Theory of Computation 11 Recursively Enumerable Sets

Frank Stephan

Department of Computer Science

Department of Mathematics

National University of Singapore

fstephan@comp.nus.edu.sg

Repetition 1

Models of Computation

- Turing Machine: States like Finite Automaton plus Turing tape carrying input/output and working space; head of machine working and moving on tape; updates of symbols, states and movement given by Turing table.
- Register Machine: Adding and Subtracting and Comparing natural numbers in registers; conditional and unconditional jumps between numbered statements.
- Primitive recursive and μ-recursive functions: Functions defined from some base functions together with concatenation, primitive recursion and, in the case of μ-recursive functions, search for places where some condition holds.

Repetition 2

Example: Multiplication can be done naively by repeated addition.

Line 1: Function Mult$(\mathbf{R_1}, \mathbf{R_2})$;
Line 2: $\mathbf{R_3} = \mathbf{0}$;
Line 3: $\mathbf{R_4} = \mathbf{0}$;
Line 4: If $\mathbf{R_3} = \mathbf{R_1}$ Then Goto Line 8;
Line 5: $\mathbf{R_4} = \mathbf{R_4} + \mathbf{R_2}$;
Line 6: $\mathbf{R_3} = \mathbf{R_3} + \mathbf{1}$;
Line 7: Goto Line 4;
Line 8: Return$(\mathbf{R_4})$.

Repetition 3

Primitive Recurisive: Addition, Multiplication, Subtraction, Exponentiation, Factorial, Choose-Function, Outcomes of Comparisons, Linear Functrions and Polynomials.

Not Primitive Recursive: Ackermann Function:

- $\mathbf{f(0, y) = y + 1}$;
- $\mathbf{f(x + 1, 0) = f(x, 1)}$;
- $\mathbf{f(x + 1, y + 1) = f(x, f(x + 1, y))}$.

Partial Recursive: Primitive recursive plus search for an input which makes function to $\mathbf{0}$.

Repetition 4

Theorem 10.23

For a partial function f, the following are equivalent:

- f as a function from strings to strings can be computed by a Turing machine;
- f as a function from natural numbers to natural numbers can be computed by a register machine;
- f as a function from natural numbers to natural numbers is partial recursive.

Church's Thesis

All reasonable models of computation over Σ^* and $\mathbb{N}$ are equivalent and give the same notion as the partial recursive functions.

Repetition 5

One measures the size $\mathbf{n}$ of the input in the number of its symbols or by $\mathbf{log(x) = \min\{n \in \mathbb{N} : x \leq 2^n\}}$.

Theorem 10.25

A function $\mathbf{f}$ is computable by a Turing machine in time $\mathbf{p(n)}$ for some polynomial $\mathbf{p}$ iff $\mathbf{f}$ is computable by a register machine in time $\mathbf{q(n)}$ for some polynomial $\mathbf{q}$.

Theorem 10.26

A function $\mathbf{f}$ is computable by a Turing machine in space $\mathbf{p(n)}$ for some polynomial $\mathbf{p}$ iff $\mathbf{f}$ is computable by a register machine in such a way that all registers take at most the value $\mathbf{2^{q(n)}}$ for some polynomial $\mathbf{q}$.

The notions in Complexity Theory are also relatively invariant against changes of the model of computation; however, one has to interpret the word “reasonable” of Church in a stronger way than in recursion theory.

Primitive Recursive

Theorem

A function is primitive recursive iff it can be computed by a register program where the only type of goto-commands which can go backwards are For-Loops, where one cannot go into or out of a For-Loop and once the For-Loop is started, its boundaries cannot be modified and the loop-variable can only be updated by the commands of the loop itself.

Remark

One can replace the Goto-commands completely by allowing only For-Loops, If-Then-Else statements and Switch-statements which are properly nested.

For full generality of Partial-Recursive functions, one would then also need While-Loops in addition to the For-Loops.

Example

Line 1: Function Factor$(\mathbf{R_1}, \mathbf{R_2})$;
Line 2: $\mathbf{R_3} = \mathbf{R_1}$;
Line 3: $\mathbf{R_4} = \mathbf{0}$;
Line 4: If $\mathbf{R_2} < \mathbf{2}$ Then Goto Line 10;
Line 5: For $\mathbf{R_5} = \mathbf{0}$ to $\mathbf{R_1}$
Line 6: If Remainder$(\mathbf{R_3}, \mathbf{R_2}) > \mathbf{0}$ Then Goto Line 9;
Line 7: $\mathbf{R_3} =$ Divide$(\mathbf{R_3}, \mathbf{R_2})$;
Line 8: $\mathbf{R_4} = \mathbf{R_4} + \mathbf{1}$;
Line 9: Next $\mathbf{R_5}$;
Line 10: Return$(\mathbf{R_4})$;

This function computes how often $\mathbf{R_2}$ is a factor of $\mathbf{R_1}$ and is primitive recursive.

Collatz Function

Not known whether primitive recursive or whether total at all.

Line 1: Function Collatz$(\mathbf{R_1})$;
Line 2: If Remainder$(\mathbf{R_1}, \mathbf{2}) = \mathbf{0}$ Then Goto Line 6;
Line 3: If $\mathbf{R_1} = \mathbf{1}$ Then Goto Line 8;
Line 4: $\mathbf{R_1} =$ Mult$(\mathbf{R_1}, \mathbf{3}) + \mathbf{1}$;
Line 5: Goto Line 2;
Line 6: $\mathbf{R_1} =$ Divide$(\mathbf{R_1}, \mathbf{2})$;
Line 7: Goto Line 2;
Line 8: Return$(\mathbf{R_1})$;

Lothar Collatz conjectured in 1937 that this function is total.

Simulating Collatz Function

Line 1: Function Collatz$(\mathbf{R_1}, \mathbf{R_2})$;
Line 2: $\mathbf{LN} = \mathbf{2}$;
Line 3: For $\mathbf{T} = \mathbf{0}$ to $\mathbf{R_2}$
Line 4: If $\mathbf{LN} = \mathbf{2}$ Then Begin If Remainder$(\mathbf{R_1}, \mathbf{2}) = \mathbf{0}$
Then $\mathbf{LN} = \mathbf{6}$ Else $\mathbf{LN} = \mathbf{3}$; Goto Line 10 End;
Line 5: If $\mathbf{LN} = \mathbf{3}$ Then Begin If $\mathbf{R_1} = \mathbf{1}$ Then $\mathbf{LN} = \mathbf{8}$
Else $\mathbf{LN} = \mathbf{4}$; Goto Line 10 End;
Line 6: If $\mathbf{LN} = \mathbf{4}$ Then Begin $\mathbf{R_1} =$ Mult$(\mathbf{R_1}, \mathbf{3}) + \mathbf{1}$;
$\mathbf{LN} = \mathbf{5}$; Goto Line 10 End;
Line 7: If $\mathbf{LN} = \mathbf{5}$ Then Begin $\mathbf{LN} = \mathbf{2}$; Goto Line 10 End;
Line 8: If $\mathbf{LN} = \mathbf{6}$ Then Begin $\mathbf{R_1} =$ Divide$(\mathbf{R_1}, \mathbf{2})$;
$\mathbf{LN} = \mathbf{7}$; Goto Line 10 End;
Line 9: If $\mathbf{LN} = \mathbf{7}$ Then Begin $\mathbf{LN} = \mathbf{2}$; Goto Line 10 End;
Line 10: Next $\mathbf{T}$;
Line 11: If $\mathbf{LN} = \mathbf{8}$ Then Return$(\mathbf{R_1} + \mathbf{1})$ Else Return$(\mathbf{0})$.

Exercise 11.1

Write a program for a primitive recursive function which simulate the following function with input $\mathbf{R_1}$ for $\mathbf{R_2}$ steps.

Line 1: Function Expo$(\mathbf{R_1})$;
Line 2: $\mathbf{R_3 = 1}$;
Line 3: If $\mathbf{R_1 = 0}$ Then Goto Line 7;
Line 4: $\mathbf{R_3 = R_3 + R_3}$;
Line 5: $\mathbf{R_1 = R_1 - 1}$;
Line 6: Goto Line 3;
Line 7: Return$(\mathbf{R_3})$.

Exercise 11.2

Write a program for a primitive recursive function which simulate the following function with input $\mathbf{R_1}$ for $\mathbf{R_2}$ steps.

Line 1: Function Repeatadd$(\mathbf{R_1})$;
Line 2: $\mathbf{R_3 = 3}$;
Line 3: If $\mathbf{R_1 = 0}$ Then Goto Line 7;
Line 4: $\mathbf{R_3 = R_3 + R_3 + R_3 + 3}$;
Line 5: $\mathbf{R_1 = R_1 - 1}$;
Line 6: Goto Line 3;
Line 7: Return$(\mathbf{R_3})$.

Bounded Simulation

Theorem 11.3

For every partial-recursive function $\mathbf{f}$ there is a primitive recursive function $\mathbf{g}$ and a register machine $\mathbf{M}$ such that for all $\mathbf{t}$,

> If $\mathbf{f}(\mathbf{x_1}, \ldots, \mathbf{x_n})$ is computed by $\mathbf{M}$ within $\mathbf{t}$ steps
> Then $\mathbf{g}(\mathbf{x_1}, \ldots, \mathbf{x_n}, \mathbf{t}) = \mathbf{f}(\mathbf{x_1}, \ldots, \mathbf{x_n}) + \mathbf{1}$
> Else $\mathbf{g}(\mathbf{x_1}, \ldots, \mathbf{x_n}, \mathbf{t}) = \mathbf{0}$.

In short words, $\mathbf{g}$ simulates the program $\mathbf{M}$ of $\mathbf{f}$ for $\mathbf{t}$ steps and if an output $\mathbf{y}$ comes then $\mathbf{g}$ outputs $\mathbf{y} + \mathbf{1}$ else $\mathbf{g}$ outputs $\mathbf{0}$.

Recursively Enumerable

Theorem 11.4

The following notions are equivalent for a set $\mathbf{A} \subseteq \mathbb{N}$:

(a) $\mathbf{A}$ is the range of a partial recursive function;

(b) $\mathbf{A}$ is empty or $\mathbf{A}$ is the range of a total recursive function;

(c) $\mathbf{A}$ is empty or $\mathbf{A}$ is the range of a primitive recursive function;

(d) $\mathbf{A}$ is the set of inputs on which some register machine terminates;

(e) $\mathbf{A}$ is the domain of a partial recursive function;

(f) There is a two-place recursive function $\mathbf{g}$ such that $\mathbf{A} = \{\mathbf{x} : \exists \mathbf{y}\, [\mathbf{g}(\mathbf{x}, \mathbf{y}) > \mathbf{0}]\}$.

Definition 11.5

The set $\mathbf{A}$ is recursively enumerable iff it satisfies any of the above equivalent properties.

(a) to (c) and (c) to (b)

If $\mathbf{A}$ is empty then $(\mathbf{c})$ holds; if $\mathbf{A}$ is not empty then there is an element $\mathbf{a} \in \mathbf{A}$ which is now taken as a constant. For the partial function $\mathbf{f}$ whose range $\mathbf{A}$ is, there is, by Theorem 11.3, a primitive function $\mathbf{g}$ such that either $\mathbf{g(x,t) = 0}$ or $\mathbf{g(x,t) = f(x) + 1}$ and whenever $\mathbf{f(x)}$ takes a value there is also a $\mathbf{t}$ with $\mathbf{g(x,t) = f(x) + 1}$. Now one defines a new function $\mathbf{h}$ which is also primitive recursive such that if $\mathbf{g(x,t) = 0}$ then $\mathbf{h(x,t) = a}$ else $\mathbf{h(x,t) = g(x,t) - 1}$. The range of $\mathbf{h}$ is $\mathbf{A}$.

$(\mathbf{c}) \Rightarrow (\mathbf{b})$: This follows by definition as every primitive recursive function is also recursive.

(b) to (d) and (d) to (e)

$(\mathbf{b}) \Rightarrow (\mathbf{d})$: Given a function $\mathbf{h}$ whose range is $\mathbf{A}$, one can make a register machine which simulates $\mathbf{h}$ and searches over all possible inputs and checks whether $\mathbf{h}$ on these inputs is $\mathbf{x}$. If such inputs are found then the search terminates else the register machine runs forever. Thus $\mathbf{x} \in \mathbf{A}$ iff the register machine program following this behaviour terminates after some time.

$(\mathbf{d}) \Rightarrow (\mathbf{e})$: The domain of a register machine is the set of inputs on which it halts and outputs a return value. Thus this implication is satisfied trivially by taking the function for $(\mathbf{e})$ to be exactly the function computed from the register program for $(\mathbf{d})$.

(e) to (f) and (f) to (a)

$(\mathbf{e}) \Rightarrow (\mathbf{f})$: Given a register program $\mathbf{f}$ whose domain $\mathbf{A}$ is according to $(\mathbf{e})$, one takes the function $\mathbf{g}$ as defined by Theorem 11.3 and this function indeed satisfies that $\mathbf{f}(\mathbf{x})$ is defined iff there is a $\mathbf{t}$ such that $\mathbf{g}(\mathbf{x}, \mathbf{t}) > \mathbf{0}$.

$(\mathbf{f}) \Rightarrow (\mathbf{a})$: Given the function $\mathbf{g}$ as defined in $(\mathbf{f})$, one defines that if there is a $\mathbf{t}$ with $\mathbf{g}(\mathbf{x}, \mathbf{t}) > \mathbf{0}$ then $\mathbf{f}(\mathbf{x}) = \mathbf{x}$ else $\mathbf{f}(\mathbf{x})$ is undefined. The latter comes by infinite search for a $\mathbf{t}$ which is not found. Thus the partial recursive function $\mathbf{f}$ has range $\mathbf{A}$.

Decidable and Undecidable Problems

A set $\mathbf{L}$ is called decidable or recursive iff there is a recursive function $\mathbf{f}$ such that, for all $\mathbf{x}$, if $\mathbf{x} \in \mathbf{L}$ then $\mathbf{f(x) = 1}$ else $\mathbf{f(x) = 0}$. One says that the function $\mathbf{f}$ decides the membership in $\mathbf{L}$.

A set $\mathbf{L}$ is called undecidable or nonrecursive iff there is no such recursive function $\mathbf{f}$ deciding the membership in $\mathbf{L}$.

Observation

Every recursive set is recursively enumerable.

The Halting Problem

Definition [Turing 1936]
Let $\mathbf{e}, \mathbf{x} \mapsto \varphi_{\mathbf{e}}(\mathbf{x})$ be a universal partial recursive function covering all one-variable partial recursive functions. Then the set $\{(\mathbf{e}, \mathbf{x}) : \varphi_{\mathbf{e}}(\mathbf{x}) \text{ is defined}\}$ is called the general halting problem and $\mathbf{K} = \{\mathbf{e} : \varphi_{\mathbf{e}}(\mathbf{e}) \text{ is defined}\}$ is called the diagonal halting problem.

The name stems from the fact that $\varphi_{\mathbf{e}}(\mathbf{x})$ is defined iff the $\mathbf{e}$-th register machine with input $\mathbf{x}$ halts and produces some output.

Theorem [Turing 1936]
Both the diagonal halting problem and the general halting problem are recursively enumerable and undecidable.

Proof

Let F be a function which simulates $\varphi_{\mathbf{e}}(\mathbf{x})$ and assume that there is a function Halt which can check whether $\varphi_{\mathbf{e}}(\mathbf{e})$ halts. If so, then $\text{Halt}(\mathbf{e}) = \mathbf{1}$ else $\text{Halt}(\mathbf{e}) = \mathbf{0}$. Now condider this program.

Line 1: Function Diagonalise$(\mathbf{R_1})$;
Line 2: $\mathbf{R_2} = \mathbf{0}$;
Line 3: If $\text{Halt}(\mathbf{R_1}) = \mathbf{0}$ Then Goto Line 5;
Line 4: $\mathbf{R_2} = \text{F}(\mathbf{R_1}, \mathbf{R_1}) + \mathbf{1}$;
Line 5: Return$(\mathbf{R_2})$.

Function Diagonalise

The function Diagonalise has only one input.

If $\varphi_{\mathbf{e}}(\mathbf{e})$ is undefined then Halt$(\mathbf{e}) = \mathbf{0}$ and Diagonalise$(\mathbf{e}) = \mathbf{0}$.

If $\varphi_{\mathbf{e}}(\mathbf{e})$ is defined then Halt$(\mathbf{e}) = \mathbf{1}$ and F$(\mathbf{e}, \mathbf{e}) = \varphi_{\mathbf{e}}(\mathbf{e})$ will be computed in Line 4 and the output will be $\varphi_{\mathbf{e}}(\mathbf{e}) + \mathbf{1}$.

Thus Diagonalise$(\mathbf{e})$ differs from $\varphi_{\mathbf{e}}(\mathbf{e})$ for all $\mathbf{e}$ and is not among $\varphi_{\mathbf{0}}, \varphi_{\mathbf{1}}, \ldots$; as all partial-recursive functions with one input are in this list, Diagonalise cannot be recursive and therefore Halt also cannot be recursive.

The halting problem equals $\{(\mathbf{e}, \mathbf{x}) : \mathbf{F}(\mathbf{e}, \mathbf{x})$ halts$\}$. Thus it is the domain of a partial recursive function and recursively enumerable. Similarly, $\mathbf{K} = \{\mathbf{e} : \mathbf{F}(\mathbf{e}, \mathbf{e})$ halts$\}$ is the domain of a partial-recursive function and recursively enumerable.

R.E. and Recursive

Theorem 11.9

A set $\mathbf{L}$ is recursive iff both $\mathbf{L}$ and $\mathbb{N} - \mathbf{L}$ are recursively enumerable.

Exercise 11.10

Prove this connection.

Exercises 11.11 – 11.13

Prove that the following variants of the halting problem are undecidable:

11.11: $\{\mathbf{e} : \varphi_{\mathbf{e}}(\mathbf{2e} + \mathbf{5})$ is defined$\}$;

11.12: $\{\mathbf{e} : \varphi_{\mathbf{e}}(\mathbf{e^2} + \mathbf{1})$ is defined$\}$;

11.13: $\{\mathbf{e} : \varphi_{\mathbf{e}}(\mathbf{e}/\mathbf{2})$ is defined$\}$, where $\mathbf{1}/\mathbf{2}$ is rounded to $\mathbf{0}$ and $\mathbf{3}/\mathbf{2}$ to $\mathbf{1}$ and so on.

Further Homeworks 11.14-11.16

Show that the following sets are recursively enumerable by proving that a register machine halts exactly on the members of the set:

Exercise 11.14: $\{\mathbf{x} \in \mathbb{N} : \mathbf{x}$ is a square$\}$.

Exercise 11.15: $\{\mathbf{x} \in \mathbb{N} : \mathbf{x}$ is prime$\}$.

Exercise 11.16

Prove that the set $\{\mathbf{e} : \varphi_{\mathbf{e}}(\mathbf{e}/\mathbf{2})$ is defined$\}$ is recursively enumerable by proving that it is the range of a primitive recursive function. Here $\mathbf{e}/\mathbf{2}$ is the downrounded value of $\mathbf{e}$ divided by $\mathbf{2}$, so $\mathbf{1}/\mathbf{2}$ is $\mathbf{0}$ and $\mathbf{3}/\mathbf{2}$ is $\mathbf{1}$.

Further Homeworks 11.17-11.19

Exercise 11.17: Prove or disprove: Every recursively enumerable set is either $\emptyset$ or the range of a function which can be computed in polynomial time.

Exercise 11.18: Prove or disprove: Every recursively enumerable set is either $\emptyset$ or the domain of a function $\mathbf{f}$ where the graph $\{(\mathbf{x}, \mathbf{f}(\mathbf{x})) : \mathbf{x} \in \mathbf{dom}(\mathbf{f})\}$ can be decided in polynomial time, that is, given inputs $\mathbf{x}, \mathbf{y}$, one can decide in polynomial time whether $(\mathbf{x}, \mathbf{y}) = (\mathbf{x}, \mathbf{f}(\mathbf{x}))$.

Exercise 11.19: Prove or disprove: Every recursively enumerable set is either $\emptyset$ or the domain of a $\{\mathbf{0}, \mathbf{1}\}$-valued function $\mathbf{f}$ where the graph $\{(\mathbf{x}, \mathbf{f}(\mathbf{x})) : \mathbf{x} \in \mathbf{dom}(\mathbf{f})\}$ can be decided in polynomial time.

Second Half of Lecture: Midterm Test