation

Sometimes called semi-unbounded tape

Note that tape is always finitely marked, can have instantaneous descr. (ID) as triplet \((L, R, \text{state})\)

WHERE \( L \) is # denoted by binary value of left of scanned square;
\( R \) is # denoted by right and scanned,
read right to left; state is state #
Typically done with finite string

$\alpha \gamma \beta$

$\alpha'$ is shortest string that encompasses all non-blanks left of scanned square

$\gamma$ is current state

$\beta$ is shortest string that encompasses all non-blanks right of scanned

If we have $\{0, 1\}$ tape alphabet

Then ID can be 3 numbers

$(L, R, \text{STATE})$

$L$ is $\alpha$, interpreted as binary number

$R$ is $\beta$, $1$'s

Note: If $\alpha = \gamma$ or $\beta = \gamma$, use 0

State is $\gamma$
**Register Machines**

**My Standard Computation**

\[
\begin{array}{cccc}
X_1 & \cdots & X_n & 0 \\
R_1 & \cdots & R_n & 0 \\
\Rightarrow & X_1 & \cdots & X_n \\
& R_1 & \cdots & R_n \\
\end{array}
\]

*WHERE* \( Y = f(x_1, \ldots, x_n) \)

*INC \_ij \_R* 

*DEC \_ij \_R* [p, z] 

**Although won't show, can mimic TM by storing ID in 3 registers**

**Can even create prologue that starts as above and creates that 3 register encoding**

**Can have epilogue that extracts answer from triple at end of simulation**

ID is just 

\(<R_1, R_2, \ldots, R_n, \text{state} > \text{ pairing} \)

or

\(<R_1, R_2, \ldots, P_{r_k}, \text{state} > \text{ pairing} \)
Factor Replacement System

$$3x^1 \ldots P_k \Rightarrow 2^y$$

$$y = f(x_1, \ldots, x_k)$$

$$\frac{2}{3} \text{ OR } 3x \Rightarrow 2x$$

$$\frac{2}{5} \text{ OR } 5x \Rightarrow 2x$$

$$3^x 5^y \Rightarrow 2^{x+y}$$

$$\frac{1}{3.5} \text{ OR } 15x \Rightarrow x$$

$$\frac{2}{3} \text{ OR } 3x \Rightarrow 2x$$

$$\frac{2}{5} \Rightarrow 2^n$$

$$3^x 5^y \Rightarrow 2^\max(x-y,0)$$

$$a_1/b_1$$

$$a_2/b_2$$

$$a_3/b_3$$

$$b_{k_1}x \Rightarrow a_{k_1}x$$

$$b_{k_2}x \Rightarrow a_{k_2}x$$
REGISTER TO FR S

REGISTERS

STORE A AS POWERS OF PRIMES

STORE STATE AS POWER OF NEXT PRIMES

PRESENCE IS EASY (R² > 0)

Pr

STATE CHECK IS EASY

Pn+3

WHERE A REGISTERS

\[
\begin{align*}
  m. \ & \text{DECK}[i,j]\mathcal{J} \\
  \text{ORDER} & \quad \begin{cases} 
    Pr Pn+m \rightarrow Pn+i \quad x \\
    Pn+m \rightarrow Pn+j \quad x
  \end{cases}
\end{align*}
\]

\[
\begin{align*}
  m. \ & \text{INC } k[i,j] \\
  Pn+m \rightarrow Pr Pn+i \quad x
\end{align*}
\]
Is Power Of 2

\[ 3^2 \times 5 \rightarrow 5 \times 7 \times \]
\[ 3 \times 5 \times \rightarrow x \]
\[ \text{CAME UP ODD} \]
\[ \text{CASE OF } 1 = 2^0 \]
\[ \text{WAS DIV BY 2} \]
\[ \text{GET READY TO DIVIDE BY 2 AGAIN} \]
\[ \text{GO UP TO DIVIDE BY 2} \]
\[ \text{CASE OF 0} \]
\[ \text{CLEAN UP ODD CASE} \]

\[ 3^2 \times 5 \rightarrow 7 \times 5 \rightarrow 7 \times 11 \rightarrow 3 \times 11 \rightarrow 3 \times 5 \]
\[ \cdots \]
\[ 3^1 \times 5 \rightarrow 2 = 2^1 \]

\[ 3^{x+1} \times 5 \rightarrow \cdots \]
\[ 3 \times 7 \times 5 \rightarrow 7 \times \rightarrow 1 \]
\[ 3 \times 5 \rightarrow 2 = 2^1 \]
\[ (3^1 \times 5) \]
\[ 5 \rightarrow 1 = 2^0 \]
\[ (2^0 \times 5) \]