# Computability Theory I

## Introduction

Guoqiang Li

Shanghai Jiao Tong University

Sep. 19, 2014

# Instructor and Teaching Assistant

- Guoqiang LI

# Instructor and Teaching Assistant

- Guoqiang LI
  - Homepage: http://basics.sjtu.edu.cn/∼liguoqiang
  - Course page: http://basics.sjtu.edu.cn/∼liguoqiang/teaching/comp14/index.htm
  - Email: li-gq@cs.sjtu.edu.cn
  - Office: Rm. 1212, Building of Software
  - Phone: 3420-4167

# Instructor and Teaching Assistant

- Guoqiang LI
  - Homepage: http://basics.sjtu.edu.cn/∼liguoqiang
  - Course page: http://basics.sjtu.edu.cn/∼liguoqiang/teaching/comp14/index.htm
  - Email: li-gq@cs.sjtu.edu.cn
  - Office: Rm. 1212, Building of Software
  - Phone: 3420-4167
  - YSSY: many IDs…
  - weibo: http://www.weibo.com/flyinsail
- TA:

# Instructor and Teaching Assistant

- Guoqiang LI
  - Homepage: http://basics.sjtu.edu.cn/∼liguoqiang
  - Course page: http://basics.sjtu.edu.cn/∼liguoqiang/teaching/comp14/index.htm
  - Email: li-gq@cs.sjtu.edu.cn
  - Office: Rm. 1212, Building of Software
  - Phone: 3420-4167
  - YSSY: many IDs…
  - weibo: http://www.weibo.com/flyinsail
- TA:
  - Mingzhang HUANG: mingzhanghuang@gmail.com
  - Xiuting TAO: taoxiuting@gmail.com
- Office hour: Wed. 14:00-17:00 @ SEIEE 3-327

What do you think you can learn from this course?

# Aim of the Course

- Q: Can the course improve the skill of programming?

# Aim of the Course

- Q: Can the course improve the skill of programming?
  - A: Nope!

# Aim of the Course

- Q: Can the course improve the skill of programming?
  - A: Nope!
- Q: Can the course improve the ability of algorithms?

# Aim of the Course

- Q: Can the course improve the skill of programming?
  - A: Nope!
- Q: Can the course improve the ability of algorithms?
  - A: Perhaps, seldom.

# Aim of the Course

- Q: Can the course improve the skill of programming?
  - A: Nope!
- Q: Can the course improve the ability of algorithms?
  - A: Perhaps, seldom.
- The course may provide a view of computation, an overlook of what we are doing in computer science, and a basic study of theoretical computer science.

# Aim of the Course

- Q: Can the course improve the skill of programming?
  - A: Nope!
- Q: Can the course improve the ability of algorithms?
  - A: Perhaps, seldom.
- The course may provide a view of computation, an overlook of what we are doing in computer science, and a basic study of theoretical computer science.
- It is rather a philosophy than a technique, although some parts are quite technically.

# It May Answer

# It May Answer

- A software company that is developing a compiler capable of checking if a program contains a loop.

# It May Answer

- A software company that is developing a compiler capable of checking if a program contains a loop.
- A hardware company that is determined to design a computer that can solve problems that no existing computers can solve.

# It May Answer

- A software company that is developing a compiler capable of checking if a program contains a loop.
- A hardware company that is determined to design a computer that can solve problems that no existing computers can solve.
- A service provider that is working on a theorem prover that is supposed to answer every question about numbers.

# History of Velocity

Human beings are keen on speed, and cannot stop the step to chase moving as fast as possible.

# History of Velocity

Human beings are keen on speed, and cannot stop the step to chase moving as fast as possible.

- Wheels: Mid-4th millennium BC.

# History of Velocity

Human beings are keen on speed, and cannot stop the step to chase moving as fast as possible.

- Wheels: Mid-4th millennium BC.
- Automobiles: 1762

# History of Velocity

Human beings are keen on speed, and cannot stop the step to chase moving as fast as possible.

- Wheels: Mid-4th millennium BC.
- Automobiles: 1762
- Trains: 1807
- Airplanes: 1903

# History of Velocity

Human beings are keen on speed, and cannot stop the step to chase moving as fast as possible.

- Wheels: Mid-4th millennium BC.
- Automobiles: 1762
- Trains: 1807
- Airplanes: 1903
- Supersonic: 1947 $343.2 m/s$

# History of Velocity

Human beings are keen on speed, and cannot stop the step to chase moving as fast as possible.

- Wheels: Mid-4th millennium BC.
- Automobiles: 1762
- Trains: 1807
- Airplanes: 1903
- Supersonic: 1947 $343.2m/s$
- Circular velocity 1957 $7.9km/s$
- Earth escape velocity 1959 $11.2km/s$
- Solar system escape velocity 1977 $16.7km/s$

# History of Velocity

Human beings are keen on speed, and cannot stop the step to chase moving as fast as possible.

- Wheels: Mid-4th millennium BC.
- Automobiles: 1762
- Trains: 1807
- Airplanes: 1903
- Supersonic: 1947 $343.2m/s$
- Circular velocity 1957 $7.9km/s$
- Earth escape velocity 1959 $11.2km/s$
- Solar system escape velocity 1977 $16.7km/s$
- Q: Can we achieve in arbitrarily fast velocity?

# History of Velocity

Human beings are keen on speed, and cannot stop the step to chase moving as fast as possible.

- Wheels: Mid-4th millennium BC.
- Automobiles: 1762
- Trains: 1807
- Airplanes: 1903
- Supersonic: 1947 $343.2m/s$
- Circular velocity 1957 $7.9km/s$
- Earth escape velocity 1959 $11.2km/s$
- Solar system escape velocity 1977 $16.7km/s$
- Q: Can we achieve in arbitrarily fast velocity?
    - Grandfather paradox

# Computation

Human beings are also keen on computation, and cannot stop the step to chase computing as complex as possible.

# Computation

Human beings are also keen on computation, and cannot stop the step to chase computing as complex as possible.

- Decimal system: AD 600

# Computation

Human beings are also keen on computation, and cannot stop the step to chase computing as complex as possible.

- Decimal system: AD 600
- Basic arithmetic: Al Khwarizmi (780 - 850)

# Computation

Human beings are also keen on computation, and cannot stop the step to chase computing as complex as possible.

- Decimal system: AD 600
- Basic arithmetic: Al Khwarizmi (780 - 850)
- ENIAC: 1946

# Computation

Human beings are also keen on computation, and cannot stop the step to chase computing as complex as possible.

- Decimal system: AD 600
- Basic arithmetic: Al Khwarizmi (780 - 850)
- ENIAC: 1946
- NP problem
- The curse of exponential time

# Computation

Human beings are also keen on computation, and cannot stop the step to chase computing as complex as possible.

- Decimal system: AD 600
- Basic arithmetic: Al Khwarizmi (780 - 850)
- ENIAC: 1946
- NP problem
- The curse of exponential time
- Advanced algorithms: simplex, DPLL, antichain.

# Computation

Human beings are also keen on computation, and cannot stop the step to chase computing as complex as possible.

- Decimal system: AD 600
- Basic arithmetic: Al Khwarizmi (780 - 850)
- ENIAC: 1946
- NP problem
- The curse of exponential time
- Advanced algorithms: simplex, DPLL, antichain.
- Q: Can we achieve in arbitrarily complex computation?

What problems can be solved by computers?

*Computer science is no more about computers than astronomy is about telescopes.*

*Edsger Dijkstra*

*Let us begin to learn some basic astronomical phenomena!*

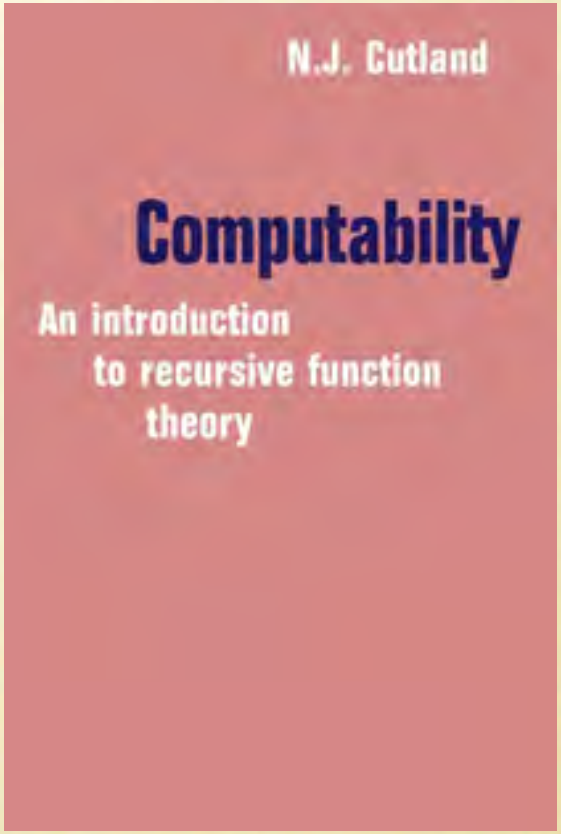
*The technique part is quite similar to puzzles of wise men.*

*So, please have a fun!*

*Intuition is extremely important!*

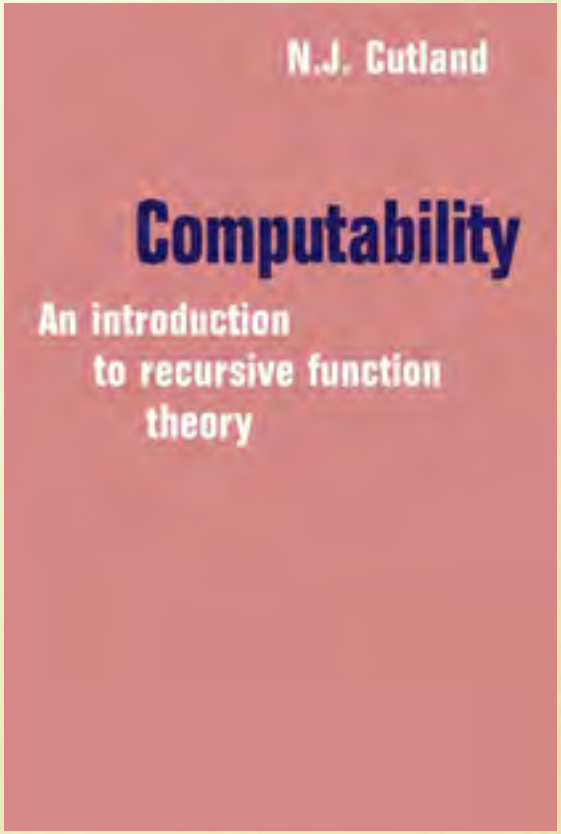
# Reference Book

- Computability: An Introduction to Recursive Function Theory.
  - Nigel J. Cutland



# Reference Book

- Computability: An Introduction to Recursive Function Theory.
  - Nigel J. Cutland
- plus extra reading materials.



# Scoring Policy

- 10% Attendance.
- 20% Assignments.
- 70% Final exam.

# Scoring Policy

- 10% Attendance.
- 20% Assignments.
  - Four assignments.

- 70% Final exam.

# Scoring Policy

- 10% Attendance.
- 20% Assignments.
  - Four assignments.
  - Each one is 5 pts.
- 70% Final exam.

# Scoring Policy

- 10% Attendance.
- 20% Assignments.
  - Four assignments.
  - Each one is 5 pts.
  - Work out individually.
- 70% Final exam.

# Scoring Policy

- 10% Attendance.
- 20% Assignments.
  - Four assignments.
  - Each one is 5 pts.
  - Work out individually.
- 70% Final exam.
- There are also several homework. The answer may be given in exercise lectures, two or three times.

# Special Requirements

A notebook and a pen.

Any questions?

# 0. Prologue

Effective Solutions

What problems can be solved by computers?

# Famous Problems

- Diophantine equations
- Shortest path problem
- Travelling salesman problem (TSP)
- Graph isomorphism problem (GI)

# Intuition

An effective procedure consists of a finite set of instructions which, given an input from some set of possible inputs, enables us to obtain an output through a systematic execution of the instructions that terminates in a finite number of steps.

# Intuition

# Intuition

Theorem proving is in general not effective.

Proof verification is effective.

# Intuition

Theorem proving is in general not effective.

Proof verification is effective.

Unbounded search is in general not effective.

Bounded search is effective.

# Representation of Problem

- How does a computer solve the GI problem or the TSP Problem?

# Representation of Problem

- How does a computer solve the GI problem or the TSP Problem?
- How is a problem instance (a graph) represented in a computer?

# Representation of Problem

- How does a computer solve the GI problem or the TSP Problem?
- How is a problem instance (a graph) represented in a computer?
- How is the answer to a problem instance represented?

# Representation of Problem

- How does a computer solve the GI problem or the TSP Problem?
- How is a problem instance (a graph) represented in a computer?
- How is the answer to a problem instance represented?
- How is an effective procedure formalized?

# Representation of Problem

- How does a computer solve the GI problem or the TSP Problem?
- How is a problem instance (a graph) represented in a computer?
- How is the answer to a problem instance represented?
- How is an effective procedure formalized?

- Can every function from $\mathbb{N}$ to $\mathbb{N}$ be calculated by a C program?

# Representation of Problem

- How does a computer solve the GI problem or the TSP Problem?
- How is a problem instance (a graph) represented in a computer?
- How is the answer to a problem instance represented?
- How is an effective procedure formalized?

- Can every function from $\mathbb{N}$ to $\mathbb{N}$ be calculated by a C program?
  - Negative.

# Punchline

- In a formal theory of computability, every problem instance can be represented by a number and every number represents a problem instance.

# Punchline

- In a formal theory of computability, every problem instance can be represented by a number and every number represents a problem instance.
- A problem is a function $f : \mathbb{N} \to \mathbb{N}$ from numbers to numbers.

# Punchline

- In a formal theory of computability, every problem instance can be represented by a number and every number represents a problem instance.
- A problem is a function $f : \mathbb{N} \to \mathbb{N}$ from numbers to numbers.
- A problem is computable if it can be calculated by a program.

*Everything is number!*

*Pythagoras*

Decision Problem

# Decision Problem

A problem $f : \mathbb{N} \to \mathbb{N}$ is a decision problem if the range $ran(f)$ of $f$ is $\{0, 1\}$, where $1$ denotes a 'yes' answer and $0$ a 'no' answer.

A decision problem $g$ can be identified with the set $\{n \mid g(n) = 1\}$.

Conversely a subset $A$ of $\mathbb{N}$ can be seen as a decision problem via the characteristic function of $A$:

$$c_A(n) = \begin{cases} 1, & \text{if } x \in A, \\ 0, & \text{otherwise.} \end{cases}$$

# Decision Problem as Predicate

A decision problem can be stated as a predicate $P(x)$ on number.

It relates to the problem-as-function viewpoint by the following characteristic function of $P(x)$:

$$c_P(n) = \begin{cases} 1, & \text{if } P(n) \text{ is valid,} \\ 0, & \text{otherwise.} \end{cases}$$

$$\begin{aligned} \text{Decision Problem} &\Leftrightarrow \text{Subset of } \mathbb{N} \\ &\Leftrightarrow \text{Predicate on } \mathbb{N} \end{aligned}$$

Several Problems

# Problem I

Is the function $tower(x)$ defined below computable?

$$tower(x) = \underbrace{2^{2^{\cdots^{2}}}}_{x}$$

# Problem I

Is the function $tower(x)$ defined below computable?

$$tower(x) = \underbrace{2^{2^{\cdots^{2}}}}_{x}$$

Theoretically it is computable.

# Problem II

Consider the function $f$ defined as follows:

$$f(n) = \begin{cases} 1, & \text{if } n > 1 \text{ and } 2n \text{ is the sum of 2 primes,} \\ 0, & \text{otherwise.} \end{cases}$$

The Goldbach Conjecture remains unsolved. Is $f$ computable?

# Problem II

Consider the function $f$ defined as follows:

$$f(n) = \begin{cases} 1, & \text{if } n > 1 \text{ and } 2n \text{ is the sum of 2 primes,} \\ 0, & \text{otherwise.} \end{cases}$$

The Goldbach Conjecture remains unsolved. Is $f$ computable?

It is clearly computable even if we do not know what it is.

# Problem III

Consider the function $g$ defined as follows:

$$g(n) = \begin{cases} 1, & \text{if there is a run of exactly } n \text{ consecutive } 7's \\ & \text{in the decimal expansion of } \pi, \\ 0, & \text{otherwise.} \end{cases}$$

It is known that $\pi$ can be calculated by $4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots\right)$.
Is $g$ computable?

# Problem III

Consider the function $g$ defined as follows:

$$g(n) = \begin{cases} 1, & \text{if there is a run of exactly } n \text{ consecutive } 7\text{'s} \\ & \text{in the decimal expansion of } \pi, \\ 0, & \text{otherwise.} \end{cases}$$

It is known that $\pi$ can be calculated by $4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots\right)$.
Is $g$ computable?

We do not know whether it is computable or not.

# Problem IV

Consider the function $h$ defined as follows:

$$h(n) = \begin{cases} 1, & \text{if } n \text{ is the machine code of a } C \text{ program that} \\ & \text{terminates in all inputs,} \\ 0, & \text{otherwise.} \end{cases}$$

# Problem IV

Consider the function $h$ defined as follows:

$$h(n) = \begin{cases} 1, & \text{if } n \text{ is the machine code of a } C \text{ program that} \\ & \text{terminates in all inputs,} \\ 0, & \text{otherwise.} \end{cases}$$

This is the Halting Problem, a well known undecidable problem. In other words there does not exist any C program calculating $h$.

The only general approach to check if a function is defined on all numbers is to calculate it on all inputs.

# Problem V

Consider the function $i$ defined as follows:

$$i(x,n,t) = \begin{cases} 1, & \text{if on input } x, \text{ the machine coded by } n \\ & \text{terminates in } t \text{ steps,} \\ 0, & \text{otherwise.} \end{cases}$$

There could be a number of ways to interpret "$t$ steps".

# Problem V

Consider the function $i$ defined as follows:

$$i(x,n,t) = \begin{cases} 1, & \text{if on input } x, \text{ the machine coded by } n \\ & \text{terminates in } t \text{ steps,} \\ 0, & \text{otherwise.} \end{cases}$$

There could be a number of ways to interpret "$t$ steps".

The function $i$ is intuitively computable.

# Next Lecture

The examples try to suggest that in order to study computability one might as well look for a theory of computable functions.

# Next Lecture

The examples try to suggest that in order to study computability one might as well look for a theory of computable functions.

We will begin with a machine model, register machine.

# Homework

- home reading: diagonal method.
- home reading: Presburger arithmetic.

# Computability Theory II

## Unlimited Register Machine

Guoqiang Li

Shanghai Jiao Tong University

Sep. 26, 2014

# Review Tips

# Computable Functions

- In a formal theory of computability, every problem instance can be represented by a number and every number represents a problem instance.
- A problem is a function $f : \mathbb{N} \to \mathbb{N}$ from numbers to numbers.
- A problem is computable if it can be calculated by a program.

# Decision Problem

A problem $f : \mathbb{N} \to \mathbb{N}$ is a decision problem if the range $ran(f)$ of $f$ is $\{0, 1\}$, where $1$ denotes a 'yes' answer and $0$ a 'no' answer.

A decision problem $g$ can be identified with the set $\{n \mid g(n) = 1\}$.

Conversely a subset $A$ of $\mathbb{N}$ can be seen as a decision problem via the characteristic function of $A$:

$$c_A(n) = \begin{cases} 1, & \text{if } x \in A, \\ 0, & \text{otherwise.} \end{cases}$$

# Decision Problem as Predicate

A decision problem can be stated as a predicate $P(x)$ on number.

It relates to the problem-as-function viewpoint by the following characteristic function of $P(x)$:

$$c_P(n) = \begin{cases} 1, & \text{if } P(n) \text{ is valid,} \\ 0, & \text{otherwise.} \end{cases}$$

$$\begin{aligned} \text{Decision Problem} &\Leftrightarrow \text{Subset of } \mathbb{N} \\ &\Leftrightarrow \text{Predicate on } \mathbb{N} \end{aligned}$$

# Register Machine

# Remark

Register Machines are more advanced than Turing Machines.

# Remark

Register Machine Models can be classified into three groups:

- CM (Counter Machine Model).
- RAM (Random Access Machine Model).
- RASP (Random Access Stored Program Machine Model).

# Synopsis

1. Unlimited Register Machine
2. Definability in URM

# Unlimited Register Machine

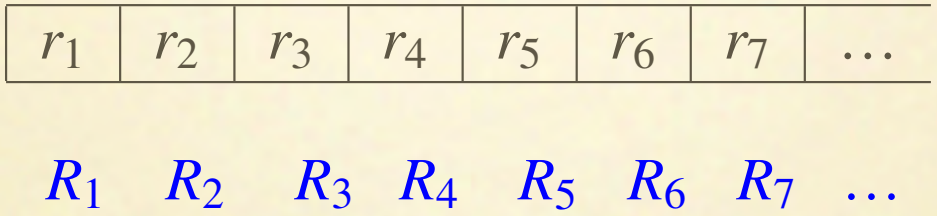
# Unlimited Register Machine Model

The Unlimited Register Machine Model belongs to the CM class.

Computability and Recursive Functions, by J. Shepherdson and H. Sturgis, in Journal of Symbolic Logic (**32**):1-63, 1965.

# Register

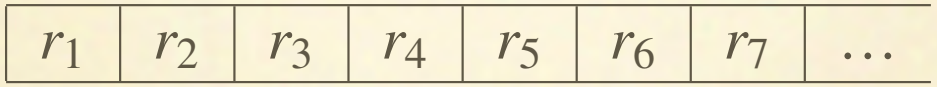
An Unlimited Register Machine (URM) has an infinite number of register labeled $R_1, R_2, R_3, \ldots$.




Every register can hold a natural number at any moment.

# Register

An Unlimited Register Machine (URM) has an infinite number of register labeled $R_1, R_2, R_3, \ldots$.




$R_1$ $R_2$ $R_3$ $R_4$ $R_5$ $R_6$ $R_7$ …

Every register can hold a natural number at any moment.

The registers can be equivalently written as for example

$$[r_1, r_2, r_3]_1^3 [r_4]_4^4 [r_5, r_6, r_7]_5^7 [0, 0, 0, \ldots]_8^\infty$$

or simply

$$[r_1, r_2, r_3]_1^3 [r_4]_4^4 [r_5, r_6, r_7]_5^7.$$

# Program

A URM also has a program, which is a finite list of instructions.

# Instruction

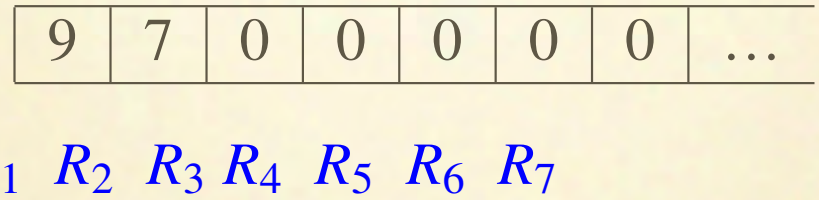
| Type | Instruction | Response of the URM |
|---|---|---|
| Zero | $Z(n)$ | Replace $r_n$ by $0$. |
| Successor | $S(n)$ | Add $1$ to $r_n$. |
| Transfer | $T(m,n)$ | Copy $r_m$ to $R_n$. |
| Jump | $J(m,n,q)$ | If $r_m = r_n$, go to the $q$-th instruction; otherwise go to the next instruction. |

# Program Rules

- $P = \{I_1, I_2, \cdots, I_s\} \rightarrow$ URM.
- URM starts by obeying instruction $I_1$.
- When URM finishes obeying $I_k$, it proceeds to the next instruction in the computation,
  - if $I_k$ is not a jump instruction, then the next instruction is $I_{k+1}$;
  - if $I_k = J(m,n,q)$ then next instruction is
    - $I_q$, if $r_m = r_n$; or
    - $I_{k+1}$, otherwise.
- Computation stops when the next instruction is $I_v$, where $v > s$.
  - if $k = s$, and $I_s$ is an arithmetic instruction;
  - if $I_k = J(m,n,q)$, $r_m = r_n$ and $q > s$;
  - if $I_k = J(m,n,q)$, $r_m \neq r_n$ and $k = s$.

# Computation

Registers:




Program:

$I_1 : J(1,2,6)$
$I_2 : S(2)$
$I_3 : S(3)$
$I_4 : J(1,2,6)$
$I_5 : J(1,1,2)$
$I_6 : T(3,1)$

# Configuration and Computation

Configuration: register contents + current instruction number.

# Configuration and Computation

Configuration: register contents + current instruction number.

Initial configuration, computation, final configuration.

# Some Notation

Suppose $P$ is the program of a URM and $a_1, a_2, a_3, \ldots$ are the numbers stored in the registers.

- $P(a_1, a_2, \ldots, a_m)$ is $P(a_1, a_2, \ldots, a_m, 0, 0, \ldots)$.
- $P(a_1, a_2, a_3, \ldots)$ is the initial configuration.
- $P(a_1, a_2, a_3, \ldots)\downarrow$ means that the computation converges.
- $P(a_1, a_2, a_3, \ldots)\uparrow$ means that the computation diverges.

# Definability in URM

# URM-Computable Function

Let $f(\widetilde{x})$ be an $n$-ary (partial) function.

What does it mean that a URM computes $f(\widetilde{x})$?

# URM-Computable Function

Suppose $P$ is the program of a URM and $a_1, \ldots, a_n, b \in \mathbb{N}$.

# URM-Computable Function

Suppose $P$ is the program of a URM and $a_1, \ldots, a_n, b \in \mathbb{N}$.

The computation $P(a_1, \ldots, a_n)$ converges to $b$ if $P(a_1, \ldots, a_n) \downarrow$ and $r_1 = b$ in the final configuration.

# URM-Computable Function

Suppose $P$ is the program of a URM and $a_1, \ldots, a_n, b \in \mathbb{N}$.

The computation $P(a_1, \ldots, a_n)$ converges to $b$ if $P(a_1, \ldots, a_n) \downarrow$ and $r_1 = b$ in the final configuration.

In this case we write $P(a_1, \ldots, a_n) \downarrow b$.

# URM-Computable Function

Suppose $P$ is the program of a URM and $a_1, \ldots, a_n, b \in \mathbb{N}$.

The computation $P(a_1, \ldots, a_n)$ converges to $b$ if $P(a_1, \ldots, a_n) \downarrow$ and $r_1 = b$ in the final configuration.

In this case we write $P(a_1, \ldots, a_n) \downarrow b$.

$P$ URM-computes $f$ if, for all $a_1, \ldots, a_n, b \in \mathbb{N}$, $P(a_1, \ldots, a_n) \downarrow b$ iff $f(a_1, \ldots, a_n) = b$.

# URM-Computable Function

Suppose $P$ is the program of a URM and $a_1, \ldots, a_n, b \in \mathbb{N}$.

The computation $P(a_1, \ldots, a_n)$ converges to $b$ if $P(a_1, \ldots, a_n) \downarrow$ and $r_1 = b$ in the final configuration.

In this case we write $P(a_1, \ldots, a_n) \downarrow b$.

$P$ URM-computes $f$ if, for all $a_1, \ldots, a_n, b \in \mathbb{N}$, $P(a_1, \ldots, a_n) \downarrow b$ iff $f(a_1, \ldots, a_n) = b$.

The function $f$ is URM-definable if there is a program that URM-computes $f$.

We shall abbreviate “URM-computable” to “computable”.

Let

$$\mathscr{C}$$

be the set of computable functions and

$$\mathscr{C}_n$$

be the set of $n$-ary computable functions.

# Example of URM I

Construct a URM that computes $x + y$.

# Example of URM I

Construct a URM that computes $x + y$.

$I_1 : J(3, 2, 5)$
$I_2 : S(1)$
$I_3 : S(3)$
$I_4 : J(1, 1, 1)$

# Example of URM II

Construct a URM that computes $x \dot{-} 1 = \begin{cases} x - 1, & \text{if } x > 0, \\ 0, & \text{if } x = 0. \end{cases}$

# Example of URM II

Construct a URM that computes $x \dot{-} 1 = \begin{cases} x - 1, & \text{if } x > 0, \\ 0, & \text{if } x = 0. \end{cases}$

$I_1 : J(1,4,8)$
$I_2 : S(3)$
$I_3 : J(1,3,7)$
$I_4 : S(2)$
$I_5 : S(3)$
$I_6 : J(1,1,3)$
$I_7 : T(2,1)$

# Example of URM III

Construct a URM that computes $x \div 2 = \begin{cases} x/2, & \text{if } x \text{ is even,} \\ \text{undefined}, & \text{if } x \text{ is odd.} \end{cases}$

# Example of URM III

Construct a URM that computes $x \div 2 = \begin{cases} x/2, & \text{if } x \text{ is even,} \\ \text{undefined}, & \text{if } x \text{ is odd.} \end{cases}$

$I_1 : J(1,2,6)$
$I_2 : S(3)$
$I_3 : S(2)$
$I_4 : S(2)$
$I_5 : J(1,1,1)$
$I_6 : T(3,1)$

# Example of URM IV

Construct a URM that computes $f(x) = \lfloor 3x/4 \rfloor$

# Example of URM IV

Construct a URM that computes $f(x) = \lfloor 3x/4 \rfloor$

$I_1$ Z(2)
$I_2$ Z(3)
$I_3$ Z(4)
$I_4$ J(1,2,10)
$I_5$ S(2)
$I_6$ S(3)
$I_7$ S(3)
$I_8$ S(3)
$I_9$ J(1,1,4)
$I_{10}$ Z(2)
$I_{11}$ J(2,3,21)
$I_{12}$ S(2)
$I_{13}$ J(2,3,21)
$I_{14}$ S(2)
$I_{15}$ J(2,3,21)
$I_{16}$ S(2)
$I_{17}$ J(2,3,21)
$I_{18}$ S(2)
$I_{19}$ S(4)
$I_{20}$ J(1,1,11)
$I_{21}$ T(4,1)

# Function Defined by Program

$$f_P^n(a_1, \ldots, a_n) = \begin{cases} b, & \text{if } P(a_1, \ldots, a_n) \downarrow b, \\ \text{undefined}, & \text{if } P(a_1, \ldots, a_n) \uparrow . \end{cases}$$

# Program in Standard Form

A program $P = I_1, \ldots, I_s$ is in standard form if, for every jump instruction $J(m, n, q)$ we have $q \leq s + 1$.

# Program in Standard Form

A program $P = I_1, \ldots, I_s$ is in standard form if, for every jump instruction $J(m, n, q)$ we have $q \leq s + 1$.

For every program there is a program in standard form that computes the same function.

# Program in Standard Form

A program $P = I_1, \ldots, I_s$ is in standard form if, for every jump instruction $J(m, n, q)$ we have $q \leq s + 1$.

For every program there is a program in standard form that computes the same function.

We will focus exclusively on programs in standard form.

# Program Composition

Given Programs $P$ and $Q$, how do we construct the sequential composition $P; Q$?

# Program Composition

Given Programs $P$ and $Q$, how do we construct the sequential composition $P; Q$?

The jump instructions of $P$ and $Q$ must be modified.

# Some Notations

Suppose the program $P$ computes $f$.

Let $\rho(P)$ be the least number $i$ such that the register $R_i$ is not used by the program $P$.

# Some Notations

The notation $P[l_1, \ldots, l_n \to l]$ stands for the following program

$$
\begin{array}{rcl}
I_1 & : & T(l_1, 1) \\
 & \vdots & \\
I_n & : & T(l_n, n) \\
I_{n+1} & : & Z(n+1) \\
 & \vdots & \\
I_{\rho(P)} & : & Z(\rho(P)) \\
\_ & : & P \\
\_ & : & T(1, l)
\end{array}
$$

# Computability Theory III

## Primitive Recursive Function

Guoqiang Li

Shanghai Jiao Tong University

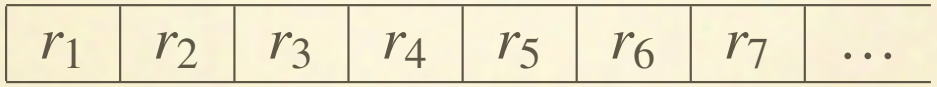
Oct. 10, 2014

*Assignment 1 is announced! (deadline Oct. 24)*

# Review Tips

# Register

An Unlimited Register Machine (URM) has an infinite number of register labeled $R_1, R_2, R_3, \ldots$.




$R_1$ $R_2$ $R_3$ $R_4$ $R_5$ $R_6$ $R_7$ …

Every register can hold a natural number at any moment.

The registers can be equivalently written as for example

$$[r_1, r_2, r_3]_1^3 [r_4]_4^4 [r_5, r_6, r_7]_5^7 [0, 0, 0, \ldots]_8^\infty$$

or simply

$$[r_1, r_2, r_3]_1^3 [r_4]_4^4 [r_5, r_6, r_7]_5^7.$$

# Instruction

| Type | Instruction | Response of the URM |
|---|---|---|
| Zero | $Z(n)$ | Replace $r_n$ by $0$. |
| Successor | $S(n)$ | Add $1$ to $r_n$. |
| Transfer | $T(m,n)$ | Copy $r_m$ to $R_n$. |
| Jump | $J(m,n,q)$ | If $r_m = r_n$, go to the $q$-th instruction; otherwise go to the next instruction. |

# Recursive Function

# Recursion Theory

Recursion Theory offers a mathematical model for the study of effective calculability.

1. All effective objects can be encoded by natural numbers.
2. All effective procedures can be modeled by functions from numbers to numbers.

# Synopsis

1. Primitive Recursive Function

# Primitive Recursive Function

# Basic Definitions

# Initial Function

1. The zero function
   - $\mathbf{0}$
   - $\mathbf{0}(\widetilde{x}) = 0$
2. The successor function
   - $s(x) = x + 1$
3. The projection function
   - $U_i^n(x_1, \ldots, x_n) = x_i$

# Composition

Suppose $f(y_1, \ldots, y_k)$ is a $k$-ary function and $g_1(\widetilde{x}), \ldots, g_k(\widetilde{x})$ are $n$-ary functions, where $\widetilde{x}$ abbreviates $x_1, \ldots, x_n$.

The composition function $h(\widetilde{x})$ is defined by

$$h(\widetilde{x}) \quad = \quad f(g_1(\widetilde{x}), \ldots, g_k(\widetilde{x})),$$

# Recursion

Suppose that $f(\widetilde{x})$ is an $n$-ary function and $g(\widetilde{x}, y, z)$ is an $(n+2)$-ary function.

The recursion function $h(\widetilde{x}, y)$ is defined by

$$h(\widetilde{x}, 0) = f(\widetilde{x}), \tag{1}$$

$$h(\widetilde{x}, y+1) = g(\widetilde{x}, y, h(\widetilde{x}, y)). \tag{2}$$

Clearly there is a unique function that satisfies (1) and (2).

# Primitive Recursive Recursion

The set of primitive recursive function is the least set generated from the initial functions, composition and recursion.

# Dummy Parameter

**Proposition**

Suppose that $f(y_1, \ldots, y_k)$ is a primitive recursive and that $x_{i_1}, \ldots, x_{i_k}$ is a sequence of $k$ variables from $x_1, \ldots, x_n$ (possibly with repetition). Then the function $h$ given by

$$h(x_1, \ldots, x_n) \quad = \quad f(x_{i_1}, \ldots, x_{i_k})$$

is primitive recursive.

**Proof**

$h(\widetilde{x}) = f(\mathsf{U}^n_{i_1}(\widetilde{x}), \ldots, \mathsf{U}^n_{i_k}(\widetilde{x}))$.

# Basic Arithmetic Function

# Basic Arithmetic Function

- $x + y$
-
$$\begin{aligned} x + 0 &= x, \\ x + (y + 1) &= s(x + y). \end{aligned}$$
- $xy$
-
$$\begin{aligned} x0 &= 0, \\ x(y + 1) &= xy + x. \end{aligned}$$
- $x^y$
-
$$\begin{aligned} x^0 &= 1, \\ x^{y+1} &= x^y x. \end{aligned}$$

# Quiz

$x + y + z$

# Basic Arithmetic Function

- $x \dot{-} 1$
-

$$\begin{aligned} 0 \dot{-} 1 &= 0, \\ (x+1) \dot{-} 1 &= x. \end{aligned}$$

- $x \dot{-} y \stackrel{\text{def}}{=} \begin{cases} x - y, & \text{if } x \geq y, \\ 0, & \text{otherwise.} \end{cases}$
-

$$\begin{aligned} x \dot{-} 0 &= x, \\ x \dot{-} (y+1) &= (x \dot{-} y) \dot{-} 1. \end{aligned}$$

# Basic Arithmetic Function

- $\mathsf{sg}(x) \stackrel{\text{def}}{=} \begin{cases} 0, & \text{if } x = 0, \\ 1, & \text{if } x \neq 0. \end{cases}$
-

$$\begin{aligned} \mathsf{sg}(0) &= 0, \\ \mathsf{sg}(x+1) &= 1. \end{aligned}$$

- $\overline{\mathsf{sg}}(x) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } x = 0, \\ 0, & \text{if } x \neq 0. \end{cases}$
-

$$\overline{\mathsf{sg}}(x) = 1 \dot{-} \mathsf{sg}(x).$$

# Basic Arithmetic Function

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

$$\begin{aligned} 0! &= 1, \\ (x+1)! &= x!(x+1). \end{aligned}$$

- $\min(x, y)$
- $\min(x, y) = x \dot{-} (x \dot{-} y).$
- $\max(x, y)$
- $\max(x, y) = x + (y \dot{-} x).$

# Basic Arithmetic Function

$\mathsf{rm}(x,y) \stackrel{\text{def}}{=}$ the remainder when $y$ is devided by $x$

$$\mathsf{rm}(x,y+1) \stackrel{\text{def}}{=} \begin{cases} \mathsf{rm}(x,y)+1 & \text{if } \mathsf{rm}(x,y)+1 < x, \\ 0, & \text{otherwise.} \end{cases}$$

The recursive definition is given by

$$\begin{aligned} \mathsf{rm}(x,0) &= 0, \\ \mathsf{rm}(x,y+1) &= (\mathsf{rm}(x,y)+1)\mathrm{sg}(x \dot{-} (\mathsf{rm}(x,y)+1)). \end{aligned}$$

# Basic Arithmetic Function

$\mathsf{qt}(x,y) \stackrel{\text{def}}{=}$ the quotient when $y$ is devided by $x$

$$\mathsf{qt}(x,y+1) \stackrel{\text{def}}{=} \begin{cases} \mathsf{qt}(x,y)+1, & \text{if } \mathsf{rm}(x,y)+1=x, \\ \mathsf{qt}(x,y), & \text{if } \mathsf{rm}(x,y)+1 \neq x. \end{cases}$$

The recursive definition is given by

$$\begin{aligned} \mathsf{qt}(x,0) &= 0, \\ \mathsf{qt}(x,y+1), &= \mathsf{qt}(x,y) + \overline{\mathrm{sg}}(x-(\mathsf{rm}(x,y)+1)). \end{aligned}$$

# Basic Arithmetic Function

$$\mathsf{div}(x,y) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } x \text{ divides } y, \\ 0, & \text{otherwise.} \end{cases}$$

$$\mathsf{div}(x,y) = \overline{\mathrm{sg}}(\mathsf{rm}(x,y)).$$

# Bounded Minimalisation Operator

# Bounded Sum and Bounded Product

Bounded sum:

$$\begin{aligned}\sum_{y<0} f(\widetilde{x},y) &= 0,\\ \sum_{y<z+1} f(\widetilde{x},y) &= \sum_{y<z} f(\widetilde{x},y)+f(\widetilde{x},z).\end{aligned}$$

Bounded product:

$$\begin{aligned}\prod_{y<0} f(\widetilde{x},y) &= 1,\\ \prod_{y<z+1} f(\widetilde{x},y) &= (\prod_{y<z} f(\widetilde{x},y))\cdot f(\widetilde{x},z).\end{aligned}$$

# Bounded Sum and Bounded Product

By composition the following functions are also primitive recursive if $k(\widetilde{x}, \widetilde{w})$ is primitive recursive:

$$\sum_{z<k(\widetilde{x},\widetilde{w})} f(\widetilde{x}, z)$$

and

$$\prod_{z<k(\widetilde{x},\widetilde{w})} f(\widetilde{x}, z).$$

# Bounded Minimization Operator

Bounded search:

$$\mu z{<}y(f(\widetilde{x},z)=0) \stackrel{\text{def}}{=} \begin{cases} \text{the least } z<y, & \text{such that } f(\widetilde{x},z)=0; \\ y, & \text{if there is no such } z. \end{cases}$$

**Proposition**

If $f(\widetilde{x},z)$ is primitive recursive, then so is $\mu z{<}y(f(\widetilde{x},z)=0)$

**Proof**

$\mu z{<}y(f(\widetilde{x},z)=0) = \sum_{v<y}(\prod_{u<v+1} \mathsf{sg}(f(\widetilde{x},u)))$

# Bounded Minimization Operator

If $f(\widetilde{x}, z)$ and $k(\widetilde{x}, \widetilde{w})$ are primitive recursive functions, then so is the function

$$\mu z{<}k(\widetilde{x}, \widetilde{w})(f(\widetilde{x}, z) = 0).$$

# Primitive Recursive Predicate

# Primitive Recursive Predicate

Suppose $M(x_1, \ldots, x_n)$ is an $n$-ary predicate of natural numbers. The characteristic function $c_M(\widetilde{x})$, where $\widetilde{x} = x_1, \ldots, x_n$, is

$$c_M(a_1, \ldots, a_n) = \begin{cases} 1, & \text{if } M(a_1, \ldots, a_n) \text{ holds,} \\ 0, & \text{if otherwise.} \end{cases}$$

The predicate $M(\widetilde{x})$ is primitive recursive if $c_M$ is primitive recursive.

# Closure Property

## Proposition

The following statements are valid:

- If $R(\widetilde{x})$ is a primitive recursive predicate, then so is $\neg R(\widetilde{x})$.
- If $R(\widetilde{x})$, $S(\widetilde{x})$ are primitive recursive predicates, then the following predicates are primitive recursive:
  - $R(\widetilde{x}) \wedge S(\widetilde{x})$;
  - $R(\widetilde{x}) \vee S(\widetilde{x})$.
- If $R(\widetilde{x}, y)$ is a primitive recursive predicate, then the following predicates are primitive recursive:
  - $\forall z < y.R(\widetilde{x}, z)$;
  - $\exists z < y.R(\widetilde{x}, z)$.

## Proof

For example $c_{\forall z<y.R(\widetilde{x},z)}(\widetilde{x}, y) = \prod_{z<y} c_R(\widetilde{x}, z)$.

# Definition by Case

## Proposition

Suppose that $f_1(\widetilde{x}), \ldots, f_k(\widetilde{x})$ are primitive recursive functions, and $M_1(\widetilde{x}), \ldots, M_k(\widetilde{x})$ are primitive recursive predicates, such that for every $\widetilde{x}$ exactly one of $M_1(\widetilde{x}), \ldots, M_k(\widetilde{x})$ holds. Then the function $g(\widetilde{x})$ given by

$$g(\widetilde{x}) = \begin{cases} f_1(\widetilde{x}), & \text{if } M_1(\widetilde{x}) \text{ holds,} \\ f_2(\widetilde{x}), & \text{if } M_2(\widetilde{x}) \text{ holds,} \\ \vdots & \\ f_k(\widetilde{x}), & \text{if } M_k(\widetilde{x}) \text{ holds.} \end{cases}$$

is primitive recursive.

## Proof

$g(\widetilde{x}) = c_{M_1}(\widetilde{x}) f_1(\widetilde{x}) + \ldots + c_{M_k}(\widetilde{x}) f_k(\widetilde{x})$

# More Arithmetic Functions

# More Arithmetic Functions

The following functions are primitive recursive.

1. $D(x) =$ the number of divisors of $x$;
2. $Pr(x) = \begin{cases} 1, & \text{if } x \text{ is prime,} \\ 0, & \text{if } x \text{ is not prime.} \end{cases}$
3. $\mathsf{p}_x =$ the $x$-th prime number;
4. $(x)_y = \begin{cases} k, & k \text{ is the exponent of } p_y \text{ in the prime} \\ & \text{factorisation of } x, \text{ for } x, y > 0, \\ 0, & \text{if } x = 0 \text{ or } y = 0. \end{cases}$

# More Arithmetic Functions

Proof

1. $D(x) = \sum_{y<x+1} \mathrm{div}(y, x)$.
2. $Pr(x) = \overline{\mathsf{sg}}(|D(x) - 2|)$.
3. $\mathsf{p}_x$ can be recursively defined as follows:

$$\begin{aligned} \mathsf{p}_0 &= 0, \\ \mathsf{p}_{x+1} &= \mu z < (1 + \mathsf{p}_x!)\left(1 \dot{-} (z \dot{-} \mathsf{p}_x) Pr(z) = 0\right). \end{aligned}$$

4. $(x)_y = \mu z{<}x(\mathrm{div}(\mathsf{p}_y^{z+1}, x) = 0)$.

# Encoding a Finite Sequence

Suppose $s = (a_1, a_2, \ldots, a_n)$ is a finite sequence of numbers. It can be coded by the following number

$$b = \mathsf{p}_1^{a_1+1} \mathsf{p}_2^{a_2+1} \ldots \mathsf{p}_n^{a_n+1}.$$

Then the length of $s$ can be recovered from

$$\mu z{<}b((b)_{z+1} = 0),$$

and the $i$-th component can be recovered from

$$(b)_i \dot{-} 1.$$

# Not all Computable Functions are Primitive Recursive

Using the fact that all primitive recursive functions are total, a diagonalisation argument shows that non-primitive recursive computable functions must exist.

The same diagonalisation argument applies to all finite axiomatizations of computable total function.

Onward to the partial functions!

# Computability Theory IV

## Recursive Function

Guoqiang Li

Shanghai Jiao Tong University

Oct. 17, 2014

# Review Tips

# Initial Function

1. The zero function
   - $\mathbf{0}$
   - $\mathbf{0}(\widetilde{x}) = 0$
2. The successor function
   - $s(x) = x + 1$
3. The projection function
   - $U_i^n(x_1, \ldots, x_n) = x_i$

# Composition

Suppose $f(y_1, \ldots, y_k)$ is a $k$-ary function and $g_1(\widetilde{x}), \ldots, g_k(\widetilde{x})$ are $n$-ary functions, where $\widetilde{x}$ abbreviates $x_1, \ldots, x_n$.

The composition function $h(\widetilde{x})$ is defined by

$$h(\widetilde{x}) \quad = \quad f(g_1(\widetilde{x}), \ldots, g_k(\widetilde{x})),$$

# Recursion

Suppose that $f(\widetilde{x})$ is an $n$-ary function and $g(\widetilde{x}, y, z)$ is an $(n+2)$-ary function.

The recursion function $h(\widetilde{x}, y)$ is defined by

$$h(\widetilde{x}, 0) = f(\widetilde{x}), \tag{1}$$

$$h(\widetilde{x}, y+1) = g(\widetilde{x}, y, h(\widetilde{x}, y)). \tag{2}$$

Clearly there is a unique function that satisfies (1) and (2).

# Quiz

$LCM(x, y)$

# Quiz

$$LCM(x,y)$$

Sol. $LCM(x,y) = \mu z < xy + 1(div(x,z)div(y,z) = 1)$.

# Quiz

$$LCM(x,y)$$

Sol. $LCM(x,y) = \mu z < xy + 1(div(x,z)div(y,z) = 1)$.

$$HCF(x,y))$$

# Quiz

$$LCM(x,y)$$

Sol. $LCM(x,y) = \mu z < xy + 1(div(x,z)div(y,z) = 1)$.

$$HCF(x,y))$$

Sol. $HCF(x,y) = \frac{xy}{LCM(x,y)}$.

# Synopsis

1. Recursive Function
2. Ackermann Function
3. Definability in URM

# Recursive Function

# An Example

$$g(x) = \begin{cases} \sqrt{x} & \text{if } x \text{ is a pefect square.} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

# Minimization Operator, or Search Operator

Minimization function, or $\mu$-function, or search function:

$$\mu y(f(\widetilde{x}, y) = 0) \quad \simeq \quad \begin{cases} \text{the least } y \text{ such that} \\ \quad f(\widetilde{x}, z) \text{ is defined for all } z \leq y, \text{ and} \\ \quad f(\widetilde{x}, y) = 0, \\ \text{undefined if otherwise.} \end{cases}$$

Here $\simeq$ is the computational equality.

# Minimization Operator, or Search Operator

Minimization function, or $\mu$-function, or search function:

$$\mu y(f(\widetilde{x}, y) = 0) \quad \simeq \quad \begin{cases} \text{the least } y \text{ such that} \\ \quad f(\widetilde{x}, z) \text{ is defined for all } z \leq y, \text{ and} \\ \quad f(\widetilde{x}, y) = 0, \\ \text{undefined if otherwise.} \end{cases}$$

Here $\simeq$ is the computational equality.

- The recursion operation is a well-founded going-down procedure.

# Minimization Operator, or Search Operator

Minimization function, or $\mu$-function, or search function:

$$\mu y(f(\widetilde{x}, y) = 0) \quad \simeq \quad \begin{cases} \text{the least } y \text{ such that} \\ \quad f(\widetilde{x}, z) \text{ is defined for all } z \leq y, \text{ and} \\ \quad f(\widetilde{x}, y) = 0, \\ \text{undefined if otherwise.} \end{cases}$$

Here $\simeq$ is the computational equality.

- The recursion operation is a well-founded going-down procedure.
- The search operation is a possibly divergent going-up procedure.

# An Example

$$g(x) = \begin{cases} \sqrt{x} & \text{if } x \text{ is a pefect square.} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

# An Example

$$g(x) = \begin{cases} \sqrt{x} & \text{if } x \text{ is a pefect square.} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

$$f(x,y) = |x - y^2|$$

# Recursive Function

The set of recursive functions is the least set generated from the initial functions, composition, recursion and minimization.

# Decidable Predicate

A predicate $R(\widetilde{x})$ is decidable if its characteristic function

$$c_R(\widetilde{x}) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } R(\widetilde{x}) \text{ is true,} \\ 0, & \text{otherwise.} \end{cases}$$

is a recursive function.

# Decidable Predicate

A predicate $R(\widetilde{x})$ is decidable if its characteristic function

$$c_R(\widetilde{x}) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } R(\widetilde{x}) \text{ is true,} \\ 0, & \text{otherwise.} \end{cases}$$

is a recursive function. The predicate $R(\widetilde{x})$ is partially decidable if its partial characteristic function

$$\chi_R(\widetilde{x}) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } R(\widetilde{x}) \text{ is true,} \\ \uparrow, & \text{otherwise.} \end{cases}$$

is a recursive function.

# Closure Property

The following statements are valid:

- If $R(\widetilde{x})$ is decidable, then so is $\neg R(\widetilde{x})$.
- If $R(\widetilde{x}), S(\widetilde{x})$ are (partially) decidable, then the following predicates are (partially) decidable:
  - $R(\widetilde{x}) \wedge S(\widetilde{x})$;
  - $R(\widetilde{x}) \vee S(\widetilde{x})$.
- If $R(\widetilde{x}, y)$ is (partially) decidable, then the following predicates are (partially) decidable:
  - $\forall z < y . R(\widetilde{x}, y)$;
  - $\exists z < y . R(\widetilde{x}, y)$.

# Definition by Cases

Suppose $f_1(\widetilde{x}), \ldots, f_k(\widetilde{x})$ are recursive and $M_1(\widetilde{x}), \ldots, M_k(\widetilde{x})$ are partially decidable. For every $\widetilde{x}$ at most one of $M_1(\widetilde{x}), \ldots, M_k(\widetilde{x})$ holds. Then the function $g(\widetilde{x})$ given by

$$g(\widetilde{x}) \simeq \begin{cases} f_1(\widetilde{x}), & \text{if } M_1(\widetilde{x}) \text{ holds,} \\ f_2(\widetilde{x}), & \text{if } M_2(\widetilde{x}) \text{ holds,} \\ \vdots & \\ f_k(\widetilde{x}), & \text{if } M_k(\widetilde{x}) \text{ holds.} \end{cases}$$

is recursive.

# Minimization via Decidable Predicate

Suppose $R(x, y)$ is a partially decidable predicate. The function

$$
\begin{aligned}
g(x) &= \mu y R(\widetilde{x}, y) \\
&= \begin{cases} \text{the least } y \text{ such that } R(\widetilde{x}, y) \text{ holds}, & \text{if there is such a } y \\ \text{undefined}, & \text{otherwise.} \end{cases}
\end{aligned}
$$

is recursive.

# Minimization via Decidable Predicate

Suppose $R(x, y)$ is a partially decidable predicate. The function

$$
\begin{aligned}
g(x) &= \mu y R(\widetilde{x}, y) \\
&= \begin{cases} \text{the least } y \text{ such that } R(\widetilde{x}, y) \text{ holds}, & \text{if there is such a } y \\ \text{undefined}, & \text{otherwise.} \end{cases}
\end{aligned}
$$

is recursive.

Proof

$g(\widetilde{x}) = \mu y(\overline{\mathsf{sg}}(\chi_R(\widetilde{x}, y)) = 0)$.

# Comment

The $\mu$-operator allows one to define partial functions.

# Comment

The $\mu$-operator allows one to define partial functions.

The diagonalisation argument does not apply to the set $\mathfrak{F}$ of recursive functions.

# Comment

The $\mu$-operator allows one to define partial functions.

The diagonalisation argument does not apply to the set $\mathfrak{F}$ of recursive functions.

Using the $\mu$-operator, one may define total functions that are not primitive recursive.

# Minimization Operator is a Search Operator

It is clear from the above proof why the minimization operator is sometimes called a search operator.

# Definable Function

A function is definable if there is a recursive function calculating it.

Ackermann function

# Ackermann Function

The Ackermann function [1928] is defined as follows:

$$\begin{aligned} \psi(0, y) &\simeq y + 1, \\ \psi(x + 1, 0) &\simeq \psi(x, 1), \\ \psi(x + 1, y + 1) &\simeq \psi(x, \psi(x + 1, y)). \end{aligned}$$

The equations clearly define a total function.

# Ackermann is not Primitive Recursive

Lemma 1.

$\psi(1, m) = m + 2$ and $\psi(2, m) = 2m + 3$

# Ackermann is not Primitive Recursive

Lemma 1.

$\psi(1,m) = m+2$ and $\psi(2,m) = 2m+3$

Lemma 2.

$\psi(n,m) \geq m+1$

# Ackermann is not Primitive Recursive

# Ackermann is not Primitive Recursive

**Lemma 3.**

The Ackermann function is monotone:

$$\begin{aligned} \psi(n,m) &< \psi(n,m+1), \\ \psi(n,m) &< \psi(n+1,m). \end{aligned}$$

# Ackermann is not Primitive Recursive

**Lemma 3.**
The Ackermann function is monotone:

$$\begin{aligned} \psi(n,m) &< \psi(n,m+1), \\ \psi(n,m) &< \psi(n+1,m). \end{aligned}$$

**Lemma 4.**
The Ackermann function grows faster on the first parameter:

$$\psi(n,m+1) \leq \psi(n+1,m)$$

# Ackermann is not Primitive Recursive

Lemma 5.
$\psi(n,m) + C$ is dominated by $\psi(J,m)$ for some large enough $J$:

$$\begin{aligned} \psi(n,m) + \psi(n',m) &< \psi(\max(n,n') + 4, m), \\ \psi(n,m) + m &< \psi(n+4,m). \end{aligned}$$

# Ackermann is not Primitive Recursive

Lemma 6.

Let $f(\widetilde{x})$ be a $k$-ary primitive recursive function. Then there exists some $J$ such that for all $n_1, \ldots, n_k$ we have that

$$f(n_1, \ldots, n_k) < \psi(J, \sum_{i=1}^{k} n_k).$$

# Ackermann is not Primitive Recursive

Lemma 6.

Let $f(\widetilde{x})$ be a $k$-ary primitive recursive function. Then there exists some $J$ such that for all $n_1, \ldots, n_k$ we have that

$$f(n_1, \ldots, n_k) < \psi(J, \sum_{i=1}^{k} n_k).$$

*Proof.* The proof is by structural induction.

# Ackermann is not Primitive Recursive

Lemma 6.

Let $f(\widetilde{x})$ be a $k$-ary primitive recursive function. Then there exists some $J$ such that for all $n_1, \ldots, n_k$ we have that

$$f(n_1, \ldots, n_k) < \psi(J, \sum_{i=1}^{k} n_k).$$

*Proof.* The proof is by structural induction.

(i) $f$ is one of the initial functions. In this case take $J$ to be 1.

# Ackermann is not Primitive Recursive

(ii) $f$ is the composition function $h(g_1(\widetilde{x}), \ldots, g_m(\widetilde{x}))$. Then

$$\begin{aligned}
f(\widetilde{n}) &= h(g_1(\widetilde{n}), \ldots, g_m(\widetilde{n})) \\
&< \psi(J_0, \sum_{i=1}^{m} g_i(\widetilde{n})) < \psi(J_0, \sum_{i=1}^{m} \psi(J_i, \sum_{j=1}^{k} n_j)) \\
&< \psi(J_0, \psi(J^*, \sum_{j=1}^{k} n_j)) < \psi(J^*, \psi(J^*+1, \sum_{j=1}^{k} n_j)) \\
&= \psi(J^*+1, \sum_{j=1}^{k} n_j + 1) \leq \psi(J^*+2, \sum_{j=1}^{k} n_j).
\end{aligned}$$

Now set $J = J^* + 2$.

# Ackermann is not Primitive Recursive

(iii) Suppose $f$ is defined by the recursion:

$$\begin{aligned} f(\widetilde{x},0) &\simeq h(\widetilde{x}), \\ f(\widetilde{x},y+1) &\simeq g(\widetilde{x},y,f(\widetilde{x},y)). \end{aligned}$$

# Ackermann is not Primitive Recursive

(iii) Suppose $f$ is defined by the recursion:

$$\begin{aligned} f(\widetilde{x},0) &\simeq h(\widetilde{x}), \\ f(\widetilde{x},y+1) &\simeq g(\widetilde{x},y,f(\widetilde{x},y)). \end{aligned}$$

Then $h(\widetilde{n}) < \psi(J_h, \sum \widetilde{n})$ and $g(\widetilde{n},m,p) < \psi(J_g, \sum \widetilde{n} + m + p)$.

# Ackermann is not Primitive Recursive

(iii) Suppose $f$ is defined by the recursion:

$$\begin{aligned} f(\widetilde{x},0) &\simeq h(\widetilde{x}), \\ f(\widetilde{x},y+1) &\simeq g(\widetilde{x},y,f(\widetilde{x},y)). \end{aligned}$$

Then $h(\widetilde{n}) < \psi(J_h, \sum \widetilde{n})$ and $g(\widetilde{n},m,p) < \psi(J_g, \sum \widetilde{n} + m + p)$.

It is easy to prove

$$f(n_1,\ldots,n_k,m) < \psi(J, \sum_{i=1}^{k} n_k + m)$$

by induction on $m$.

# Ackermann is not Primitive Recursive

Now suppose $\psi(x, y)$ was primitive recursive.
By composition $\psi(x, x)$ would be primitive recursive.

# Ackermann is not Primitive Recursive

Now suppose $\psi(x, y)$ was primitive recursive.
By composition $\psi(x, x)$ would be primitive recursive.

According to the Lemma 6

$$\psi(n, n) < \psi(J, n)$$

for some $J$ and all $n$,

# Ackermann is not Primitive Recursive

Now suppose $\psi(x, y)$ was primitive recursive.
By composition $\psi(x, x)$ would be primitive recursive.

According to the Lemma 6

$$\psi(n, n) < \psi(J, n)$$

for some $J$ and all $n$, which would lead to the contradiction

$$\psi(J, J) < \psi(J, J).$$

# Ackermann is not Primitive Recursive

**Theorem**

The Ackermann function grows faster than every primitive recursive function.

# Ackermann Function is Recursive

**Theorem**

The Ackermann function is recursive.

# Ackermann Function is Recursive

A finite set $S$ of triples is said to be suitable if the followings hold:
(i) if $(0, y, z) \in S$ then $z = y + 1$;
(ii) if $(x + 1, 0, z) \in S$ then $(x, 1, z) \in S$;
(iii) if $(x + 1, y + 1, z) \in S$ then $\exists u.((x + 1, y, u) \in S \wedge (x, u, z) \in S)$.

# Ackermann Function is Recursive

A finite set $S$ of triples is said to be suitable if the followings hold:
(i) if $(0, y, z) \in S$ then $z = y + 1$;
(ii) if $(x + 1, 0, z) \in S$ then $(x, 1, z) \in S$;
(iii) if $(x + 1, y + 1, z) \in S$ then $\exists u.((x + 1, y, u) \in S \wedge (x, u, z) \in S)$.

A triple $(x, y, z)$ can be coded up by $2^x 3^y 5^z$.
A set $\{u_1, \ldots, u_k\}$ can be coded up by $p_{u_1} \cdots p_{u_k}$.

# Ackermann Function is Recursive

A finite set $S$ of triples is said to be suitable if the followings hold:
(i) if $(0, y, z) \in S$ then $z = y + 1$;
(ii) if $(x + 1, 0, z) \in S$ then $(x, 1, z) \in S$;
(iii) if $(x + 1, y + 1, z) \in S$ then $\exists u.((x + 1, y, u) \in S \wedge (x, u, z) \in S)$.

A triple $(x, y, z)$ can be coded up by $2^x 3^y 5^z$.
A set $\{u_1, \ldots, u_k\}$ can be coded up by $p_{u_1} \cdots p_{u_k}$.

Let $R(x, y, v)$ be "$v$ is a legal code and $\exists z < v.(x, y, z) \in S_v$".

# Ackermann Function is Recursive

A finite set $S$ of triples is said to be suitable if the followings hold:
(i) if $(0, y, z) \in S$ then $z = y + 1$;
(ii) if $(x + 1, 0, z) \in S$ then $(x, 1, z) \in S$;
(iii) if $(x + 1, y + 1, z) \in S$ then $\exists u.((x + 1, y, u) \in S \wedge (x, u, z) \in S)$.

A triple $(x, y, z)$ can be coded up by $2^x 3^y 5^z$.
A set $\{u_1, \ldots, u_k\}$ can be coded up by $p_{u_1} \cdots p_{u_k}$.

Let $R(x, y, v)$ be "$v$ is a legal code and $\exists z < v.(x, y, z) \in S_v$".

The Ackermann function $\psi(x, y) \simeq \mu z((x, y, z) \in S_{\mu v R(x,y,v)})$.

# Definability in URM

# Definability of Initial Function

**Fact**. The initial functions are URM-definable.

# Definability of Composition

**Fact**. If $f(y_1, \ldots, y_k)$ and $g_1(\widetilde{x}), \ldots, g_k(\widetilde{x})$ are URM-definable, then the composition function $h(\widetilde{x})$ given by

$$h(\widetilde{x}) \quad \simeq \quad f(g_1(\widetilde{x}), \ldots, g_k(\widetilde{x}))$$

is URM-definable.

# Some Notations

Suppose the program $P$ computes $f$.

Let $\rho(P)$ be the least number $i$ such that the register $R_i$ is not used by the program $P$.

# Some Notations

The notation $P[l_1, \ldots, l_n \to l]$ stands for the following program

$$
\begin{array}{rcl}
I_1 & : & T(l_1, 1) \\
 & \vdots & \\
I_n & : & T(l_n, n) \\
I_{n+1} & : & Z(n+1) \\
 & \vdots & \\
I_{\rho(P)} & : & Z(\rho(P)) \\
\_ & : & P \\
\_ & : & T(1, l)
\end{array}
$$

# Definability of Composition

Let $F, G_1, \ldots, G_k$ be programs that compute $f, g_1, \ldots, g_k$.

Let $m$ be $\max\{n, k, \rho(F), \rho(G_1), \ldots, \rho(G_k)\}$.

# Definability of Composition

Let $F, G_1, \ldots, G_k$ be programs that compute $f, g_1, \ldots, g_k$.

Let $m$ be $\max\{n, k, \rho(F), \rho(G_1), \ldots, \rho(G_k)\}$.

Registers:

$$[\ldots]_1^m [\widetilde{x}]_{m+1}^{m+n} [g_1(\widetilde{x})]_{m+n+1}^{m+n+1} \cdots [g_k(\widetilde{x})]_{m+n+k}^{m+n+k}$$

# Definability of Composition

The program for $h$:

$$
\begin{array}{rcl}
I_1 & : & T(1, m+1) \\
 & \vdots & \\
I_n & : & T(n, m+n) \\
I_{n+1} & : & G_1[m+1, m+2, \ldots, m+n \ \rightarrow \ m+n+1] \\
 & \vdots & \\
I_{n+k} & : & G_k[m+1, m+2, \ldots, m+n \ \rightarrow \ m+n+k] \\
I_{n+k+1} & : & F[m+n+1 \ldots, m+n+k \ \rightarrow \ 1]
\end{array}
$$

# Definability of Recursion

**Fact**. Suppose $f(\widetilde{x})$ and $g(\widetilde{x}, y, z)$ are URM-definable.
The recursion function $h(\widetilde{x}, y)$ defined by the following recursion

$$\begin{aligned} h(\widetilde{x}, 0) &\simeq f(\widetilde{x}), \\ h(\widetilde{x}, y+1) &\simeq g(\widetilde{x}, y, h(\widetilde{x}, y)) \end{aligned}$$

is URM-definable.

# Definability of Recursion

Let $F$ compute $f$ and $G$ compute $g$. Let $m$ be $\max\{n, \rho(F), \rho(G)\}$.

# Definability of Recursion

Let $F$ compute $f$ and $G$ compute $g$. Let $m$ be $\max\{n, \rho(F), \rho(G)\}$.

Registers: $[\ldots]_1^m [\widetilde{x}]_{m+1}^{m+n} [y]_{m+n+1}^{m+n+1} [k]_{m+n+2}^{m+n+2} [h(\widetilde{x}, k)]_{m+n+3}^{m+n+3}$.

# Definability of Recursion

Let $F$ compute $f$ and $G$ compute $g$. Let $m$ be $\max\{n, \rho(F), \rho(G)\}$.

Registers: $[\ldots]_1^m [\widetilde{x}]_{m+1}^{m+n} [y]_{m+n+1}^{m+n+1} [k]_{m+n+2}^{m+n+2} [h(\widetilde{x}, k)]_{m+n+3}^{m+n+3}$.

Program:

$$
\begin{array}{rcl}
I_1 & : & T(1, m+1) \\
 & & \vdots \\
I_{n+1} & : & T(n+1, m+n+1) \\
I_{n+2} & : & F[1, 2, \ldots, n \rightarrow m+n+3] \\
I_{n+3} & : & J(m+n+2, m+n+1, n+7) \\
I_{n+4} & : & G[m+1, \ldots, m+n, m+n+2, m+n+3 \rightarrow m+n+3] \\
I_{n+5} & : & S(m+n+2) \\
I_{n+6} & : & J(1, 1, n+3) \\
I_{n+7} & : & T(m+n+3, 1)
\end{array}
$$

# Definability of Minimization

**Fact**. If $f(\widetilde{x}, y)$ is URM-definable, then the minimization function $\mu y(f(\widetilde{x}, y) = 0)$ is URM-definable.

# Definability of Minimization

Suppose $F$ computes $f(\widetilde{x}, y)$. Let $m$ be $\max\{n+1, \rho(F)\}$.

# Definability of Minimization

Suppose $F$ computes $f(\widetilde{x}, y)$. Let $m$ be $\max\{n+1, \rho(F)\}$.

Registers: $[\ldots]_1^m [\widetilde{x}]_{m+1}^{m+n} [k]_{m+n+1}^{m+n+1} [0]_{m+n+2}^{m+n+2}$.

# Definability of Minimization

Suppose $F$ computes $f(\widetilde{x}, y)$. Let $m$ be $\max\{n+1, \rho(F)\}$.

Registers: $[\ldots]_1^m [\widetilde{x}]_{m+1}^{m+n} [k]_{m+n+1}^{m+n+1} [0]_{m+n+2}^{m+n+2}$.

Program:

$$
\begin{array}{rcl}
I_1 & : & T(1, m+1) \\
 & & \vdots \\
I_n & : & T(n, m+n) \\
I_{n+1} & : & F[m+1, m+2, \ldots, m+n+1 \ \rightarrow \ 1] \\
I_{n+2} & : & J(1, m+n+2, n+5) \\
I_{n+3} & : & S(m+n+1) \\
I_{n+4} & : & J(1, 1, n+1) \\
I_{n+5} & : & T(m+n+1, 1)
\end{array}
$$

# Main Result

**Theorem**. All recursive functions are URM-definable.

# Homework

- Read the proof that Ackermann function is not primitive.
- Try to solve the exercises in Chapter 1 & 2 as many as possible.

# Computability Theory V

## Turing Machine

Guoqiang Li

Shanghai Jiao Tong University

Oct. 24, 2014

*Assignment 2 is announced! (deadline Nov. 7)*

Turing Machine

# Alan Turing

**Alan Turing** (23Jun.1912-7Jun.1954), an English student of Church, introduced a machine model for effective calculation in

"On Computable Numbers, with an Application to the Entsheidungs problem",

Proc. of the London Mathematical Society, **42**:230-265, 1936.

Turing Machine, Halting Problem, Turing Test

British Prime Minister Gordon Brown:

"…I am pleased to have the chance to say how deeply sorry I and we all are for what happened to him ... So on behalf of the British government, and all those who live freely thanks to Alan's work, I am very proud to say: we're sorry, you deserved so much better."

# Motivation

What are necessary for a machine to calculate a function?

# Motivation

What are necessary for a machine to calculate a function?

- The machine should be able to interpret numbers;
- The machine must be able to operate and manipulate numbers according to a set of predefined instructions;

# Motivation

What are necessary for a machine to calculate a function?

- The machine should be able to interpret numbers;
- The machine must be able to operate and manipulate numbers according to a set of predefined instructions;

and

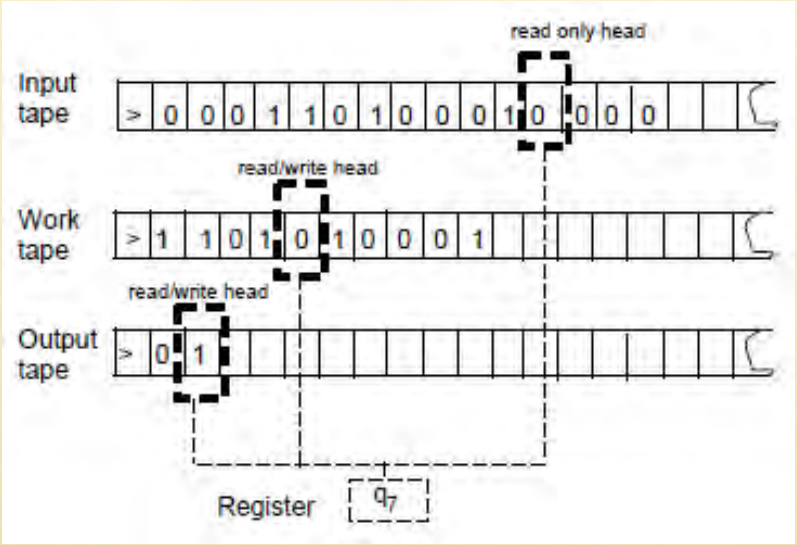
- The input number has to be stored in an accessible place;
- There should be an accessible place for the machine to store the intermediate results;
- The output number has to be put in an accessible place.

# Turing Machine

A $k$-tape Turing Machine $\mathbb{M}$ has $k$-tapes such that

- The first tape is the read-only input tape.
- The other $k-1$ tapes are the read/write work tapes.
- The $k$-th tape is also used as the output tape.

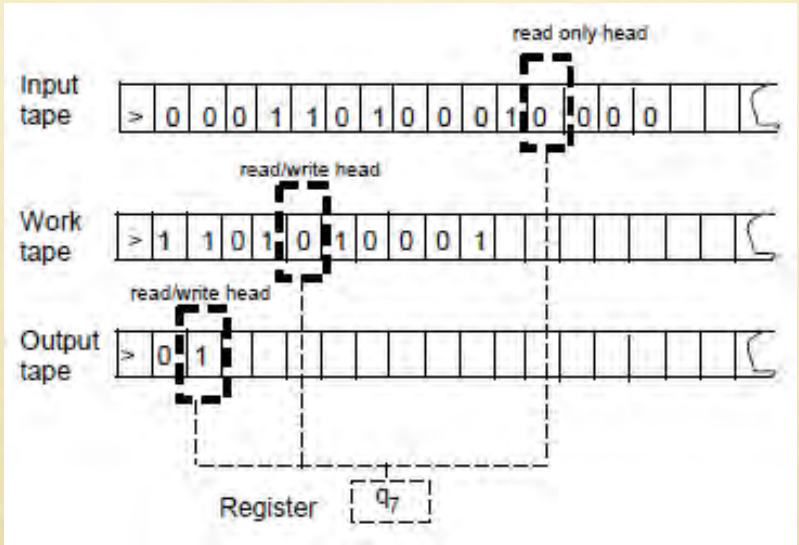
Every tape comes with a read/write head.



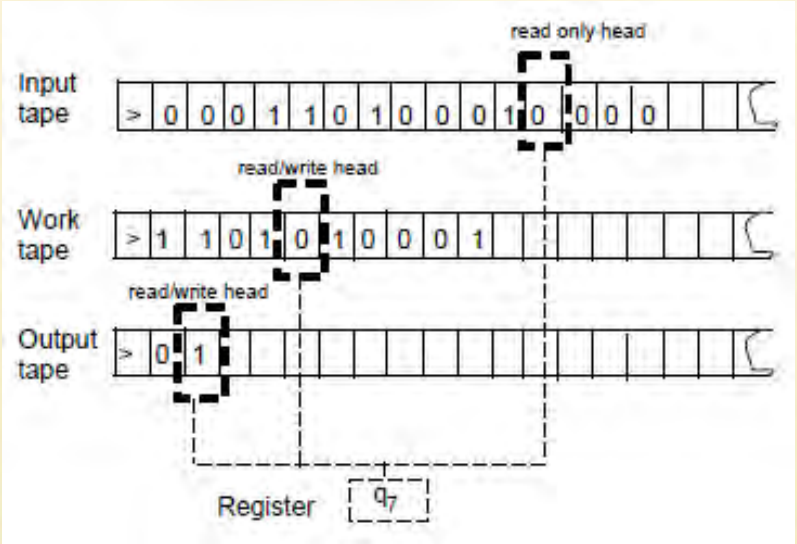
# Turing Machine

The machine is described by a tuple $(\Gamma, Q, \delta)$ containing

- A finite set $\Gamma$, called alphabet, of symbols. It contains a blank symbol $\square$, a start symbol $\triangleright$, and the digits $0$ and $1$.
- A finite set $Q$ of states. It contains a start state $q_{\mathrm{s}}$ and a halting state $q_{\mathrm{h}}$.
- A transition function $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\leftarrow, -, \rightarrow\}^k$, describing the rules of each computation step.



# Computation and Configuration




Configuration, initial configuration, final configuration, computation step

# An Example

$Q = \{q_s, q_h, q_1, q_2, q_3\}$, $\Gamma = \{0, 1, \square, \triangleright\}$,
and $\delta$ is as follows:

| $p \in Q$ | $\sigma \in \Gamma$ | $\delta(p, \sigma)$ |
|---|---|---|
| $q_s$ | $0$ | $(q_s, 0, \rightarrow)$ |
| $q_s$ | $1$ | $(q_s, 1, \rightarrow)$ |
| $q_s$ | $\square$ | $(q_1, \square, \leftarrow)$ |
| $q_s$ | $\triangleright$ | $(q_s, \triangleright, \rightarrow)$ |
| $q_1$ | $0$ | $(q_2, \square, \rightarrow)$ |
| $q_1$ | $1$ | $(q_3, \square, \rightarrow)$ |
| $q_1$ | $\square$ | $(q_1, \square, -)$ |
| $q_1$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_2$ | $0$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $1$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\square$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_3$ | $0$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $1$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\square$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |

# An Example

$Q = \{q_s, q_h, q_1, q_2, q_3\}$, $\Gamma = \{0, 1, \square, \triangleright\}$,
and $\delta$ is as follows:

| $p \in Q$ | $\sigma \in \Gamma$ | $\delta(p, \sigma)$ |
|---|---|---|
| $q_s$ | $0$ | $(q_s, 0, \rightarrow)$ |
| $q_s$ | $1$ | $(q_s, 1, \rightarrow)$ |
| $q_s$ | $\square$ | $(q_1, \square, \leftarrow)$ |
| $q_s$ | $\triangleright$ | $(q_s, \triangleright, \rightarrow)$ |
| $q_1$ | $0$ | $(q_2, \square, \rightarrow)$ |
| $q_1$ | $1$ | $(q_3, \square, \rightarrow)$ |
| $q_1$ | $\square$ | $(q_1, \square, -)$ |
| $q_1$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_2$ | $0$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $1$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\square$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_3$ | $0$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $1$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\square$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |

Start the machine with input 010

# An Example

$Q = \{q_s, q_h, q_1, q_2, q_3\}$, $\Gamma = \{0, 1, \square, \triangleright\}$,
and $\delta$ is as follows:

| $p \in Q$ | $\sigma \in \Gamma$ | $\delta(p, \sigma)$ |
|---|---|---|
| $q_s$ | 0 | $(q_s, 0, \rightarrow)$ |
| $q_s$ | 1 | $(q_s, 1, \rightarrow)$ |
| $q_s$ | $\square$ | $(q_1, \square, \leftarrow)$ |
| $q_s$ | $\triangleright$ | $(q_s, \triangleright, \rightarrow)$ |
| $q_1$ | 0 | $(q_2, \square, \rightarrow)$ |
| $q_1$ | 1 | $(q_3, \square, \rightarrow)$ |
| $q_1$ | $\square$ | $(q_1, \square, -)$ |
| $q_1$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_2$ | 0 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | 1 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\square$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_3$ | 0 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | 1 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\square$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |

Start the machine with input 010

- $q_s$, $\underline{\triangleright}010$

# An Example

$Q = \{q_s, q_h, q_1, q_2, q_3\}$, $\Gamma = \{0, 1, \square, \triangleright\}$,
and $\delta$ is as follows:

| $p \in Q$ | $\sigma \in \Gamma$ | $\delta(p, \sigma)$ |
|---|---|---|
| $q_s$ | 0 | $(q_s, 0, \rightarrow)$ |
| $q_s$ | 1 | $(q_s, 1, \rightarrow)$ |
| $q_s$ | $\square$ | $(q_1, \square, \leftarrow)$ |
| $q_s$ | $\triangleright$ | $(q_s, \triangleright, \rightarrow)$ |
| $q_1$ | 0 | $(q_2, \square, \rightarrow)$ |
| $q_1$ | 1 | $(q_3, \square, \rightarrow)$ |
| $q_1$ | $\square$ | $(q_1, \square, -)$ |
| $q_1$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_2$ | 0 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | 1 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\square$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_3$ | 0 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | 1 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\square$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |

Start the machine with input 010

- $q_s$, $\underline{\triangleright}010$
- $q_s$, $\triangleright\underline{0}10$

# An Example

$Q = \{q_s, q_h, q_1, q_2, q_3\}$, $\Gamma = \{0, 1, \square, \triangleright\}$,
and $\delta$ is as follows:

| $p \in Q$ | $\sigma \in \Gamma$ | $\delta(p, \sigma)$ |
|---|---|---|
| $q_s$ | 0 | $(q_s, 0, \rightarrow)$ |
| $q_s$ | 1 | $(q_s, 1, \rightarrow)$ |
| $q_s$ | $\square$ | $(q_1, \square, \leftarrow)$ |
| $q_s$ | $\triangleright$ | $(q_s, \triangleright, \rightarrow)$ |
| $q_1$ | 0 | $(q_2, \square, \rightarrow)$ |
| $q_1$ | 1 | $(q_3, \square, \rightarrow)$ |
| $q_1$ | $\square$ | $(q_1, \square, -)$ |
| $q_1$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_2$ | 0 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | 1 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\square$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_3$ | 0 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | 1 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\square$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |

Start the machine with input 010

- $q_s, \underline{\triangleright}010$
- $q_s, \triangleright\underline{0}10$
- $q_s, \triangleright0\underline{1}0$

# An Example

$Q = \{q_s, q_h, q_1, q_2, q_3\}$, $\Gamma = \{0, 1, \square, \triangleright\}$,
and $\delta$ is as follows:

| $p \in Q$ | $\sigma \in \Gamma$ | $\delta(p, \sigma)$ |
|---|---|---|
| $q_s$ | 0 | $(q_s, 0, \rightarrow)$ |
| $q_s$ | 1 | $(q_s, 1, \rightarrow)$ |
| $q_s$ | $\square$ | $(q_1, \square, \leftarrow)$ |
| $q_s$ | $\triangleright$ | $(q_s, \triangleright, \rightarrow)$ |
| $q_1$ | 0 | $(q_2, \square, \rightarrow)$ |
| $q_1$ | 1 | $(q_3, \square, \rightarrow)$ |
| $q_1$ | $\square$ | $(q_1, \square, -)$ |
| $q_1$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_2$ | 0 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | 1 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\square$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_3$ | 0 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | 1 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\square$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |

Start the machine with input 010

- $q_s, \underline{\triangleright}010$
- $q_s, \triangleright\underline{0}10$
- $q_s, \triangleright0\underline{1}0$
- $q_s, \triangleright01\underline{0}$

# An Example

$Q = \{q_s, q_h, q_1, q_2, q_3\}$, $\Gamma = \{0, 1, \square, \triangleright\}$,
and $\delta$ is as follows:

| $p \in Q$ | $\sigma \in \Gamma$ | $\delta(p, \sigma)$ |
|---|---|---|
| $q_s$ | $0$ | $(q_s, 0, \rightarrow)$ |
| $q_s$ | $1$ | $(q_s, 1, \rightarrow)$ |
| $q_s$ | $\square$ | $(q_1, \square, \leftarrow)$ |
| $q_s$ | $\triangleright$ | $(q_s, \triangleright, \rightarrow)$ |
| $q_1$ | $0$ | $(q_2, \square, \rightarrow)$ |
| $q_1$ | $1$ | $(q_3, \square, \rightarrow)$ |
| $q_1$ | $\square$ | $(q_1, \square, -)$ |
| $q_1$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_2$ | $0$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $1$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\square$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_3$ | $0$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $1$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\square$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |

Start the machine with input 010

- $q_s, \underline{\triangleright}010$
- $q_s, \triangleright\underline{0}10$
- $q_s, \triangleright0\underline{1}0$
- $q_s, \triangleright01\underline{0}$
- $q_s, \triangleright010\underline{\square}$

# An Example

$Q = \{q_s, q_h, q_1, q_2, q_3\}$, $\Gamma = \{0, 1, \square, \triangleright\}$,
and $\delta$ is as follows:

| $p \in Q$ | $\sigma \in \Gamma$ | $\delta(p, \sigma)$ |
|---|---|---|
| $q_s$ | 0 | $(q_s, 0, \rightarrow)$ |
| $q_s$ | 1 | $(q_s, 1, \rightarrow)$ |
| $q_s$ | $\square$ | $(q_1, \square, \leftarrow)$ |
| $q_s$ | $\triangleright$ | $(q_s, \triangleright, \rightarrow)$ |
| $q_1$ | 0 | $(q_2, \square, \rightarrow)$ |
| $q_1$ | 1 | $(q_3, \square, \rightarrow)$ |
| $q_1$ | $\square$ | $(q_1, \square, -)$ |
| $q_1$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_2$ | 0 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | 1 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\square$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_3$ | 0 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | 1 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\square$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |

Start the machine with input 010

- $q_s, \underline{\triangleright}010$
- $q_s, \triangleright\underline{0}10$
- $q_s, \triangleright0\underline{1}0$
- $q_s, \triangleright01\underline{0}$
- $q_s, \triangleright010\underline{\square}$
- $q_1, \triangleright01\underline{0}\square$

# An Example

$Q = \{q_s, q_h, q_1, q_2, q_3\}$, $\Gamma = \{0, 1, \square, \triangleright\}$,
and $\delta$ is as follows:

| $p \in Q$ | $\sigma \in \Gamma$ | $\delta(p, \sigma)$ |
|---|---|---|
| $q_s$ | 0 | $(q_s, 0, \rightarrow)$ |
| $q_s$ | 1 | $(q_s, 1, \rightarrow)$ |
| $q_s$ | $\square$ | $(q_1, \square, \leftarrow)$ |
| $q_s$ | $\triangleright$ | $(q_s, \triangleright, \rightarrow)$ |
| $q_1$ | 0 | $(q_2, \square, \rightarrow)$ |
| $q_1$ | 1 | $(q_3, \square, \rightarrow)$ |
| $q_1$ | $\square$ | $(q_1, \square, -)$ |
| $q_1$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_2$ | 0 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | 1 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\square$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_3$ | 0 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | 1 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\square$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |

Start the machine with input 010

- $q_s, \underline{\triangleright}010$
- $q_s, \triangleright\underline{0}10$
- $q_s, \triangleright0\underline{1}0$
- $q_s, \triangleright01\underline{0}$
- $q_s, \triangleright010\underline{\square}$
- $q_1, \triangleright01\underline{0}\square$
- $q_2, \triangleright01\square\underline{\square}$

# An Example

$Q = \{q_s, q_h, q_1, q_2, q_3\}$, $\Gamma = \{0, 1, \square, \triangleright\}$,
and $\delta$ is as follows:

| $p \in Q$ | $\sigma \in \Gamma$ | $\delta(p, \sigma)$ |
|---|---|---|
| $q_s$ | 0 | $(q_s, 0, \rightarrow)$ |
| $q_s$ | 1 | $(q_s, 1, \rightarrow)$ |
| $q_s$ | $\square$ | $(q_1, \square, \leftarrow)$ |
| $q_s$ | $\triangleright$ | $(q_s, \triangleright, \rightarrow)$ |
| $q_1$ | 0 | $(q_2, \square, \rightarrow)$ |
| $q_1$ | 1 | $(q_3, \square, \rightarrow)$ |
| $q_1$ | $\square$ | $(q_1, \square, -)$ |
| $q_1$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_2$ | 0 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | 1 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\square$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_3$ | 0 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | 1 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\square$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |

Start the machine with input 010

- $q_s, \underline{\triangleright}010$
- $q_s, \triangleright\underline{0}10$
- $q_s, \triangleright0\underline{1}0$
- $q_s, \triangleright01\underline{0}$
- $q_s, \triangleright010\underline{\square}$
- $q_1, \triangleright01\underline{0}\square$
- $q_2, \triangleright01\square\underline{\square}$
- $q_s, \triangleright01\underline{\square}0$

# An Example

$Q = \{q_s, q_h, q_1, q_2, q_3\}$, $\Gamma = \{0, 1, \square, \triangleright\}$,
and $\delta$ is as follows:

| $p \in Q$ | $\sigma \in \Gamma$ | $\delta(p, \sigma)$ |
|---|---|---|
| $q_s$ | 0 | $(q_s, 0, \rightarrow)$ |
| $q_s$ | 1 | $(q_s, 1, \rightarrow)$ |
| $q_s$ | $\square$ | $(q_1, \square, \leftarrow)$ |
| $q_s$ | $\triangleright$ | $(q_s, \triangleright, \rightarrow)$ |
| $q_1$ | 0 | $(q_2, \square, \rightarrow)$ |
| $q_1$ | 1 | $(q_3, \square, \rightarrow)$ |
| $q_1$ | $\square$ | $(q_1, \square, -)$ |
| $q_1$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_2$ | 0 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | 1 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\square$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_3$ | 0 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | 1 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\square$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |

Start the machine with input 010

- $q_s, \underline{\triangleright}010$
- $q_s, \triangleright\underline{0}10$
- $q_s, \triangleright0\underline{1}0$
- $q_s, \triangleright01\underline{0}$
- $q_s, \triangleright010\underline{\square}$
- $q_1, \triangleright01\underline{0}\square$
- $q_2, \triangleright01\square\underline{\square}$
- $q_s, \triangleright01\underline{\square}0$
- $q_1, \triangleright0\underline{1}\square0$

# An Example

$Q = \{q_s, q_h, q_1, q_2, q_3\}$, $\Gamma = \{0, 1, \square, \triangleright\}$,
and $\delta$ is as follows:

| $p \in Q$ | $\sigma \in \Gamma$ | $\delta(p, \sigma)$ |
|---|---|---|
| $q_s$ | 0 | $(q_s, 0, \rightarrow)$ |
| $q_s$ | 1 | $(q_s, 1, \rightarrow)$ |
| $q_s$ | $\square$ | $(q_1, \square, \leftarrow)$ |
| $q_s$ | $\triangleright$ | $(q_s, \triangleright, \rightarrow)$ |
| $q_1$ | 0 | $(q_2, \square, \rightarrow)$ |
| $q_1$ | 1 | $(q_3, \square, \rightarrow)$ |
| $q_1$ | $\square$ | $(q_1, \square, -)$ |
| $q_1$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_2$ | 0 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | 1 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\square$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_3$ | 0 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | 1 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\square$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |

Start the machine with input 010

- $q_s, \underline{\triangleright}010$
- $q_s, \triangleright\underline{0}10$
- $q_s, \triangleright0\underline{1}0$
- $q_s, \triangleright01\underline{0}$
- $q_s, \triangleright010\underline{\square}$
- $q_1, \triangleright01\underline{0}\square$
- $q_2, \triangleright01\square\underline{\square}$
- $q_s, \triangleright01\underline{\square}0$
- $q_1, \triangleright0\underline{1}\square0$
- $q_3, \triangleright0\square\underline{\square}0$

# An Example

$Q = \{q_s, q_h, q_1, q_2, q_3\}$, $\Gamma = \{0, 1, \square, \triangleright\}$,
and $\delta$ is as follows:

| $p \in Q$ | $\sigma \in \Gamma$ | $\delta(p, \sigma)$ |
|---|---|---|
| $q_s$ | 0 | $(q_s, 0, \rightarrow)$ |
| $q_s$ | 1 | $(q_s, 1, \rightarrow)$ |
| $q_s$ | $\square$ | $(q_1, \square, \leftarrow)$ |
| $q_s$ | $\triangleright$ | $(q_s, \triangleright, \rightarrow)$ |
| $q_1$ | 0 | $(q_2, \square, \rightarrow)$ |
| $q_1$ | 1 | $(q_3, \square, \rightarrow)$ |
| $q_1$ | $\square$ | $(q_1, \square, -)$ |
| $q_1$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_2$ | 0 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | 1 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\square$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_3$ | 0 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | 1 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\square$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |

Start the machine with input 010

- $q_s, \underline{\triangleright}010$
- $q_s, \triangleright\underline{0}10$
- $q_s, \triangleright0\underline{1}0$
- $q_s, \triangleright01\underline{0}$
- $q_s, \triangleright010\underline{\square}$
- $q_1, \triangleright01\underline{0}\square$
- $q_2, \triangleright01\square\underline{\square}$
- $q_s, \triangleright01\underline{\square}0$
- $q_1, \triangleright0\underline{1}\square0$
- $q_3, \triangleright0\square\underline{\square}0$
- $q_s, \triangleright0\underline{\square}10$

# An Example

$Q = \{q_s, q_h, q_1, q_2, q_3\}$, $\Gamma = \{0, 1, \square, \triangleright\}$,
and $\delta$ is as follows:

| $p \in Q$ | $\sigma \in \Gamma$ | $\delta(p, \sigma)$ |
|---|---|---|
| $q_s$ | 0 | $(q_s, 0, \rightarrow)$ |
| $q_s$ | 1 | $(q_s, 1, \rightarrow)$ |
| $q_s$ | $\square$ | $(q_1, \square, \leftarrow)$ |
| $q_s$ | $\triangleright$ | $(q_s, \triangleright, \rightarrow)$ |
| $q_1$ | 0 | $(q_2, \square, \rightarrow)$ |
| $q_1$ | 1 | $(q_3, \square, \rightarrow)$ |
| $q_1$ | $\square$ | $(q_1, \square, -)$ |
| $q_1$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_2$ | 0 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | 1 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\square$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_3$ | 0 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | 1 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\square$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |

Start the machine with input 010

- $q_s, \underline{\triangleright}010$
- $q_s, \triangleright\underline{0}10$
- $q_s, \triangleright0\underline{1}0$
- $q_s, \triangleright01\underline{0}$
- $q_s, \triangleright010\underline{\square}$
- $q_1, \triangleright01\underline{0}\square$
- $q_2, \triangleright01\square\underline{\square}$
- $q_s, \triangleright01\underline{\square}0$
- $q_1, \triangleright0\underline{1}\square0$
- $q_3, \triangleright0\square\underline{\square}0$
- $q_s, \triangleright0\underline{\square}10$
- $q_1, \triangleright\underline{0}\square10$

# An Example

$Q = \{q_s, q_h, q_1, q_2, q_3\}$, $\Gamma = \{0, 1, \square, \triangleright\}$,
and $\delta$ is as follows:

| $p \in Q$ | $\sigma \in \Gamma$ | $\delta(p, \sigma)$ |
|---|---|---|
| $q_s$ | 0 | $(q_s, 0, \rightarrow)$ |
| $q_s$ | 1 | $(q_s, 1, \rightarrow)$ |
| $q_s$ | $\square$ | $(q_1, \square, \leftarrow)$ |
| $q_s$ | $\triangleright$ | $(q_s, \triangleright, \rightarrow)$ |
| $q_1$ | 0 | $(q_2, \square, \rightarrow)$ |
| $q_1$ | 1 | $(q_3, \square, \rightarrow)$ |
| $q_1$ | $\square$ | $(q_1, \square, -)$ |
| $q_1$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_2$ | 0 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | 1 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\square$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_3$ | 0 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | 1 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\square$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |

Start the machine with input 010

- $q_s, \underline{\triangleright}010$
- $q_s, \triangleright\underline{0}10$
- $q_s, \triangleright0\underline{1}0$
- $q_s, \triangleright01\underline{0}$
- $q_s, \triangleright010\underline{\square}$
- $q_1, \triangleright01\underline{0}\square$
- $q_2, \triangleright01\square\underline{\square}$
- $q_s, \triangleright01\underline{\square}0$
- $q_1, \triangleright0\underline{1}\square0$
- $q_3, \triangleright0\square\underline{\square}0$
- $q_s, \triangleright0\underline{\square}10$
- $q_1, \triangleright\underline{0}\square10$
- $q_2, \triangleright\square\underline{\square}10$

# An Example

$Q = \{q_s, q_h, q_1, q_2, q_3\}$, $\Gamma = \{0, 1, \square, \triangleright\}$,
and $\delta$ is as follows:

| $p \in Q$ | $\sigma \in \Gamma$ | $\delta(p, \sigma)$ |
|---|---|---|
| $q_s$ | 0 | $(q_s, 0, \rightarrow)$ |
| $q_s$ | 1 | $(q_s, 1, \rightarrow)$ |
| $q_s$ | $\square$ | $(q_1, \square, \leftarrow)$ |
| $q_s$ | $\triangleright$ | $(q_s, \triangleright, \rightarrow)$ |
| $q_1$ | 0 | $(q_2, \square, \rightarrow)$ |
| $q_1$ | 1 | $(q_3, \square, \rightarrow)$ |
| $q_1$ | $\square$ | $(q_1, \square, -)$ |
| $q_1$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_2$ | 0 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | 1 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\square$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_3$ | 0 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | 1 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\square$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |

Start the machine with input 010

- $q_s, \underline{\triangleright}010$
- $q_s, \triangleright\underline{0}10$
- $q_s, \triangleright0\underline{1}0$
- $q_s, \triangleright01\underline{0}$
- $q_s, \triangleright010\underline{\square}$
- $q_1, \triangleright01\underline{0}\square$
- $q_2, \triangleright01\square\underline{\square}$
- $q_s, \triangleright01\underline{\square}0$
- $q_1, \triangleright0\underline{1}\square0$
- $q_3, \triangleright0\square\underline{\square}0$
- $q_s, \triangleright0\underline{\square}10$
- $q_1, \triangleright\underline{0}\square10$
- $q_2, \triangleright\square\underline{\square}10$
- $q_0, \triangleright\underline{\square}010$

# An Example

$Q = \{q_s, q_h, q_1, q_2, q_3\}$, $\Gamma = \{0, 1, \square, \triangleright\}$,
and $\delta$ is as follows:

| $p \in Q$ | $\sigma \in \Gamma$ | $\delta(p, \sigma)$ |
|---|---|---|
| $q_s$ | 0 | $(q_s, 0, \rightarrow)$ |
| $q_s$ | 1 | $(q_s, 1, \rightarrow)$ |
| $q_s$ | $\square$ | $(q_1, \square, \leftarrow)$ |
| $q_s$ | $\triangleright$ | $(q_s, \triangleright, \rightarrow)$ |
| $q_1$ | 0 | $(q_2, \square, \rightarrow)$ |
| $q_1$ | 1 | $(q_3, \square, \rightarrow)$ |
| $q_1$ | $\square$ | $(q_1, \square, -)$ |
| $q_1$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_2$ | 0 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | 1 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\square$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_3$ | 0 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | 1 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\square$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |

Start the machine with input 010

- $q_s, \underline{\triangleright}010$
- $q_s, \triangleright\underline{0}10$
- $q_s, \triangleright0\underline{1}0$
- $q_s, \triangleright01\underline{0}$
- $q_s, \triangleright010\underline{\square}$
- $q_1, \triangleright01\underline{0}\square$
- $q_2, \triangleright01\square\underline{\square}$
- $q_s, \triangleright01\underline{\square}0$
- $q_1, \triangleright0\underline{1}\square0$
- $q_3, \triangleright0\square\underline{\square}0$
- $q_s, \triangleright0\underline{\square}10$
- $q_1, \triangleright\underline{0}\square10$
- $q_2, \triangleright\square\underline{\square}10$
- $q_0, \triangleright\underline{\square}010$
- $q_1, \underline{\triangleright}\square010$

# An Example

$Q = \{q_s, q_h, q_1, q_2, q_3\}$, $\Gamma = \{0, 1, \square, \triangleright\}$,
and $\delta$ is as follows:

| $p \in Q$ | $\sigma \in \Gamma$ | $\delta(p, \sigma)$ |
|---|---|---|
| $q_s$ | 0 | $(q_s, 0, \rightarrow)$ |
| $q_s$ | 1 | $(q_s, 1, \rightarrow)$ |
| $q_s$ | $\square$ | $(q_1, \square, \leftarrow)$ |
| $q_s$ | $\triangleright$ | $(q_s, \triangleright, \rightarrow)$ |
| $q_1$ | 0 | $(q_2, \square, \rightarrow)$ |
| $q_1$ | 1 | $(q_3, \square, \rightarrow)$ |
| $q_1$ | $\square$ | $(q_1, \square, -)$ |
| $q_1$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_2$ | 0 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | 1 | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\square$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_3$ | 0 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | 1 | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\square$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |

Start the machine with input 010

- $q_s, \underline{\triangleright}010$
- $q_s, \triangleright\underline{0}10$
- $q_s, \triangleright0\underline{1}0$
- $q_s, \triangleright01\underline{0}$
- $q_s, \triangleright010\underline{\square}$
- $q_1, \triangleright01\underline{0}\square$
- $q_2, \triangleright01\square\underline{\square}$
- $q_s, \triangleright01\underline{\square}0$
- $q_1, \triangleright0\underline{1}\square0$
- $q_3, \triangleright0\square\underline{\square}0$
- $q_s, \triangleright0\underline{\square}10$
- $q_1, \triangleright\underline{0}\square10$
- $q_2, \triangleright\square\underline{\square}10$
- $q_0, \triangleright\underline{\square}010$
- $q_1, \underline{\triangleright}\square010$
- $q_h, \triangleright\underline{\square}010$

# The Second Example

$Q = \{q_s, q_h, q_1\}$, $\Gamma = \{0, 1, \square, \triangleright\}$, and $\delta$ is as follows:

| $p \in Q$ | $\sigma \in \Gamma$ | $\delta(p, \sigma)$ |
|---|---|---|
| $q_s$ | $0$ | $(q_s, 0, \rightarrow)$ |
| $q_s$ | $1$ | $(q_s, 1, \rightarrow)$ |
| $q_s$ | $\square$ | $(q_1, \square, \leftarrow)$ |
| $q_s$ | $\triangleright$ | $(q_s, \triangleright, \rightarrow)$ |
| $q_1$ | $0$ | $(q_h, 1, -)$ |
| $q_1$ | $1$ | $(q_1, 0, \leftarrow)$ |
| $q_1$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |

# The Third Example

$Q = \{q_s, q_h, q_c, q_l, q_t\}$; $\Gamma = \{\square, \triangleright, 0, 1\}$; two work tapes.

# The Third Example

$Q = \{q_s, q_h, q_c, q_l, q_t\}$; $\Gamma = \{\square, \triangleright, 0, 1\}$; two work tapes.

$\langle q_s, \triangleright, \triangleright, \triangleright\rangle \to \langle q_c, \triangleright, \triangleright, \to, \to, \to\rangle$

$\langle q_c, 0, \square, \square\rangle \to \langle q_c, 0, \square, \to, \to, -\rangle$
$\langle q_c, 1, \square, \square\rangle \to \langle q_c, 1, \square, \to, \to, -\rangle$
$\langle q_c, \square, \square, \square\rangle \to \langle q_l, \square, \square, \leftarrow, -, -\rangle$

$\langle q_l, 0, \square, \square\rangle \to \langle q_l, \square, \square, \leftarrow, -, -\rangle$
$\langle q_l, 1, \square, \square\rangle \to \langle q_l, \square, \square, \leftarrow, -, -\rangle$
$\langle q_l, \triangleright, \square, \square\rangle \to \langle q_t, \square, \square, \to, \leftarrow, -\rangle$

$\langle q_t, \square, \triangleright, \square\rangle \to \langle q_h, \triangleright, 1, -, -, -\rangle$
$\langle q_t, 0, 1, \square\rangle \to \langle q_h, 1, 0, -, -, -\rangle$
$\langle q_t, 1, 0, \square\rangle \to \langle q_h, 0, 0, -, -, -\rangle$
$\langle q_t, 0, 0, \square\rangle \to \langle q_t, 0, \square, \to, \leftarrow, -\rangle$
$\langle q_t, 1, 1, \square\rangle \to \langle q_t, 1, \square, \to, \leftarrow, -\rangle$

# $\{0, 1, \square, \triangleright\}$ vs. Larger Alphabets

Suppose $\mathbb{M}$ has $k$ tapes with the alphabet $\Gamma$.

# $\{0, 1, \square, \triangleright\}$ vs. Larger Alphabets

Suppose $\mathbb{M}$ has $k$ tapes with the alphabet $\Gamma$.

A symbol of $\mathbb{M}$ is encoded by a string $\sigma \in \{0, 1\}^*$ of length $\log |\Gamma|$.

# $\{0, 1, \square, \triangleright\}$ vs. Larger Alphabets

Suppose $\mathbb{M}$ has $k$ tapes with the alphabet $\Gamma$.

A symbol of $\mathbb{M}$ is encoded by a string $\sigma \in \{0,1\}^*$ of length $\log|\Gamma|$.

States: A state $q$ is turned into states $q, \langle q, \sigma_1^1, \ldots, \sigma_1^k\rangle$ where $|\sigma_1^1| = \ldots = |\sigma_1^k| = 1$, ..., $\langle q, \sigma_{\log|\Gamma|}^1, \ldots, \sigma_{\log|\Gamma|}^k\rangle$ where $|\sigma_{\log|\Gamma|}^1| = \ldots = |\sigma_{\log|\Gamma|}^k| = \log|\Gamma|$.

# $\{0, 1, \square, \triangleright\}$ vs. Larger Alphabets

Suppose $\mathbb{M}$ has $k$ tapes with the alphabet $\Gamma$.

A symbol of $\mathbb{M}$ is encoded by a string $\sigma \in \{0,1\}^*$ of length $\log|\Gamma|$.

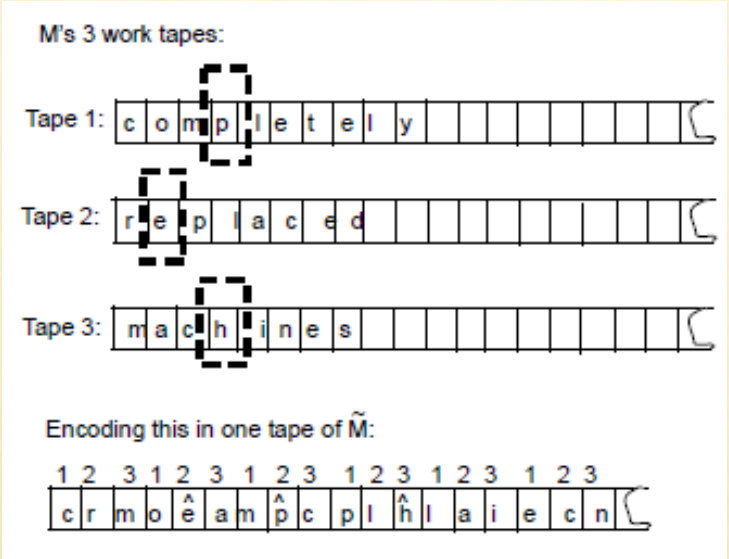
States: A state $q$ is turned into states $q, \langle q, \sigma_1^1, \ldots, \sigma_1^k\rangle$ where $|\sigma_1^1| = \ldots = |\sigma_1^k| = 1$, ..., $\langle q, \sigma_{\log|\Gamma|}^1, \ldots, \sigma_{\log|\Gamma|}^k\rangle$ where $|\sigma_{\log|\Gamma|}^1| = \ldots = |\sigma_{\log|\Gamma|}^k| = \log|\Gamma|$.

A computation step of $\mathbb{M}$ is simulated in $\widetilde{\mathbb{M}}$ by $\log|\Gamma|$ steps to read, $\log|\Gamma|$ steps to write, and $\log|\Gamma|$ steps to relocate the heads.

# One Tape vs. Many Tapes

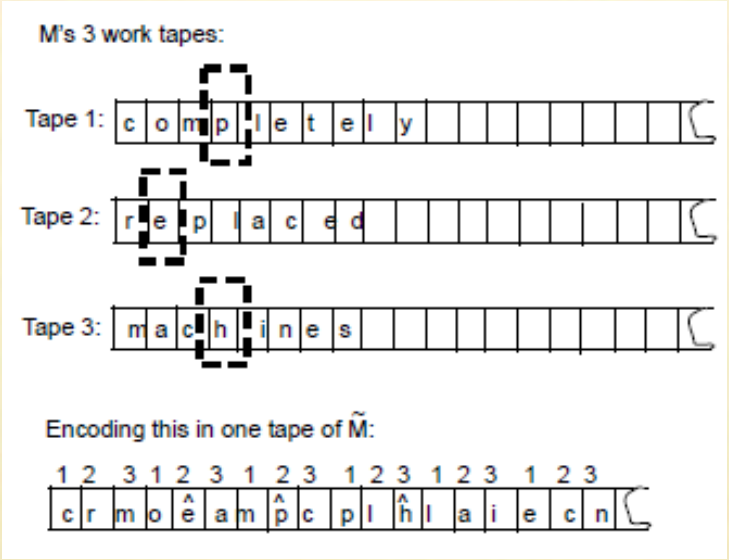
# One Tape vs. Many Tapes

The basic idea is to interleave $k$ tapes into one tape.
The first $n + 1$ cells are reserved for the input.



# One Tape vs. Many Tapes

The basic idea is to interleave $k$ tapes into one tape.
The first $n + 1$ cells are reserved for the input.




Every symbol $a$ of $\mathbb{M}$ is turned into two symbols $a, \widehat{a}$ in $\widetilde{\mathbb{M}}$, with $\widehat{a}$ used to indicate head position.

# One Tape vs. Many Tapes

# One Tape vs. Many Tapes

The machine $\widetilde{\mathbb{M}}$ copies the input bits to the first imaginary tape. The head then moves left to the $(n+2)$-th cell.

# One Tape vs. Many Tapes

The machine $\widetilde{\mathbb{M}}$ copies the input bits to the first imaginary tape. The head then moves left to the $(n+2)$-th cell.

Sweeping the tape cells from left to right. Record in the register the $k$ symbols marked with the hat $\widehat{\_}$.

# One Tape vs. Many Tapes

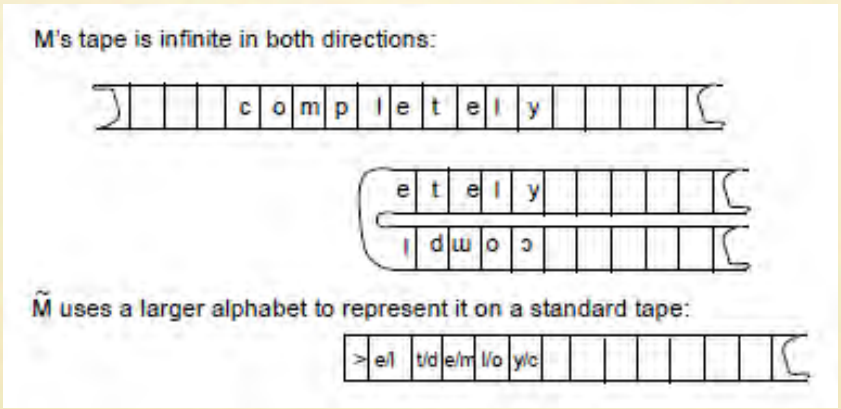
The machine $\widetilde{\mathbb{M}}$ copies the input bits to the first imaginary tape. The head then moves left to the $(n+2)$-th cell.

Sweeping the tape cells from left to right. Record in the register the $k$ symbols marked with the hat $\widehat{\_}$.

Sweeping the tape cells from right to left to update using the transitions of $\mathbb{M}$.

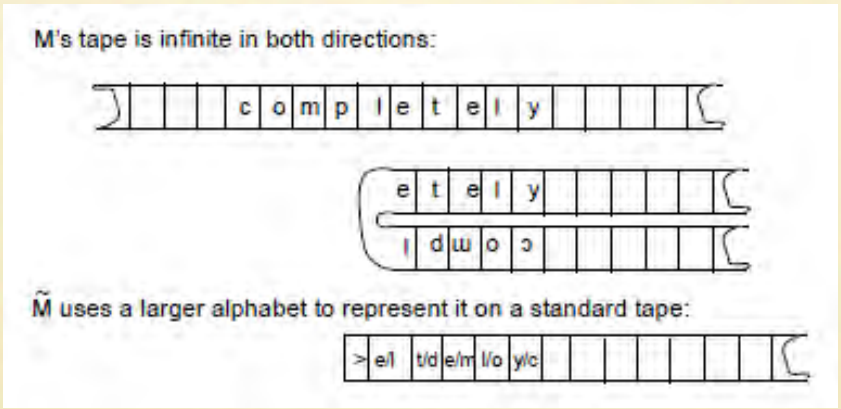
# One Unidirectional vs. Bidirectional Tape

The idea is that $\widetilde{\mathbb{M}}$ makes use of the alphabet $\Gamma \times \Gamma$.



# One Unidirectional vs. Bidirectional Tape

The idea is that $\widetilde{\mathbb{M}}$ makes use of the alphabet $\Gamma \times \Gamma$.




Every state $q$ of $\mathbb{M}$ is turned into $\overline{q}$ and $\underline{q}$.

# Simulation of TM by URM

# Simulating TM by URM

Suppose $\mathbb{M}$ is a $3$-tape TM with the alphabet $\{0, 1, \square, \triangleright\}$.

# Simulating TM by URM

Suppose $\mathbb{M}$ is a 3-tape TM with the alphabet $\{0, 1, \square, \triangleright\}$.

The URM that simulates $\mathbb{M}$ can be designed as follows:

- Suppose that $R_m$ is the right most register that is used by a program calculating $x \dot{-} 1$.
- The head positions are stored in $R_{m+1}, R_{m+2}, R_{m+3}$.
- The three binary strings in the tapes are stored respectively in
  $R_{m+4}, R_{m+7}, R_{m+10}, \ldots$,
  $R_{m+5}, R_{m+8}, R_{m+11}, \ldots$,
  $R_{m+6}, R_{m+9}, R_{m+12}, \ldots$.
- The states of $\mathbb{M}$ are encoded by the states of the URM.
- The transition function of $\mathbb{M}$ can be easily simulated by the program of the URM.

# Homework

Encode the addition function by $k$-tape Turing machine two nature number partitioned by $\sharp$ on the input tape, for example, 11010$\#$1001, and then try to encode the function by 1-tape Turing machine.

# Computability Theory VI

## Church-Turing Thesis

Guoqiang Li

Shanghai Jiao Tong University

Oct. 31, 2014

# Church-Turing Thesis

# Fundamental Question

How do computation models characterize the informal notion of effective computability?

# Fundamental Result

**Theorem**. The set of functions definable (the Turing Machine Model, the URM Model) is precisely the set of functions definable in the Recursive Function Model.

# Fundamental Result

**Theorem**. The set of functions definable (the Turing Machine Model, the URM Model) is precisely the set of functions definable in the Recursive Function Model.

*Proof*.
We showed that
$\mu$-definable $\Rightarrow$ $\lambda$-definable $\Rightarrow$ Turing definable $\Rightarrow$ URM-definable.

We will show that URM-definable $\Rightarrow$ $\mu$-definable.

# Church-Turing Thesis

**Church-Turing Thesis**.

The functions definable in all computation models are the same. They are precisely the computable functions.

# Church-Turing Thesis

**Church-Turing Thesis**.

The functions definable in all computation models are the same. They are precisely the computable functions.

1. Church believed that all computable functions are $\lambda$-definable.
2. Kleene termed it Church Thesis.
3. Gödel accepted it only after he saw Turing's equivalence proof.
4. Church-Turing Thesis is now universally accepted.

# Computable Function

Let $\mathcal{C}$ be the set of all computable functions.

Let $\mathcal{C}_n$ be the set of all $n$-ary computable functions.

# Power of Church-Turing Thesis

**No** one has come up with a computable function that is not in $\mathcal{C}$.

When you are convincing people of your model of computation, you are constructing an effective translation from a well-known computation model to your model.

# Use of Church-Turing Thesis

Church-Turing Thesis allows us to give an informal argument for the computability of a function.

We will make use of a computable function without explicitly defining it.

# Comment on Church-Turing Thesis

CTT and Physical Implementation

- Deterministic Turing Machines are physically implementable. This is the well-known von Neumann Architecture.
- Are quantum computers physically implementable? Can a quantum computer compute more? Can it compute more efficiently?

# Comment on Church-Turing Thesis

CTT and Physical Implementation

- Deterministic Turing Machines are physically implementable. This is the well-known von Neumann Architecture.
- Are quantum computers physically implementable? Can a quantum computer compute more? Can it compute more efficiently?

CTT, is it a Law of Nature or a Wisdom of Human?

# Synopsis

1. Gödel Encoding (section 4.1)
2. URM is Recursive (Appendix of chapter 5)

# Gödel Encoding

*Everything is number!*

# Godel's Insight

The set of syntactical objects of a formal system is denumerable.

# Godel's Insight

The set of syntactical objects of a formal system is denumerable.

More importantly, every syntactical object can be coded up effectively by a number in such a way that a unique syntactical object can be recovered from the number.

# Godel's Insight

The set of syntactical objects of a formal system is denumerable.

More importantly, every syntactical object can be coded up effectively by a number in such a way that a unique syntactical object can be recovered from the number.

This is the crucial technique Gödel used in his proof of the Incompleteness Theorem.

# Enumeration

An enumeration of a set $X$ is a surjection $g : \mathbb{N} \to X$;
this is often represented by writing $\{x_0, x_1, x_2, \ldots\}$.

It is an enumeration without repetition if $g$ is injective.

# Denumeration

A set $X$ is denumerable if there is a bijection $f : X \to \mathbb{N}$.
(denumerate = denote + enumerate)

# Denumeration

A set $X$ is denumerable if there is a bijection $f : X \to \mathbb{N}$.
(denumerate = denote + enumerate)

Let $X$ be a set of "finite objects".

Then $X$ is effectively denumerable if there is a bijection $f : X \to \mathbb{N}$ such that both $f$ and $f^{-1}$ are computable.

# Effective Denumerable Set

**Fact**. $\mathbb{N} \times \mathbb{N}$ is effectively denumerable.

# Effective Denumerable Set

**Fact**. $\mathbb{N} \times \mathbb{N}$ is effectively denumerable.

*Proof*. A bijection $\pi : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is defined by

$$\begin{aligned} \pi(m,n) &\stackrel{\text{def}}{=} 2^m(2n+1) - 1, \\ \pi^{-1}(l) &\stackrel{\text{def}}{=} (\pi_1(l), \pi_2(l)), \end{aligned}$$

where

$$\begin{aligned} \pi_1(x) &\stackrel{\text{def}}{=} (x+1)_1, \\ \pi_2(x) &\stackrel{\text{def}}{=} ((x+1)/2^{\pi_1(x)} - 1)/2. \end{aligned}$$

# Effective Denumerable Set

**Fact**. $\mathbb{N}^+ \times \mathbb{N}^+ \times \mathbb{N}^+$ is effectively denumerable.

# Effective Denumerable Set

**Fact**. $\mathbb{N}^+ \times \mathbb{N}^+ \times \mathbb{N}^+$ is effectively denumerable.

*Proof*. A bijection $\zeta : \mathbb{N}^+ \times \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}$ is defined by

$$\begin{aligned} \zeta(m,n,q) &\overset{\text{def}}{=} \pi(\pi(m-1,n-1),q-1), \\ \zeta^{-1}(l) &\overset{\text{def}}{=} (\pi_1(\pi_1(l))+1, \pi_2(\pi_1(l))+1, \pi_2(l)+1). \end{aligned}$$

# Effective Denumerable Set

**Fact**. $\bigcup_{k>0} \mathbb{N}^k$ is effectively denumerable.

*Proof*. A bijection $\tau : \bigcup_{k>0} \mathbb{N}^k \to \mathbb{N}$ is defined by

$$\begin{aligned}\tau(a_1, \ldots, a_k) \stackrel{\text{def}}{=} \; & 2^{a_1} + 2^{a_1+a_2+1} + 2^{a_1+a_2+a_3+2} + \ldots \\ & + 2^{a_1+a_2+a_3+\ldots,a_k+k-1} - 1.\end{aligned}$$

Now given $x$ it is easy to find $b_1 < b_2 < \ldots < b_k$ such that

$$2^{b_1} + 2^{b_2} + 2^{b_3} + \ldots + 2^{b_k} = x + 1.$$

It is then clear how to calculate $a_1, a_2, a_3, \ldots, a_k$. Details are next.

# Effective Denumerable Set

A number $x \in \mathbb{N}$ has a unique expression as

$$x = \sum_{i=0}^{\infty} \alpha_i 2^i,$$

where $\alpha_i$ is either $0$ or $1$ for all $i \geq 0$.

1. The function $\alpha(i,x) = \alpha_i$ is primitive recursive:

$$\alpha(i,x) = \mathsf{rm}(2, \mathsf{qt}(2^i, x)).$$

2. The function $\ell(x) = \mathit{if}\ x > 0\ \mathit{then}\ k\ \mathit{else}\ 0$ is primitive recursive:

$$\ell(x) = \sum_{i<x} \alpha(i,x).$$

# Effective Denumerable Set

3. If $x > 0$ then it has a unique expression as

$$x = 2^{b_1} + 2^{b_2} + \ldots + 2^{b_k},$$

where $1 \leq k$ and $0 \leq b_1 < b_2 < \ldots < b_k$.

The function $\mathsf{b}(i,x) = \mathit{if}\ (x > 0) \wedge (1 \leq i \leq \ell(x))\ \mathit{then}\ b_i\ \mathit{else}\ 0$ is primitive recursive:

$$\mathsf{b}(i,x) = \begin{cases} \mu y{<}x \left( \sum_{k \leq y} \alpha(k,x) = i \right), & \text{if } (x > 0) \wedge (1 \leq i \leq \ell(x)); \\ 0, & \text{otherwise.} \end{cases}$$

# Effective Denumerable Set

4. If $x > 0$ then it has a unique expression as

$$x = 2^{a_1} + 2^{a_1+a_2+1} + \ldots + 2^{a_l+a_2+\ldots+a_k+k-1}.$$

The function $\mathsf{a}(i,x) = a_i$ is primitive recursive:

$$\begin{aligned} \mathsf{a}(i,x) &= \mathsf{b}(i,x), \text{ if } i = 0 \text{ or } i = 1, \\ \mathsf{a}(i+1,x) &= (\mathsf{b}(i+1,x) \dot{-} \mathsf{b}(i,x)) \dot{-} 1, \text{ if } i \geq 1. \end{aligned}$$

We conclude that $a_1, a_2, a_3, \ldots, a_k$ can be calculated by primitive recursive functions.

# Encoding Program

Let $\mathcal{I}$ be the set of all instructions.

Let $\mathcal{P}$ be the set of all programs.

The objects in $\mathcal{I}$, and $\mathcal{P}$ as well, are “finite objects”.

# Encoding Program

**Theorem**. $\mathcal{I}$ is effectively denumerable.

*Proof*. The bijection $\beta : \mathcal{I} \to \mathbb{N}$ is defined as follows:

$$\begin{aligned}
\beta(Z(n)) &= 4(n-1), \\
\beta(S(n)) &= 4(n-1)+1, \\
\beta(T(m,n)) &= 4\pi(m-1,n-1)+2, \\
\beta(J(m,n,q)) &= 4\zeta(m,n,q)+3.
\end{aligned}$$

The converse $\beta^{-1}$ is easy.

# Encoding Program

**Theorem**. $\mathcal{P}$ is effectively denumerable.

*Proof*. The bijection $\gamma : \mathcal{P} \to \mathbb{N}$ is defined as follows:

$$\gamma(P) \quad = \quad \tau(\beta(I_1), \ldots, \beta(I_s)),$$

assuming $P = I_1, \ldots, I_s$.

The converse $\gamma^{-1}$ is obvious.

# Gödel Number of Program

The value $\gamma(P)$ is called the Gödel number of $P$.

# Gödel Number of Program

The value $\gamma(P)$ is called the Gödel number of $P$.

$$\begin{aligned} P_n &= \text{the programme with Godel index } n \\ &= \gamma^{-1}(n) \end{aligned}$$

# Gödel Number of Program

The value $\gamma(P)$ is called the Gödel number of $P$.

$$\begin{aligned} P_n &= \text{the programme with Godel index } n \\ &= \gamma^{-1}(n) \end{aligned}$$

We shall fix this particular encoding function $\gamma$ throughout.

# Example

Let $P$ be the program $T(1,3), S(4), Z(6)$.

# Example

Let $P$ be the program $T(1,3), S(4), Z(6)$.

$\beta(T(1,3)) = 18$
$\beta(S(4)) = 13$
$\beta(Z(6)) = 20$

# Example

Let $P$ be the program $T(1,3), S(4), Z(6)$.

$\beta(T(1,3)) = 18$
$\beta(S(4)) = 13$
$\beta(Z(6)) = 20$

$\gamma(P) = 2^{18} + 2^{32} + 2^{53} - 1$

# Example

Consider $P_{4127}$.

# Example

Consider $P_{4127}$.

$4127 = 2^5 + 2^{12} - 1$.

# Example

Consider $P_{4127}$.

$4127 = 2^5 + 2^{12} - 1$.

$\beta(I_1) = 4 + 1$
$\beta(I_2) = 4\pi(1,0) + 2$

# Example

Consider $P_{4127}$.

$4127 = 2^5 + 2^{12} - 1$.

$\beta(I_1) = 4 + 1$
$\beta(I_2) = 4\pi(1,0) + 2$

So $P_{4127}$ is $S(2); T(2,1)$.

# URM is Recursive

# Kleene's Proof

Kleene demonstrated how to prove that machine computable functions are recursive functions.

# Proof in Detail

The states of the computation of the program $P_e(\widetilde{x})$ can be described by a configuration and an instruction number.

# Proof in Detail

The states of the computation of the program $P_e(\widetilde{x})$ can be described by a configuration and an instruction number.

A state can be coded up by the number

$$\sigma = \pi(c, j),$$

where $c$ is the configuration that codes up the current values in the registers

$$c = 2^{r_1} 3^{r_2} \ldots = \prod_{i \geq 1} p_i^{r_i},$$

and $j$ is the next instruction number.

# Proof in Detail

To describe the changes of the states of $P_e(\widetilde{x})$, we introduce three $(n+2)$-ary functions:

$$
\begin{aligned}
\mathsf{c}_n(e,\widetilde{x},t) &= \text{the configuration after } t \text{ steps of } P_e(\widetilde{x}),\\
\mathsf{j}_n(e,\widetilde{x},t) &= \text{the number of the next instruction after } t \text{ steps}\\
&\quad \text{of } P_e(\widetilde{x}) \text{ (it is 0 if } P_e(\widetilde{x}) \text{ stops in } t \text{ or less steps)},\\
\sigma_n(e,\widetilde{x},t) &= \pi(\mathsf{c}_n(e,\widetilde{x},t),\mathsf{j}_n(e,\widetilde{x},t)).
\end{aligned}
$$

# Proof in Detail

To describe the changes of the states of $P_e(\widetilde{x})$, we introduce three $(n+2)$-ary functions:

$$
\begin{aligned}
\mathsf{c}_n(e,\widetilde{x},t) &= \text{the configuration after } t \text{ steps of } P_e(\widetilde{x}),\\
\mathsf{j}_n(e,\widetilde{x},t) &= \text{the number of the next instruction after } t \text{ steps}\\
&\quad \text{of } P_e(\widetilde{x}) \text{ (it is 0 if } P_e(\widetilde{x}) \text{ stops in } t \text{ or less steps)},\\
\sigma_n(e,\widetilde{x},t) &= \pi(\mathsf{c}_n(e,\widetilde{x},t),\mathsf{j}_n(e,\widetilde{x},t)).
\end{aligned}
$$

If $\sigma_n$ is primitive recursive, then $\mathsf{c}_n, \mathsf{j}_n$ are primitive recursive!

# Proof in Detail

If the computation of $P_e(\widetilde{x})$ stops, it does so in

$$\mu t(\mathsf{j}_n(e, \widetilde{x}, t) = 0)$$

steps.

# Proof in Detail

If the computation of $P_e(\widetilde{x})$ stops, it does so in

$$\mu t(\mathsf{j}_n(e, \widetilde{x}, t) = 0)$$

steps.

Then the final configuration is

$$\mathsf{c}_n(e, \widetilde{x}, \mu t(\mathsf{j}_n(e, \widetilde{x}, t) = 0)).$$

# Proof in Detail

If the computation of $P_e(\widetilde{x})$ stops, it does so in

$$\mu t(\mathsf{j}_n(e, \widetilde{x}, t) = 0)$$

steps.

Then the final configuration is

$$\mathsf{c}_n(e, \widetilde{x}, \mu t(\mathsf{j}_n(e, \widetilde{x}, t) = 0)).$$

We conclude that the value of the computation $P_e(\widetilde{x})$ is

$$(\mathsf{c}_n(e, \widetilde{x}, \mu t(\mathsf{j}_n(e, \widetilde{x}, t) = 0)))_1.$$

# Proof in Detail

The function $\sigma_n$ can be defined as follows:

$$\begin{aligned} \sigma_n(e,\widetilde{x},0) &= \pi(2^{x_1}3^{x_2}\ldots p_n^{x_n},1), \\ \sigma_n(e,\widetilde{x},t+1) &= \pi(\mathsf{config}(e,\sigma_n(e,\widetilde{x},t)),\mathsf{next}(e,\sigma_n(e,\widetilde{x},t))), \end{aligned}$$

where $\mathsf{config}(e,\pi(c,j))$ is the new configuration, and $\mathsf{next}(e,\pi(c,j))$ is the number of the next instruction, after the $j$-th instruction has been executed upon $c$.

# Proof in Detail

$$\mathsf{ln}(e) = \text{the number of instructions in } P_e;$$

$$\mathsf{gn}(e,j) = \begin{cases} \text{the code of } I_j \text{ in } P_e, & \text{if } 1 \le j \le \mathsf{ln}(e), \\ 0, & \text{otherwise.} \end{cases}$$

$$\mathsf{ch}(c,z) = \text{the resulting configuration when the configuration } c \text{ is operated on by the instruction with code number } z.$$

$$\mathsf{v}(c,j,z) = \begin{cases} \text{the number } j' \text{ of the next instruction when the configuration } c \text{ is operated on by the } j\text{th instruction with code } z, & \text{if } j > 0, \\ 0, & \text{if } j = 0. \end{cases}$$

# Proof in Detail

# Proof in Detail

$$\mathsf{config}(e,\sigma) \quad = \quad \begin{cases} \mathsf{ch}(\pi_1(\sigma), \mathsf{gn}(e, \pi_2(\sigma))), & \text{if } 1 \leq \pi_2(\sigma) \leq \mathsf{ln}(e), \\ \pi_1(\sigma), & \text{otherwise.} \end{cases}$$

# Proof in Detail

$$\mathsf{config}(e,\sigma) \quad = \quad \begin{cases} \mathsf{ch}(\pi_1(\sigma), \mathsf{gn}(e, \pi_2(\sigma))), & \text{if } 1 \leq \pi_2(\sigma) \leq \mathsf{ln}(e), \\ \pi_1(\sigma), & \text{otherwise.} \end{cases}$$

$$\mathsf{next}(e,\sigma) \quad = \quad \begin{cases} \mathsf{v}(\pi_1(\sigma), \pi_2(\sigma), \mathsf{gn}(e, \pi_2(\sigma))), & \text{if } 1 \leq \pi_2(\sigma) \leq \mathsf{ln}(e), \\ 0, & \text{otherwise.} \end{cases}$$

# Proof in Detail ($\mathsf{ln}, \mathsf{gn}$)

$$\mathsf{ln}(e) = \text{the number of instructions in } P_e;$$

$$\mathsf{gn}(e,j) = \begin{cases} \text{the code of } I_j \text{ in } P_e, & \text{if } 1 \leq j \leq \mathsf{ln}(e), \\ 0, & \text{otherwise.} \end{cases}$$

Both functions are primitive recursive since

$$\begin{aligned} \mathsf{ln}(e) &= \ell(e+1), \\ \mathsf{gn}(e,j) &= \mathsf{a}(j, e+1). \end{aligned}$$

# Proof in Detail (ch)

The following function

$$\mathsf{ch}(c,z) \quad = \quad \text{the resulting configuration when the configuration } c \text{ is operated on by the instruction with code number } z.$$

is primitive recursive if

$$\mathsf{ch}(c,z) \quad = \quad \begin{cases} \mathsf{zero}(c,\mathsf{u}(z)), & \text{if } \mathsf{rm}(4,z) = 0, \\ \mathsf{succ}(c,\mathsf{u}(z)), & \text{if } \mathsf{rm}(4,z) = 1, \\ \mathsf{tran}(c,\mathsf{u}_1(z),\mathsf{u}_2(z)), & \text{if } \mathsf{rm}(4,z) = 2, \\ c, & \text{if } \mathsf{rm}(4,z) = 3. \end{cases}$$

# Proof in Detail (ch)

$\mathsf{u}(z) = m$ whenever $z = \beta(Z(m))$ or $z = \beta(S(m))$:

$$\mathsf{u}(z) = \mathsf{qt}(4, z) + 1.$$

# Proof in Detail (ch)

$\mathsf{u}(z) = m$ whenever $z = \beta(Z(m))$ or $z = \beta(S(m))$:

$$\mathsf{u}(z) = \mathsf{qt}(4, z) + 1.$$

$\mathsf{u}_1(z) = m_1$ and $\mathsf{u}_2(z) = m_2$ whenever $z = \beta(T(m_1, m_2))$:

$$\begin{aligned} \mathsf{u}_1(z) &= \pi_1(\mathsf{qt}(4, z)) + 1, \\ \mathsf{u}_2(z) &= \pi_2(\mathsf{qt}(4, z)) + 1. \end{aligned}$$

# Proof in Detail (ch)

The change in the configuration $c$ effected by instruction $Z(m)$:

$$\mathsf{zero}(c, m) = \mathsf{qt}(p_m^{(c)_m}, c).$$

# Proof in Detail (ch)

The change in the configuration $c$ effected by instruction $Z(m)$:

$$\mathsf{zero}(c,m) = \mathsf{qt}(p_m^{(c)_m}, c).$$

The change in the configuration $c$ effected by instruction $S(m)$:

$$\mathsf{succ}(c,m) = p_m c.$$

# Proof in Detail (ch)

The change in the configuration $c$ effected by instruction $Z(m)$:

$$\mathsf{zero}(c,m) = \mathsf{qt}(p_m^{(c)_m}, c).$$

The change in the configuration $c$ effected by instruction $S(m)$:

$$\mathsf{succ}(c,m) = p_m c.$$

The change in the configuration $c$ effected by instruction $T(m,n)$:

$$\mathsf{tran}(c,m,n) = \mathsf{qt}(p_n^{(c)_n}, p_n^{(c)_m} c).$$

# Proof in Detail ($\mathsf{v}$)

The following function

$$\mathsf{v}(c,j,z) \quad = \quad \begin{cases} \text{the number } j' \text{ of the next instruction} & \\ \text{when the configuration } c \text{ is operated} & \text{if } j > 0, \\ \text{on by the } j\text{th instruction with code } z, & \\ 0, & \text{if } j = 0. \end{cases}$$

is primitive recursive if

$$\mathsf{v}(c,j,z) \quad = \quad \begin{cases} j+1, & \text{if } \mathsf{rm}(4,z) \neq 3, \\ j+1, & \text{if } \mathsf{rm}(4,z) = 3 \ \wedge \ (c)_{\mathsf{v}_1(z)} \neq (c)_{\mathsf{v}_2(z)}, \\ \mathsf{v}_3(z), & \text{if } \mathsf{rm}(4,z) = 3 \ \wedge \ (c)_{\mathsf{v}_1(z)} = (c)_{\mathsf{v}_2(z)}. \end{cases}$$

# Proof in Detail ($\mathsf{v}$)

$\mathsf{v}_1(z) = m_1$ and $\mathsf{v}_2(z) = m_2$ and $\mathsf{v}_3(z) = q$ if $z = \beta(J(m_1, m_2, q))$:

$$\begin{aligned}
\mathsf{v}_1(z) &= \pi_1(\pi_1(\mathsf{qt}(4, z))) + 1, \\
\mathsf{v}_2(z) &= \pi_2(\pi_1(\mathsf{qt}(4, z))) + 1, \\
\mathsf{v}_3(z) &= \pi_2(\mathsf{qt}(4, z)) + 1.
\end{aligned}$$

# Proof in Detail

We can now define the function $\mathsf{config}(\_,\_)$ by

$$\mathsf{config}(e,\sigma) \quad = \quad \begin{cases} \mathsf{ch}(\pi_1(\sigma), \mathsf{gn}(e, \pi_2(\sigma))), & \text{if } 1 \leq \pi_2(\sigma) \leq \mathsf{ln}(e), \\ \pi_1(\sigma), & \text{otherwise.} \end{cases}$$

and the function $\mathsf{next}(\_,\_)$ by

$$\mathsf{next}(e,\sigma) \quad = \quad \begin{cases} \mathsf{v}(\pi_1(\sigma), \pi_2(\sigma), \mathsf{gn}(e, \pi_2(\sigma))), & \text{if } 1 \leq \pi_2(\sigma) \leq \mathsf{ln}(e), \\ 0, & \text{otherwise.} \end{cases}$$

# Proof in Detail

We conclude that the functions $\mathsf{c}_n, \mathsf{j}_n, \sigma_n$ are primitive recursive.

# Computability Theory VII

## S-M-N Theorem

Guoqiang Li

Shanghai Jiao Tong University

Nov. 14, 2014

# Problem Index

# Motivation

By Church-Turing Thesis one may study computability theory using any of the computation models.

It is much more instructive however to carry out the study in a model independent manner.

The first step is to assign index to computable function.

# Review Tips

# Effective Denumerable Set

$\mathbb{N} \times \mathbb{N}$

$\mathbb{N}^+ \times \mathbb{N}^+ \times \mathbb{N}^+$

$\bigcup_{k>0} \mathbb{N}^k$

# Effective Denumerable Set

$\mathbb{N} \times \mathbb{N}$

$\mathbb{N}^+ \times \mathbb{N}^+ \times \mathbb{N}^+$

$\bigcup_{k>0} \mathbb{N}^k$

$\mathcal{I}$ is effectively denumerable.

$\mathcal{P}$ is effectively denumerable.

$\gamma(P) = \tau(\beta(I_1), \ldots, \beta(I_s))$

The value $\gamma(P)$ is called the Gödel number of $P$.

# Synopsis

1. Gödel Index
2. S-m-n Theorem

Gödel Index

# Basic Idea

We see a number as an index for a problem/function if it is the Gödel number of a programme that solves/calculates the problem/function.

# Definition

Suppose $a \in \mathbb{N}$ and $n \geq 1$.

$$
\begin{aligned}
\phi_a^{(n)} &= \text{the } n \text{ ary function computed by } P_a \\
&= f_{P_a}^{(n)}, \\
W_a^{(n)} &= \text{the domain of } \phi_a^{(n)} = \{(x_1, \ldots, x_n) \mid P_a(x_1, \ldots, x_n) \downarrow\}, \\
E_a^{(n)} &= \text{the range of } \phi_a^{(n)}.
\end{aligned}
$$

The super script $(n)$ is omitted when $n = 1$.

# Example

Let $a = 4127$. Then $P_{4127} = S(2); T(2,1)$.

# Example

Let $a = 4127$. Then $P_{4127} = S(2); T(2,1)$.

If the program is seen to calculate a unary function, then

$$\begin{aligned} \phi_{4127}(x) &= 1, \\ W_{4127} &= \mathbb{N}, \\ E_{4127} &= \{1\}. \end{aligned}$$

# Example

Let $a = 4127$. Then $P_{4127} = S(2); T(2,1)$.

If the program is seen to calculate a unary function, then

$$\begin{aligned} \phi_{4127}(x) &= 1, \\ W_{4127} &= \mathbb{N}, \\ E_{4127} &= \{1\}. \end{aligned}$$

If the program is seen to calculate an $n$-ary function, then

$$\begin{aligned} \phi^{(n)}_{4127}(x_1, \ldots, x_n) &= x_2 + 1, \\ W^n_{4127} &= \mathbb{N}^n, \\ E^n_{4127} &= \mathbb{N}^+. \end{aligned}$$

# Gödel Index for Computable Function

Suppose $f$ is an $n$-ary computable function..

A number $a$ is an index for $f$ if $f = \phi_a^{(n)}$.

# Padding Lemma

**Padding Lemma**

Every computable function has infinite indices. Moreover for each $x$ we can effectively find an infinite recursive set $A_x$ of indices for $\phi_x$.

# Padding Lemma

## Padding Lemma

Every computable function has infinite indices. Moreover for each $x$ we can effectively find an infinite recursive set $A_x$ of indices for $\phi_x$.

## Proof

Systematically add useless instructions to $P_x$.

# Enumeration of Computable Function

**Proposition**

$\mathcal{C}_n$, and $\mathcal{C}$ as well, is denumerable.

# Enumeration of Computable Function

**Proposition**

$\mathcal{C}_n$, and $\mathcal{C}$ as well, is denumerable.

We may list for example all the elements of $\mathcal{C}_n$ as $\phi_0^{(n)}, \phi_1^{(n)}, \phi_2^{(n)}, \ldots$.

# Diagonal Method

**Fact**

There is a total unary function that is not computable.

# Diagonal Method

**Fact**

There is a total unary function that is not computable.

**Proof**

Suppose $\phi_0, \phi_1, \phi_2, \ldots$ is an enumeration of $\mathcal{C}$. Define

$$f(n) = \begin{cases} \phi_n(n) + 1, & \text{if } \phi_n(n) \text{ is defined,} \\ 0, & \text{if } \phi_n(n) \text{ is undefined.} \end{cases}$$

By Church-Turing Thesis the function $f(n)$ is not computable.

# Diagonal Method

**Fact**

There is a total unary function that is not computable.

**Proof**

Suppose $\phi_0, \phi_1, \phi_2, \ldots$ is an enumeration of $\mathcal{C}$. Define

$$f(n) = \begin{cases} \phi_n(n) + 1, & \text{if } \phi_n(n) \text{ is defined,} \\ 0, & \text{if } \phi_n(n) \text{ is undefined.} \end{cases}$$

By Church-Turing Thesis the function $f(n)$ is not computable.

Is the following function computable?

$$f(n) \simeq \begin{cases} \phi_n(n) + 1, & \text{if } \phi_n(n) \text{ is defined,} \\ \uparrow, & \text{if } \phi_n(n) \text{ is undefined.} \end{cases}$$

# Diagonal Method

Suppose there is a sequence $f_0, f_1, \ldots, f_n, \ldots$

Diagonalize out of $f_0, f_1, \ldots$ by making $f$ differ from $f_n$ at $n$.

# S-m-n Theorem

# Motivation

*How do different indexing systems relate?*

# S-m-n Theorem, the Unary Case

Given a binary function $f(x, y)$, we get a unary computable function $f(a, y)$ by fixing a value $a$ for $x$.

# S-m-n Theorem, the Unary Case

Given a binary function $f(x,y)$, we get a unary computable function $f(a,y)$ by fixing a value $a$ for $x$.

Let $e$ be an index for $f(a,y)$. Then

$$f(a,y) \simeq \phi_e(y)$$

# S-m-n Theorem, the Unary Case

Given a binary function $f(x, y)$, we get a unary computable function $f(a, y)$ by fixing a value $a$ for $x$.

Let $e$ be an index for $f(a, y)$. Then

$$f(a, y) \simeq \phi_e(y)$$

S-m-n Theorem states that the index $e$ can be computed from $a$.

# S-m-n Theorem, the Unary Case

**Fact**

Suppose that $f(x,y)$ is a computable function. There is a primitive recursive function $k(x)$ such that

$$f(x,y) \simeq \phi_{k(x)}(y).$$

# S-m-n Theorem, the Unary Case

Proof

# S-m-n Theorem, the Unary Case

Proof

Let $F$ be a program that computes $f$. Consider the following program

$$
\begin{array}{l}
T(1,2) \\
Z(1) \\
\left.\begin{array}{l} S(1) \\ \vdots \\ S(1) \end{array}\right\} a \text{ times} \\
F
\end{array}
$$

# S-m-n Theorem, the Unary Case

Proof

Let $F$ be a program that computes $f$. Consider the following program

$$
\begin{array}{l}
T(1,2) \\
Z(1) \\
\left.\begin{array}{l} S(1) \\ \vdots \\ S(1) \end{array}\right\} a \text{ times} \\
F
\end{array}
$$

The above program can be effectively constructed from $a$.

# S-m-n Theorem, the Unary Case

Proof

Let $F$ be a program that computes $f$. Consider the following program

$$
\begin{array}{l}
T(1,2) \\
Z(1) \\
\left.\begin{array}{l} S(1) \\ \vdots \\ S(1) \end{array}\right\} a \text{ times} \\
F
\end{array}
$$

The above program can be effectively constructed from $a$.

Let $k(a)$ be the Gödel number of the above program. It can be effectively computed from the above program.

# Examples

Let $f(x, y) = y^x$.

# Examples

Let $f(x,y) = y^x$.

Then $\phi_{k(x)}(y) = y^x$. For each fixed $n$, $k(n)$ is an index for $y^n$.

# Examples

Let $f(x,y) = y^x$.

Then $\phi_{k(x)}(y) = y^x$. For each fixed $n$, $k(n)$ is an index for $y^n$.

Let $f(x,y) \simeq \begin{cases} y, & \text{if } y \text{ is a multiple of } x, \\ \uparrow, & \text{otherwise.} \end{cases}$.

# Examples

Let $f(x,y) = y^x$.

Then $\phi_{k(x)}(y) = y^x$. For each fixed $n$, $k(n)$ is an index for $y^n$.

Let $f(x,y) \simeq \begin{cases} y, & \text{if } y \text{ is a multiple of } x, \\ \uparrow, & \text{otherwise.} \end{cases}$.

Then $\phi_{k(n)}(y)$ is defined if and only if $y$ is a multiple of $n$.

# S-m-n Theorem

## S-m-n Theorem

For $m, n$, there is an injective primitive recursive $(m+1)$-function $s_n^m(x, \widetilde{x})$ such that for all $e$ the following holds:

$$\phi_e^{(m+n)}(\widetilde{x}, \widetilde{y}) \simeq \phi_{s_n^m(e,\widetilde{x})}^{(n)}(\widetilde{y})$$

# S-m-n Theorem

**S-m-n Theorem**

For $m, n$, there is an injective primitive recursive $(m+1)$-function $s_n^m(x, \widetilde{x})$ such that for all $e$ the following holds:

$$\phi_e^{(m+n)}(\widetilde{x}, \widetilde{y}) \simeq \phi^{(n)}_{s_n^m(e,\widetilde{x})}(\widetilde{y})$$

S-m-n Theorem is also called **Parameter Theorem**.

# S-m-n Theorem

Proof

Given $e, x_1, \ldots, x_m$, we can effectively construct the following program and its index

$$
\begin{array}{l}
T(n, m+n) \\
\vdots \\
T(1, m+1) \\
Q(1, x_1) \\
\vdots \\
Q(m, x_m) \\
P_e
\end{array}
$$

where $Q(i, x)$ is the program $Z(i), \underbrace{S(i), \ldots, S(i)}_{x \text{ times}}$.

# S-m-n Theorem

Proof

Given $e, x_1, \ldots, x_m$, we can effectively construct the following program and its index

$$
\begin{aligned}
&T(n, m+n)\\
&\vdots\\
&T(1, m+1)\\
&Q(1, x_1)\\
&\vdots\\
&Q(m, x_m)\\
&P_e
\end{aligned}
$$

where $Q(i, x)$ is the program $Z(i), \underbrace{S(i), \ldots, S(i)}_{x \text{ times}}$.

The injectivity is achieved by padding enough useless instructions.

# Exercise I

Show that there is a total computable function $k$ such that for each $n$, $k(n)$ is an index of the function $\lfloor \sqrt[n]{x} \rfloor$.

# Exercise II

Show that there is a total computable function $k$ such that for each $n$, $W_{k(n)} =$ the set of perfect $n$th power.

# Exercise III

Show that there is a total computable function $k$ such that

$$W^{(m)}_{k(n)} = \{(y_1, \ldots, y_m) : y_1 + y_2 + \ldots + y_m = n\}$$

suppose $m \geq 1$.

# Computability Theory VIII

## Universal Program

Guoqiang Li

Shanghai Jiao Tong University

Nov. 28, 2014

# Enumeration Theorem

# General Remark

There are universal programs that embody all the programs.

A program is universal if upon receiving the Gödel number of a program it simulates the program indexed by the number.

# Intuition

Consider the function $\psi(x, y)$ defined as follows

$$\psi(x, y) \simeq \phi_x(y)$$

In an obvious sense $\psi(x, \_)$ is a universal function for the unary functions

$$\phi_0, \phi_1, \phi_2, \phi_3, \ldots.$$

# Universal Function

The universal function for $n$-ary computable functions is the $(n+1)$-ary function $\psi_U^{(n)}$ defined by

$$\psi_U^{(n)}(e, x_1, \ldots, x_n) \simeq \phi_e^{(n)}(x_1, \ldots, x_n).$$

We write $\psi_U$ for $\psi_U^{(1)}$.

# Universal Function

The universal function for $n$-ary computable functions is the $(n+1)$-ary function $\psi_U^{(n)}$ defined by

$$\psi_U^{(n)}(e, x_1, \ldots, x_n) \simeq \phi_e^{(n)}(x_1, \ldots, x_n).$$

We write $\psi_U$ for $\psi_U^{(1)}$.

Question: Is $\psi_U^{(n)}$ computable?

# Enumeration Theorem

**Enumeration Theorem**

For each $n$, the universal function $\psi_U^{(n)}$ is computable.

# Enumeration Theorem

## Enumeration Theorem

For each $n$, the universal function $\psi_U^{(n)}$ is computable.

## Proof

Given a number $e$, decode the number to get the program $P_e$; and then simulate the program $P_e$. If the simulation ever terminates, then return the number in $R_1$. By Church-Turing Thesis, $\psi_U^{(n)}$ is computable.

# Undecidability

**Proposition**

The problem '$\phi_x$ is total' is undecidable.

# Undecidability

**Proposition**

The problem '$\phi_x$ is total' is undecidable.

Proof

If '$\phi_x$ is total' were decidable, then by Church-Turing Thesis

$$f(x) = \begin{cases} \psi_U(x,x) + 1, & \text{if } \phi_x \text{ is total,} \\ 0, & \text{if } \phi_x \text{ is not total.} \end{cases}$$

would be a total computable function that differs from every total computable function.

# Effectiveness of Function Operation

**Proposition**

There is a total computable function $s(x, y)$ such that $\phi_{s(x,y)} = \phi_x \phi_y$ for all $x, y$.

# Effectiveness of Function Operation

**Proposition**

There is a total computable function $s(x,y)$ such that $\phi_{s(x,y)} = \phi_x \phi_y$ for all $x, y$.

Proof

Let $f(x,y,z) \simeq \phi_x(z)\phi_y(z) \simeq \psi_U(x,z)\psi_U(y,z)$.

By S-m-n Theorem there is a total function $s(x,y)$ such that $\phi_{s(x,y)}(z) \simeq f(x,y,z)$.

# Effectiveness of Set Operation

**Proposition**

There is a total computable function $s(x, y)$ such that $W_{s(x,y)} = W_x \cup W_y$.

# Effectiveness of Set Operation

**Proposition**

There is a total computable function $s(x,y)$ such that $W_{s(x,y)} = W_x \cup W_y$.

Proof

Let

$$f(x,y,z) = \begin{cases} 1, & \text{if } z \in W_x \text{ or } z \in W_y, \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

By S-m-n Theorem there is a total function $s(x,y)$ such that $\phi_{s(x,y)}(z) \simeq f(x,y,z)$. Clearly $W_{s(x,y)} = W_x \cup W_y$.

# Effectiveness of Recursion

Consider $f$ defined by the following recursion

$$f(e_1, e_2, \widetilde{x}, 0) \simeq \phi_{e_1}^{(n)}(\widetilde{x}) \simeq \psi_U^{(n)}(e_1, \widetilde{x})$$

and

$$\begin{aligned} f(e_1, e_2, \widetilde{x}, y+1) &\simeq \phi_{e_2}^{(n+2)}(\widetilde{x}, y, f(e_1, e_2, \widetilde{x}, y)) \\ &\simeq \psi_U^{(n+2)}(e_2, \widetilde{x}, y, f(e_1, e_2, \widetilde{x}, y)). \end{aligned}$$

By S-m-n Theorem, there is a total computable function $r(e_1, e_2)$ such that

$$\phi_{r(e_1,e_2)}^{(n+1)}(\widetilde{x}, y) \simeq f(e_1, e_2, \widetilde{x}, y)$$

# Non-Primitive Recursive Total Function

**Theorem**

There is a total computable function that is not primitive recursive.

# Non-Primitive Recursive Total Function

**Theorem**

There is a total computable function that is not primitive recursive.

Proof

1. The primitive recursive functions have a universal function.
2. Such a function cannot be primitive recursive by diagonalisation.

# Recursion Theorem

# Recursion Theorem

## Recursion Theorem

Let $f$ be a total unary computable function. Then there is a number $n$ such that $\phi_{f(n)} = \phi_n$.

### Proof

By S-m-n Theorem there is an injective primitive recursive function $s(x)$ such that for all $x$

$$\phi_{s(x)}(y) \simeq \begin{cases} \phi_{\phi_x(x)}(y), & \text{if } \phi_x(x)\downarrow; \\ \uparrow, & \text{otherwise.} \end{cases}$$

Let $v$ be such that $\phi_v = s \circ f$. Obviously $\phi_v$ is total and $\phi_v(v)\downarrow$.

$$\phi_{s(v)} = \phi_{\phi_v(v)} = \phi_{f(s(v))}$$

We are done by letting $n$ be $s(v)$.

# Exercise I

Show that there is a total computable function $k$ such that for each $n$, $E_{k(n)} = W_n$.

# Exercise II

Show that there is a total computable function $k(x, y)$ such that for each $x, y$, $E_{k(x,y)} = E_x \cup E_y$.

# Exercise III

Suppose $f(n)$ is computable, show that there is a total computable function $k(n)$ such that for each $n$, $W_{k(n)} = f^{-1}(W_n)$.

# Computability Theory IX

## Undecidability

Guoqiang Li

Shanghai Jiao Tong University

Nov. 28, 2014

*Assignment 3 was announced!*

*The deadline is Dec. 12.*

# Undecidability

# Decidability and Undecidability

A predicate $M(\mathbf{x})$ is decidable if its characteristic function $c_M(\mathbf{x})$ given by

$$c_M(\mathbf{x}) = \begin{cases} 1, & \text{if } M(\mathbf{x}) \text{ holds,} \\ 0, & \text{if } M(\mathbf{x}) \text{ does not hold.} \end{cases}$$

is computable.

# Decidability and Undecidability

A predicate $M(\mathbf{x})$ is decidable if its characteristic function $c_M(\mathbf{x})$ given by

$$c_M(\mathbf{x}) = \begin{cases} 1, & \text{if } M(\mathbf{x}) \text{ holds,} \\ 0, & \text{if } M(\mathbf{x}) \text{ does not hold.} \end{cases}$$

is computable.

The predicate $M(\mathbf{x})$ is undecidable if it is not decidable.

# Undecidability Result

**Theorem**

The problem '$x \in W_x$' is undecidable.

# Undecidability Result

**Theorem**

The problem '$x \in W_x$' is undecidable.

*Proof*

The characteristic function of this problem is given by

$$c(x) = \begin{cases} 1, & \text{if } x \in W_x, \\ 0, & \text{if } x \notin W_x. \end{cases}$$

# Undecidability Result

**Theorem**

The problem '$x \in W_x$' is undecidable.

*Proof*

The characteristic function of this problem is given by

$$c(x) = \begin{cases} 1, & \text{if } x \in W_x, \\ 0, & \text{if } x \notin W_x. \end{cases}$$

Suppose $c(x)$ was computable. Then the function $g(x)$ defined below would also be computable.

$$g(x) = \begin{cases} 0, & \text{if } c(x) = 0, \\ \text{undefined}, & \text{if } c(x) = 1. \end{cases}$$

# Undecidability Result

**Theorem**

The problem '$x \in W_x$' is undecidable.

*Proof*

The characteristic function of this problem is given by

$$c(x) = \begin{cases} 1, & \text{if } x \in W_x, \\ 0, & \text{if } x \notin W_x. \end{cases}$$

Suppose $c(x)$ was computable. Then the function $g(x)$ defined below would also be computable.

$$g(x) = \begin{cases} 0, & \text{if } c(x) = 0, \\ \text{undefined}, & \text{if } c(x) = 1. \end{cases}$$

Let $m$ be an index for $g$. Then

$$m \in W_m \text{ iff } c(m) = 0 \text{ iff } m \notin W_m.$$

# Undecidability Result

**Corollary**

There is a computable function $h$ such that both '$x \in Dom(h)$' and '$x \in Ran(h)$' are undecidable.

# Undecidability Result

**Corollary**

There is a computable function $h$ such that both '$x \in Dom(h)$' and '$x \in Ran(h)$' are undecidable.

*Proof*

Let

$$h(x) = \begin{cases} x, & \text{if } x \in W_x, \\ \text{undefined}, & \text{if } x \notin W_x. \end{cases}$$

# Undecidability Result

**Corollary**

There is a computable function $h$ such that both '$x \in Dom(h)$' and '$x \in Ran(h)$' are undecidable.

*Proof*

Let

$$h(x) = \begin{cases} x, & \text{if } x \in W_x, \\ \text{undefined}, & \text{if } x \notin W_x. \end{cases}$$

Clearly $x \in Dom(h)$ iff $x \in W_x$ iff $x \in Ran(h)$.

# Undecidability Result

**Theorem**

The problem '$\phi_x(y)$ is defined' is undecidable.

# Undecidability Result

**Theorem**

The problem '$\phi_x(y)$ is defined' is undecidable.

*Proof*

If $y \in W_x$ were decidable then $x \in W_x$ would be decidable.

# Undecidability Result

**Theorem**

The problem '$\phi_x(y)$ is defined' is undecidable.

*Proof*

If $y \in W_x$ were decidable then $x \in W_x$ would be decidable.

This is known as Halting Problem.

# Undecidability Result

**Theorem**

The problem '$\phi_x(y)$ is defined' is undecidable.

*Proof*

If $y \in W_x$ were decidable then $x \in W_x$ would be decidable.

This is known as Halting Problem.

In this proof we have reduced the problem '$x \in W_x$' to the problem '$y \in W_x$'. The reduction shows that the latter is at least as hard as the former.

# Undecidability Result

**Theorem**

The problem '$\phi_x = \mathbf{0}$' is undecidable.

# Undecidability Result

**Theorem**

The problem '$\phi_x = \mathbf{0}$' is undecidable.

*Proof*

Consider the function $f$ defined by

$$f(x,y) = \begin{cases} 0, & \text{if } x \in W_x, \\ \text{undefined}, & \text{if } x \notin W_x. \end{cases}$$

# Undecidability Result

**Theorem**

The problem '$\phi_x = \mathbf{0}$' is undecidable.

*Proof*

Consider the function $f$ defined by

$$f(x,y) = \begin{cases} 0, & \text{if } x \in W_x, \\ \text{undefined}, & \text{if } x \notin W_x. \end{cases}$$

By s-m-n theorem there is some total computable function $k(x)$ such that $\phi_{k(x)}(y) \simeq f(x,y)$.

# Undecidability Result

**Theorem**

The problem '$\phi_x = \mathbf{0}$' is undecidable.

*Proof*

Consider the function $f$ defined by

$$f(x,y) = \begin{cases} 0, & \text{if } x \in W_x, \\ \text{undefined}, & \text{if } x \notin W_x. \end{cases}$$

By s-m-n theorem there is some total computable function $k(x)$ such that $\phi_{k(x)}(y) \simeq f(x,y)$.

It is clear that $\phi_{k(x)} = \mathbf{0}$ iff $x \in W_x$.

# Undecidability Result

**Corollary**

The problem '$\phi_x \simeq \phi_y$' is undecidable.

# Undecidability Result

**Theorem**

Let $c$ be any number. The followings are undecidable.

(a) Acceptance Problem: '$c \in W_x$',

(b) Printing Problem: '$c \in E_x$'.

# Undecidability Result

**Theorem**

Let $c$ be any number. The followings are undecidable.

(a) Acceptance Problem: '$c \in W_x$',

(b) Printing Problem: '$c \in E_x$'.

*Proof*

Consider the function $f$ defined by

$$f(x,y) = \begin{cases} y, & \text{if } x \in W_x, \\ \text{undefined}, & \text{if } x \notin W_x. \end{cases}$$

# Undecidability Result

**Theorem**

Let $c$ be any number. The followings are undecidable.

(a) Acceptance Problem: '$c \in W_x$',

(b) Printing Problem: '$c \in E_x$'.

*Proof*

Consider the function $f$ defined by

$$f(x,y) = \begin{cases} y, & \text{if } x \in W_x, \\ \text{undefined}, & \text{if } x \notin W_x. \end{cases}$$

By s-m-n theorem there is some total computable function $k(x)$ such that $\phi_{k(x)}(y) \simeq f(x,y)$.

# Undecidability Result

**Theorem**

Let $c$ be any number. The followings are undecidable.
(a) Acceptance Problem: '$c \in W_x$',
(b) Printing Problem: '$c \in E_x$'.

*Proof*

Consider the function $f$ defined by

$$f(x,y) = \begin{cases} y, & \text{if } x \in W_x, \\ \text{undefined}, & \text{if } x \notin W_x. \end{cases}$$

By s-m-n theorem there is some total computable function $k(x)$ such that $\phi_{k(x)}(y) \simeq f(x,y)$.

It is clear that $c \in W_{k(x)}$ iff $x \in W_x$ iff $c \in E_{k(x)}$.

# More on Undecidability

# Exercise I

$x \in E_x$

# Exercise II

$W_x = W_y$

# Exercise III

$\phi_x(y) = 0$

# Exercise IV

$E_x$ is infinite.

# Exercise V

‘$\phi_x = g$’, where $g$ is any fixed computable function.

# Computability Theory XI

## Recursively Enumerable Set

Guoqiang Li

Shanghai Jiao Tong University

Dec. 12&19, 2013

# Assignment

*Assignment 4 was announced!*

*The deadline is Dec. 26!*

# An Exercise

Let $A, B \subseteq \mathbb{N}$. Define sets of $A \oplus B$ and $A \otimes B$ by

$$\begin{aligned} A \oplus B &= \{2x \mid x \in A\} \cup \{2x+1 \mid x \in B\} \\ A \otimes B &= \{\pi(x,y) \mid x \in A \wedge y \in B\} \end{aligned}$$

# An Exercise

Let $A, B \subseteq \mathbb{N}$. Define sets of $A \oplus B$ and $A \otimes B$ by

$$\begin{aligned} A \oplus B &= \{2x \mid x \in A\} \cup \{2x+1 \mid x \in B\} \\ A \otimes B &= \{\pi(x,y) \mid x \in A \wedge y \in B\} \end{aligned}$$

1. $A \oplus B$ is recursive iff $A$ and $B$ are both recursive.

# An Exercise

Let $A, B \subseteq \mathbb{N}$. Define sets of $A \oplus B$ and $A \otimes B$ by

$$\begin{aligned} A \oplus B &= \{2x \mid x \in A\} \cup \{2x+1 \mid x \in B\} \\ A \otimes B &= \{\pi(x,y) \mid x \in A \wedge y \in B\} \end{aligned}$$

1. $A \oplus B$ is recursive iff $A$ and $B$ are both recursive.
2. if $A, B \neq \emptyset$, then $A \otimes B$ is recursive iff $A$ and $B$ are both recursive.

We have seen that many sets, although not recursive, can be effectively generated in the sense that, for any such set, there is an effective procedure that produces the elements of the set in a non-stop manner.

We shall formalize this intuition in this lecture.

# Synopsis

1. Recursively Enumerable Set
2. Characterization of R.E. Set
3. Rice-Shapiro Theorem

# Recursively Enumerable Set

# The Definition of R.E. Set

The partial characteristic function of a set $A$ is given by

$$\chi_A(x) = \begin{cases} 1, & \text{if } x \in A, \\ \uparrow, & \text{if } x \notin A. \end{cases}$$

$A$ is recursively enumerable if $\chi_A$ is computable.

We shall often abbreviate 'recursively enumerable set' to 'r.e. set'.

# Partially Decidable Problem

A problem $f : \mathbb{N} \to \{0, 1\}$ is partially decidable if $dom(f)$ is r.e.

# Partially Decidable Predicate

A predicate $M(\widetilde{x})$ of natural number is partially decidable if its partial characteristic function

$$\chi_M(\widetilde{x}) = \begin{cases} 1, & \text{if } M(\widetilde{x}) \text{ holds,} \\ \uparrow, & \text{if } M(\widetilde{x}) \text{ does not hold,} \end{cases}$$

is computable.

Partially Decidable Problem $\Leftrightarrow$ Partially Decidable Predicate

$\Leftrightarrow$ Recursively Enumerable Set

# Example

The halting problem is partially decidable. Its partial characteristic function is given by

$$\chi_H(x,y) = \begin{cases} 1, & \text{if } P_x(y)\downarrow, \\ \uparrow, & \text{otherwise.} \end{cases}$$

# Example

The halting problem is partially decidable. Its partial characteristic function is given by

$$\chi_H(x,y) = \begin{cases} 1, & \text{if } P_x(y)\downarrow, \\ \uparrow, & \text{otherwise.} \end{cases}$$

$K, K_0, K_1$ are r.e..

# Example

The halting problem is partially decidable. Its partial characteristic function is given by

$$\chi_H(x,y) = \begin{cases} 1, & \text{if } P_x(y)\downarrow, \\ \uparrow, & \text{otherwise.} \end{cases}$$

$K, K_0, K_1$ are r.e.. But none of $\overline{K}, \overline{K_0}, \overline{K_1}$ is r.e..

# Index for Recursively Enumerable Set

A set is r.e. iff it is the domain of a unary computable function.

# Index for Recursively Enumerable Set

A set is r.e. iff it is the domain of a unary computable function.

So $W_0, W_1, W_2, \ldots$ is an enumeration of all r.e. sets.

# Index for Recursively Enumerable Set

A set is r.e. iff it is the domain of a unary computable function.

So $W_0, W_1, W_2, \ldots$ is an enumeration of all r.e. sets.

Every r.e. set has an infinite number of indexes.

# Closure Property

**Union Theorem**. The recursively enumerable sets are closed under union and intersection uniformly and effectively.

# Closure Property

**Union Theorem**. The recursively enumerable sets are closed under union and intersection uniformly and effectively.

*Proof*. According to S-m-n Theorem there are primitive recursive functions $r(x,y), s(x,y)$ such that

$$\begin{aligned} W_{r(x,y)} &= W_x \cup W_y, \\ W_{s(x,y)} &= W_x \cap W_y. \end{aligned}$$

# The Most Hard Recursively Enumerable Set

**Fact**. If $A \leq_m B$ and $B$ is r.e. then $A$ is r.e..

# The Most Hard Recursively Enumerable Set

**Fact**. If $A \leq_m B$ and $B$ is r.e. then $A$ is r.e..

**Theorem**. $A$ is r.e. iff $A \leq_1 K$.

*Proof*. Suppose $A$ is r.e. Let $f(x,y)$ be defined by

$$f(x,y) = \begin{cases} 1, & \text{if } x \in A, \\ \uparrow, & \text{if } x \notin A. \end{cases}$$

By S-m-n Theorem there is an injective primitive recursive function $s(x)$ s.t. $f(x,y) = \phi_{s(x)}(y)$. It is clear that $x \in A$ iff $s(x) \in K$.

# The Most Hard Recursively Enumerable Set

**Fact**. If $A \leq_m B$ and $B$ is r.e. then $A$ is r.e..

**Theorem**. $A$ is r.e. iff $A \leq_1 K$.

*Proof*. Suppose $A$ is r.e. Let $f(x,y)$ be defined by

$$f(x,y) = \begin{cases} 1, & \text{if } x \in A, \\ \uparrow, & \text{if } x \notin A. \end{cases}$$

By S-m-n Theorem there is an injective primitive recursive function $s(x)$ s.t. $f(x,y) = \phi_{s(x)}(y)$. It is clear that $x \in A$ iff $s(x) \in K$.

**Comment**. No r.e. set is more difficult than $K$.

# Characterization of R.E. Set

# Normal Form Theorem

**Normal Form Theorem**. $M(\widetilde{x})$ is partially decidable iff there is a primitive recursive predicate $R(\widetilde{x}, y)$ such that $M(\widetilde{x})$ iff $\exists y.R(\widetilde{x}, y)$.

# Normal Form Theorem

**Normal Form Theorem**. $M(\widetilde{x})$ is partially decidable iff there is a primitive recursive predicate $R(\widetilde{x}, y)$ such that $M(\widetilde{x})$ iff $\exists y.R(\widetilde{x}, y)$.

*Proof*. If $R(\widetilde{x}, y)$ is primitive recursive and $M(\widetilde{x}) \Leftrightarrow \exists y.R(\widetilde{x}, y)$, then the computable function 'if $\mu y R(\widetilde{x}, y)$ then **1** else $\uparrow$' is the partial characteristic function of $M(\widetilde{x})$.

# Normal Form Theorem

**Normal Form Theorem**. $M(\widetilde{x})$ is partially decidable iff there is a primitive recursive predicate $R(\widetilde{x}, y)$ such that $M(\widetilde{x})$ iff $\exists y.R(\widetilde{x}, y)$.

*Proof*. If $R(\widetilde{x}, y)$ is primitive recursive and $M(\widetilde{x}) \Leftrightarrow \exists y.R(\widetilde{x}, y)$, then the computable function 'if $\mu y R(\widetilde{x}, y)$ then **1** else $\uparrow$' is the partial characteristic function of $M(\widetilde{x})$.

Conversely suppose $M(\widetilde{x})$ is partially decided by $P$. Let $R(\widetilde{x}, y)$ be

$$P(\widetilde{x}) \downarrow \text{ in } y \text{ steps.}$$

Then $R(\widetilde{x}, y)$ is primitive recursive and $M(\widetilde{x}) \Leftrightarrow \exists y.R(\widetilde{x}, y)$.

# Quantifier Contraction Theorem

**Quantifier Contraction Theorem**. If $M(\widetilde{x}, y)$ is partially decidable, so is $\exists y.M(\widetilde{x}, y)$.

# Quantifier Contraction Theorem

**Quantifier Contraction Theorem**. If $M(\widetilde{x}, y)$ is partially decidable, so is $\exists y.M(\widetilde{x}, y)$.

*Proof*. Let $R(\widetilde{x}, y, z)$ be a primitive recursive predicate such that

$$M(\widetilde{x}, y) \Leftrightarrow \exists z.R(\widetilde{x}, y, z)$$

Then $\exists y.M(\widetilde{x}, y) \Leftrightarrow \exists y.\exists z.R(\widetilde{x}, y, z) \Leftrightarrow \exists u.R(\widetilde{x}, (u)_0, (u)_1)$.

# Examples

The following predicates are partially decidable:

$x \in E_y^{(n)}$

# Examples

The following predicates are partially decidable:

$x \in E_y^{(n)}$

$W_x \neq \emptyset$

# Uniformisation Theorem

**Uniformisation Theorem**. If $R(x, y)$ is partially decidable, then there is a computable function $c(x)$ such that $c(x) \downarrow$ iff $\exists y.R(x, y)$ and $c(x) \downarrow$ implies $R(x, c(x))$.

# Uniformisation Theorem

**Uniformisation Theorem**. If $R(x,y)$ is partially decidable, then there is a computable function $c(x)$ such that $c(x)\downarrow$ iff $\exists y.R(x,y)$ and $c(x)\downarrow$ implies $R(x,c(x))$.

We may think of $c(x)$ as a choice function for $R(x,y)$. The theorem states that the choice function is computable.

$A$ is r.e. iff there is a partially decidable predicate $R(x, y)$ such that $x \in A$ iff $\exists y.R(x, y)$.

# Complementation Theorem

**Complementation Theorem**. $A$ is recursive iff $A$ and $\overline{A}$ are r.e.

# Complementation Theorem

**Complementation Theorem**. $A$ is recursive iff $A$ and $\overline{A}$ are r.e.

*Proof*. Suppose $A$ and $\overline{A}$ are r.e. Then some primitive recursive predicates $R(x,y), S(x,y)$ exist such that

$$\begin{aligned} x \in A &\Leftrightarrow \exists y R(x,y), \\ x \in \overline{A} &\Leftrightarrow \exists y S(x,y). \end{aligned}$$

Now let $f(x)$ be $\mu y(R(x,y) \vee S(x,y))$.

Then $f(x)$ is total and computable, and

$$x \in A \Leftrightarrow R(x, f(x))$$

# Applying Complementation Theorem

**Fact**. $\overline{K}$ is not r.e.

# Applying Complementation Theorem

**Fact**. $\overline{K}$ is not r.e.

**Comment**. If $\overline{K} \leq_m A$ then $A$ is not r.e. either.

# Applying Complementation Theorem

**Fact**. If $A$ is r.e. but not recursive, then $\overline{A} \not\leq_m A \not\leq_m \overline{A}$.

# Applying Complementation Theorem

**Fact**. If $A$ is r.e. but not recursive, then $\overline{A} \not\leq_m A \not\leq_m \overline{A}$.

**Comment**. However $A$ and $\overline{A}$ are intuitively equally difficult.

# Graph Theorem

**Graph Theorem**. Let $f(x)$ be a partial function. Then $f(x)$ is computable iff the predicate '$f(x) \simeq y$' is partially decidable iff $\{\pi(x,y) \mid f(x) \simeq y\}$ is r.e.

# Graph Theorem

**Graph Theorem**. Let $f(x)$ be a partial function. Then $f(x)$ is computable iff the predicate '$f(x) \simeq y$' is partially decidable iff $\{\pi(x,y) \mid f(x) \simeq y\}$ is r.e.

*Proof*. If $f(x)$ is computable by $P(x)$, then

$$f(x) \simeq y \Leftrightarrow \exists t.(P(x) \downarrow y \text{ in } t \text{ steps})$$

The predicate '$P(x) \downarrow y$ in $t$ steps' is primitive recursive.

Conversely let $R(x,y,t)$ be such that

$$f(x) \simeq y \Leftrightarrow \exists t.R(x,y,t).$$

Now $f(x) = \mu y.R(x,y,\mu t.R(x,y,t))$.

# Listing Theorem

**Listing Theorem**. $A$ is r.e. iff either $A = \emptyset$ or $A$ is the range of a unary total computable function.

# Listing Theorem

**Listing Theorem**. $A$ is r.e. iff either $A = \emptyset$ or $A$ is the range of a unary total computable function.

*Proof*. Suppose $A$ is nonempty and its partial characteristic function is computed by $P$. Let $a$ be a member of $A$. The total function $g(x,t)$ given by

$$g(x,t) = \begin{cases} x, & \text{if } P(x)\downarrow \text{ in } t \text{ steps,} \\ a, & \text{otherwise.} \end{cases}$$

is computable. Clearly $A$ is the range of $h(z) = g((z)_1, (z)_2)$.

# Listing Theorem

**Listing Theorem**. $A$ is r.e. iff either $A = \emptyset$ or $A$ is the range of a unary total computable function.

*Proof*. Suppose $A$ is nonempty and its partial characteristic function is computed by $P$. Let $a$ be a member of $A$. The total function $g(x,t)$ given by

$$g(x,t) = \begin{cases} x, & \text{if } P(x)\downarrow \text{ in } t \text{ steps,} \\ a, & \text{otherwise.} \end{cases}$$

is computable. Clearly $A$ is the range of $h(z) = g((z)_1, (z)_2)$.

Conversely, $x \in A$ iff $\exists y.h(y) = x$, $\exists y.h(y) = x$ is partially decidable.

# Listing Theorem

The theorem gives rise to the terminology ‘recursively enumerable’.

# Implication of Listing Theorem

A set is r.e. iff it is the range of a computable function.

# Implication of Listing Theorem

**Corollary**. For each infinite nonrecursive r.e. $A$, there is an injective total recursive function $f$ such that $ran(f) = A$.

# Implication of Listing Theorem

**Corollary**. For each infinite nonrecursive r.e. $A$, there is an injective total recursive function $f$ such that $ran(f) = A$.

**Corollary**. Every infinite r.e. set has an infinite recursive subset.

# Implication of Listing Theorem

**Corollary**. For each infinite nonrecursive r.e. $A$, there is an injective total recursive function $f$ such that $ran(f) = A$.

**Corollary**. Every infinite r.e. set has an infinite recursive subset.

*Proof*. Suppose $A = ran(f)$. An infinite recursive subset is enumerated by the total increasing computable function $g$ given by

$$\begin{aligned} g(0) &= f(0), \\ g(n+1) &= f(\mu y(f(y) > g(n))). \end{aligned}$$

# Applying Listing Theorem

**Fact**. The set $\{x \mid \phi_x \text{ is total}\}$ is not r.e.

# Applying Listing Theorem

**Fact**. The set $\{x \mid \phi_x \text{ is total}\}$ is not r.e.

*Proof*. If $\{x \mid \phi_x \text{ is total}\}$ were a r.e. set, then there would be a total computable function $f$ whose range is the r.e. set.

The function $g(x)$ given by $g(x) = \phi_{f(x)}(x) + 1$ would be total and computable.

# Rice-Shapiro Theorem

# Rice-Shapiro Theorem

**Rice-Shapiro Theorem**. Suppose that $\mathcal{A}$ is a set of unary computable functions such that the set $\{x \mid \phi_x \in \mathcal{A}\}$ is r.e.
Then for any unary computable function $f$, $f \in \mathcal{A}$ iff there is a finite function $\theta \subseteq f$ with $\theta \in \mathcal{A}$.

# Rice-Shapiro Theorem

**Rice-Shapiro Theorem**. Suppose that $\mathcal{A}$ is a set of unary computable functions such that the set $\{x \mid \phi_x \in \mathcal{A}\}$ is r.e.
Then for any unary computable function $f$, $f \in \mathcal{A}$ iff there is a finite function $\theta \subseteq f$ with $\theta \in \mathcal{A}$.

**Comment**. Intuitively a set of recursive functions is r.e. iff it is effectively generated by an r.e. set of finite functions.

# Applications of the Rice-Shapiro Theorem

Both $Tot$ and $\overline{Tot}$ are not r.e.

$$Tot = \{x \mid \phi_x \text{ is total}\}$$

# Applications of the Rice-Shapiro Theorem

Both $Tot$ and $\overline{Tot}$ are not r.e.

$$Tot = \{x \mid \phi_x \text{ is total}\}$$

Proof

We apply the Rice-Shapiro theorem on $Tot$. For no $f \in Tot$ is there a finite $\theta \subseteq f$ with $\theta \in Tot$.

# Applications of the Rice-Shapiro Theorem

Both $Tot$ and $\overline{Tot}$ are not r.e.

$$Tot = \{x \mid \phi_x \text{ is total}\}$$

Proof

We apply the Rice-Shapiro theorem on $Tot$. For no $f \in Tot$ is there a finite $\theta \subseteq f$ with $\theta \in Tot$.

# Applications of the Rice-Shapiro Theorem

Both $Tot$ and $\overline{Tot}$ are not r.e.

$$Tot = \{x \mid \phi_x \text{ is total}\}$$

Proof

We apply the Rice-Shapiro theorem on $Tot$. For no $f \in Tot$ is there a finite $\theta \subseteq f$ with $\theta \in Tot$.

If $f$ is any total computable function, $f \notin \overline{Tot}$; but every finite function $\theta \subseteq f$ in $\overline{Tot}$.

# What Rice-Shapiro Theorem Can Do

Can we apply Rice-Shapiro Theorem to show that any of the following sets is non-r.e.:

$$
\begin{aligned}
\mathit{Fin} &= \{x \mid W_x \text{ is finite}\},\\
\mathit{Inf} &= \{x \mid W_x \text{ is infinite}\},\\
\mathit{Tot} &= \{x \mid \phi_x \text{ is total}\},\\
\mathit{Con} &= \{x \mid \phi_x \text{ is total and constant}\},\\
\mathit{Cof} &= \{x \mid W_x \text{ is cofinite}\},\\
\mathit{Rec} &= \{x \mid W_x \text{ is recursive}\},\\
\mathit{Ext} &= \{x \mid \phi_x \text{ is extensible to a total recursive function}\}.
\end{aligned}
$$

# Proof of Rice-Shapiro Theorem

Suppose $A = \{x \mid \phi_x \in \mathcal{A}\}$ is r.e.

($\Rightarrow$): Suppose $f \in \mathcal{A}$ but for all finite $\theta \subseteq f.\theta \notin \mathcal{A}$.

Let $P$ be a partial characteristic function of $K$. Define the computable function $g(z,t)$ by

$$g(z,t) \simeq \begin{cases} f(t), & \text{if } P(z) \downarrow \text{ in } t \text{ steps,} \\ \uparrow, & \text{otherwise.} \end{cases}$$

According to S-m-n Theorem, there is an injective primitive recursive function $s(z)$ such that $g(z,t) \simeq \phi_{s(z)}(t)$.

By construction $\phi_{s(z)} \subseteq f$ for all $z$.

$z \in K \Rightarrow \phi_{s(z)}$ is finite $\Rightarrow s(z) \notin A$;
$z \notin K \Rightarrow \phi_{s(z)} = f \Rightarrow s(z) \in A$.

# Proof of Rice-Shapiro Theorem

($\Leftarrow$): Suppose $f$ is a computable function and there is a finite $\theta \in \mathcal{A}$ such that $\theta \subseteq f$ and $f \notin \mathcal{A}$.

Define the computable function $g(z,t)$ by

$$g(z,t) \simeq \begin{cases} f(t), & \text{if } t \in Dom(\theta) \vee z \in K, \\ \uparrow, & \text{otherwise.} \end{cases}$$

According to S-m-n Theorem, there is an injective primitive recursive function $s(z)$ such that $g(z,t) \simeq \phi_{s(z)}(t)$.

$z \in K \Rightarrow \phi_{s(z)} = f \Rightarrow s(z) \notin A$;
$z \notin K \Rightarrow \phi_{s(z)} = \theta \Rightarrow s(z) \in A$.

# Reversing Rice-Shapiro Theorem

$\{x \mid \phi_x \in \mathcal{A}\}$ is r.e. if the following hold:

1. $\Theta = \{e(\theta) \mid \theta \in \mathcal{A} \text{ and } \theta \text{ is finite}\}$ is r.e., where $e$ is a canonical effective encoding of the finite functions.
2. $\forall f \in \mathcal{A}.\exists \text{ finite } \theta \in \mathcal{A}.\theta \subseteq f$.

# Reversing Rice-Shapiro Theorem

$\{x \mid \phi_x \in \mathcal{A}\}$ is r.e. if the following hold:

1. $\Theta = \{e(\theta) \mid \theta \in \mathcal{A} \text{ and } \theta \text{ is finite}\}$ is r.e., where $e$ is a canonical effective encoding of the finite functions.
2. $\forall f \in \mathcal{A}.\exists \text{ finite } \theta \in \mathcal{A}.\theta \subseteq f$.

**Comment**. We cannot take $e$ as the Gödel encoding function of the recursive functions. Why?

# Reversing Rice-Shapiro Theorem

$\{x \mid \phi_x \in \mathcal{A}\}$ is r.e. if the following hold:

1. $\Theta = \{e(\theta) \mid \theta \in \mathcal{A} \text{ and } \theta \text{ is finite}\}$ is r.e., where $e$ is a canonical effective encoding of the finite functions.
2. $\forall f \in \mathcal{A}.\exists \text{ finite } \theta \in \mathcal{A}.\theta \subseteq f$.

**Comment**. We cannot take $e$ as the Gödel encoding function of the recursive functions. Why? How would you define $e$?

# Homework

- Homework 6: Exercise 6.14, pp. 119 of the textbook.

# Computability Theory XII

## Various Sets

Guoqiang Li

Shanghai Jiao Tong University

Dec. 26, 2014

Creative Set

# Most Difficult Semi-Decidable Problems?

An r.e. set is very difficult if it is very non-recursive.

An r.e. set is very non-recursive if its complement is very non-r.e..

A set is very non-r.e. if it is easy to distinguish it from any r.e. set.

# Most Difficult Semi-Decidable Problems?

An r.e. set is very difficult if it is very non-recursive.

An r.e. set is very non-recursive if its complement is very non-r.e..

A set is very non-r.e. if it is easy to distinguish it from any r.e. set.

These sets are creative respectively productive.

# Synopsis

1. Productive Set
2. Creative Set
3. Simple Sets

Productive Set

# Productive Set

Suppose $W_x \subseteq \overline{K}$. Then $x \in \overline{K} \setminus W_x$.

# Productive Set

Suppose $W_x \subseteq \overline{K}$. Then $x \in \overline{K} \setminus W_x$.

So $x$ witnesses the strict inclusion $W_x \subseteq \overline{K}$.

In other words the identity function is an effective proof that $\overline{K}$ differs from every r.e. set.

# Productive Set

A set $A$ is productive if there is a total computable function $p$ such that whenever $W_x \subseteq A$, then $p(x) \in A \setminus W_x$. The function $p$ is called a productive function for $A$.

# Productive Set

A set $A$ is productive if there is a total computable function $p$ such that whenever $W_x \subseteq A$, then $p(x) \in A \setminus W_x$. The function $p$ is called a productive function for $A$.

A productive set is not r.e. by definition.

# Productive Set

$\overline{K}$ is productive.

# Productive Set

$\overline{K}$ is productive.

$\{x \mid c \notin W_x\}$ is productive.

# Productive Set

$\overline{K}$ is productive.

$\{x \mid c \notin W_x\}$ is productive.

$\{x \mid c \notin E_x\}$ is productive.

# Productive Set

$\overline{K}$ is productive.

$\{x \mid c \notin W_x\}$ is productive.

$\{x \mid c \notin E_x\}$ is productive.

$\{x \mid \phi_x(x) \neq 0\}$ is productive.

# Productive Set

Suppose $A = \{x \mid \phi_x(x) \neq 0\}$.

# Productive Set

Suppose $A = \{x \mid \phi_x(x) \neq 0\}$.

By S-m-n Theorem one gets a primitive recursive function $p(x)$ such that $\phi_{p(x)}(y) = 0$ if and only if $\phi_x(y)$ is defined. Then

$$p(x) \in W_x \Leftrightarrow p(x) \notin A.$$

So if $W_x \subseteq A$ we must have $p(x) \in A \setminus W_x$.
Thus $p$ is a productive function for $A$.

# Productive Set

**Lemma**. If $A \leq_m B$ and $A$ is productive, then $B$ is productive.

# Productive Set

**Theorem**. Suppose that $\mathcal{B}$ is a set of unary computable functions with $f_\emptyset \in \mathcal{B}$ and $\mathcal{B} \neq \mathcal{C}$. Then $B = \{x \mid \phi_x \in \mathcal{B}\}$ is productive.

# Productive Set

**Theorem**. Suppose that $\mathcal{B}$ is a set of unary computable functions with $f_\emptyset \in \mathcal{B}$ and $\mathcal{B} \neq \mathcal{C}$. Then $B = \{x \mid \phi_x \in \mathcal{B}\}$ is productive.

*Proof*. Suppose $g \notin \mathcal{B}$. Consider the function $f$ defined by

$$f(x,y) = \begin{cases} g(y), & \text{if } x \in W_x, \\ \uparrow, & \text{if } x \notin W_x. \end{cases}$$

By S-m-n Theorem there is a primitive recursive function $k(x)$ such that $\phi_{k(x)}(y) \simeq f(x,y)$.

Clearly $x \notin W_x$ iff $\phi_{k(x)} = f_\emptyset$ iff $\phi_{k(x)} \in \mathcal{B}$ iff $k(x) \in B$.

Hence $k : \overline{K} \leq_m B$.

# Property of Productive Set

**Lemma**. Suppose that $g$ is a total computable function. Then there is a primitive recursive function $p$ such that for all $x$,
$W_{p(x)} = W_x \cup \{g(x)\}$.

# Property of Productive Set

**Lemma**. Suppose that $g$ is a total computable function. Then there is a primitive recursive function $p$ such that for all $x$,
$W_{p(x)} = W_x \cup \{g(x)\}$.

*Proof*. Using S-m-n Theorem, take $p(x)$ to be a primitive recursive function such that

$$\phi_{p(x)}(y) = \begin{cases} 1, \text{if } y \in W_x \vee y = g(x), \\ \uparrow, \text{otherwise} \end{cases}$$

# Property of Productive Set

**Theorem**. A productive set contains an infinite r.e. subset.

# Property of Productive Set

**Theorem**. A productive set contains an infinite r.e. subset.

*Proof*. Suppose $p$ is a production function for $A$.

Take $e_0$ to be some index for $\emptyset$. Then $p(e_0) \in A$ by definition.

By the Lemma there is a primitive recursive function $k$ such that for all $x$, $W_{k(x)} = W_x \cup \{p(x)\}$.

Apparently $\{e_0, \ldots, k^n(e_0), \ldots\}$ is r.e.

Consequently $\{p(e_0), \ldots, p(k^n(e_0)), \ldots\}$ is a r.e. subset of $A$, which must be infinite by the definition of $k$.

# Productive Function via a Partial Function

**Proposition**. A set $A$ is productive iff there is a partial recursive function $p$ such that

$$\forall x.(W_x \subseteq A \Rightarrow (p(x)\downarrow \wedge p(x) \in A \setminus W_x)). \tag{1}$$

# Productive Function via a Partial Function

**Proposition**. A set $A$ is productive iff there is a partial recursive function $p$ such that

$$\forall x.(W_x \subseteq A \Rightarrow (p(x)\downarrow \wedge p(x) \in A \setminus W_x)). \tag{1}$$

*Proof*. Suppose $p$ is a partial recursive function satisfying (1). Let $s$ be a primitive recursive function such that

$$\phi_{s(x)}(y) = \begin{cases} y, & p(x)\downarrow \wedge\, y \in W_x, \\ \uparrow, & \text{otherwise.} \end{cases}$$

A productive function $q$ can be defined by running $p(x)$ and $p(s(x))$ in parallel and stops when either terminates.

# Productive Function Made Injective

**Proposition**. A productive set has an injective productive function.

# Productive Function Made Injective

**Proposition**. A productive set has an injective productive function.

*Proof*. Suppose $p$ is a productive function of $A$. Let

$$W_{h(x)} = W_x \cup \{p(x)\}$$

Clearly

$$W_x \subseteq A \Rightarrow W_{h(x)} \subseteq A$$

Define $q(0) = p(0)$.

If there is a least $y \in \{p(x+1), ph(x+1), ph^2(x+1), \ldots\}$ such that $y \notin \{q(0), \ldots, q(x)\}$, let $q(x+1)$ be $y$;

otherwise let $q(x+1)$ be $\mu y.y \notin \{q(0), \ldots, q(x)\}$.

It is easily seen that $q$ is an injective production function for $A$.

# Myhill's Characterization of Productive Set

**Fact**. $\overline{K} \leq_m A$ iff $\overline{K} \leq_1 A$.

# Myhill's Characterization of Productive Set

**Theorem**. (Myhill, 1955) $A$ is productive iff $\overline{K} \leq_1 A$ iff $\overline{K} \leq_m A$.

Creative Set

# Creative Set

A set $A$ is creative if it is r.e. and its complement $\overline{A}$ is productive.

# Creative Set

A set $A$ is creative if it is r.e. and its complement $\overline{A}$ is productive.

Intuitively a creative set $A$ is effectively non-recursive in the sense that the non-recursiveness of $\overline{A}$, hence the non-recursiveness of $A$, can be effectively demonstrated.

# Creative Set

$K$ is creative.

$\{x \mid c \in W_x\}$ is creative.

$\{x \mid c \in E_x\}$ is creative.

$\{x \mid \phi_x(x) = 0\}$ is creative.

# Creative Set

**Theorem**. Suppose that $\mathcal{A} \subseteq \mathcal{C}$ and let $A = \{x \mid \phi_x \in \mathcal{A}\}$. If $A$ is r.e. and $A \neq \emptyset, \mathbb{N}$, then $A$ is creative.

# Creative Set

**Theorem**. Suppose that $\mathcal{A} \subseteq \mathcal{C}$ and let $A = \{x \mid \phi_x \in \mathcal{A}\}$. If $A$ is r.e. and $A \neq \emptyset, \mathbb{N}$, then $A$ is creative.

*Proof*. Suppose $A$ is r.e. and $A \neq \emptyset, \mathbb{N}$. If $f_\emptyset \in \mathcal{A}$, then $A$ is productive by a previous theorem. This is a contradiction.

So $\overline{A}$ is productive by the same theorem. Hence $A$ is creative.

# Creative Set

The set $K_0 = \{x \mid W_x \neq \emptyset\}$ is creative. It corresponds to the set $\mathcal{A} = \{f \in \mathcal{C} \mid f \neq f_\emptyset\}$.

# Discussion

**Question**. Are all non-recursive r.e. sets creative?

# Discussion

**Question**. Are all non-recursive r.e. sets creative?

The answer is negative. By a special construction we can obtain r.e.sets that are neither recursive nor creative.

# Simple Sets

# Simple Sets

A set $A$ is simple if

1. $A$ is r.e.,
2. $\overline{A}$ is infinite,
3. $\overline{A}$ contains no infinite r.e. subset.

# Simple Sets

**Theorem**. A simple set is neither recursive nor creative.

# Simple Sets

**Theorem**. A simple set is neither recursive nor creative.

*Proof*. Since $\overline{A}$ can not be r.e., $A$ can not be recursive.

# Simple Sets

**Theorem**. A simple set is neither recursive nor creative.

*Proof*. Since $\overline{A}$ can not be r.e., $A$ can not be recursive.

(3) implies that $A$ can not be creative.

# Simple Sets

**Theorem**. There is a simple set.

# Simple Sets

**Theorem**. There is a simple set.

*Proof*. Define $f(x) = \phi_x(\mu z(\phi_x(z) > 2x))$. Let $A$ be $Ran(f)$.

1. $A$ is r.e.

# Simple Sets

**Theorem**. There is a simple set.

*Proof*. Define $f(x) = \phi_x(\mu z(\phi_x(z) > 2x))$. Let $A$ be $Ran(f)$.

1. $A$ is r.e.
2. $\overline{A}$ is infinite. This is because $A \cap \{0, 1, \ldots, 2n\}$ contains at most the elements $\{f(0), f(1), \ldots, f(n-1)\}$.

# Simple Sets

**Theorem**. There is a simple set.

*Proof*. Define $f(x) = \phi_x(\mu z(\phi_x(z) > 2x))$. Let $A$ be $Ran(f)$.

1. $A$ is r.e.
2. $\overline{A}$ is infinite. This is because $A \cap \{0, 1, \ldots, 2n\}$ contains at most the elements $\{f(0), f(1), \ldots, f(n-1)\}$.
3. Suppose $B$ is an infinite r.e. set. Then there is a total computable function $\phi_b$ such that $B = E_b$. Since $\phi_b$ is total, $f(b)$ is defined and $f(b) \in A$. Hence $B \not\subseteq \overline{A}$.

# Computability Theory I

## Introduction

Guoqiang Li

Shanghai Jiao Tong University

Sep. 19, 2014

# Instructor and Teaching Assistant

- Guoqiang LI
  - Homepage: http://basics.sjtu.edu.cn/∼liguoqiang
  - Course page: http://basics.sjtu.edu.cn/∼liguoqiang/teaching/comp14/index.htm
  - Email: li-gq@cs.sjtu.edu.cn
  - Office: Rm. 1212, Building of Software
  - Phone: 3420-4167
  - YSSY: many IDs…
  - weibo: http://www.weibo.com/flyinsail
- TA:
  - Mingzhang HUANG: mingzhanghuang@gmail.com
  - Xiuting TAO: taoxiuting@gmail.com
- Office hour: Wed. 14:00-17:00 @ SEIEE 3-327

What do you think you can learn from this course?

# Aim of the Course

- Q: Can the course improve the skill of programming?
  - A: Nope!
- Q: Can the course improve the ability of algorithms?
  - A: Perhaps, seldom.
- The course may provide a view of computation, an overlook of what we are doing in computer science, and a basic study of theoretical computer science.
- It is rather a philosophy than a technique, although some parts are quite technically.

# It May Answer

- A software company that is developing a compiler capable of checking if a program contains a loop.
- A hardware company that is determined to design a computer that can solve problems that no existing computers can solve.
- A service provider that is working on a theorem prover that is supposed to answer every question about numbers.

# History of Velocity

Human beings are keen on speed, and cannot stop the step to chase moving as fast as possible.

- Wheels: Mid-4th millennium BC.
- Automobiles: 1762
- Trains: 1807
- Airplanes: 1903
- Supersonic: 1947 $343.2m/s$
- Circular velocity 1957 $7.9km/s$
- Earth escape velocity 1959 $11.2km/s$
- Solar system escape velocity 1977 $16.7km/s$
- Q: Can we achieve in arbitrarily fast velocity?
    - Grandfather paradox

# Computation

Human beings are also keen on computation, and cannot stop the step to chase computing as complex as possible.

- Decimal system: AD 600
- Basic arithmetic: Al Khwarizmi (780 - 850)
- ENIAC: 1946
- NP problem
- The curse of exponential time
- Advanced algorithms: simplex, DPLL, antichain.
- Q: Can we achieve in arbitrarily complex computation?

What problems can be solved by computers?

*Computer science is no more about computers than astronomy is about telescopes.*

*Edsger Dijkstra*

*Let us begin to learn some basic astronomical phenomena!*

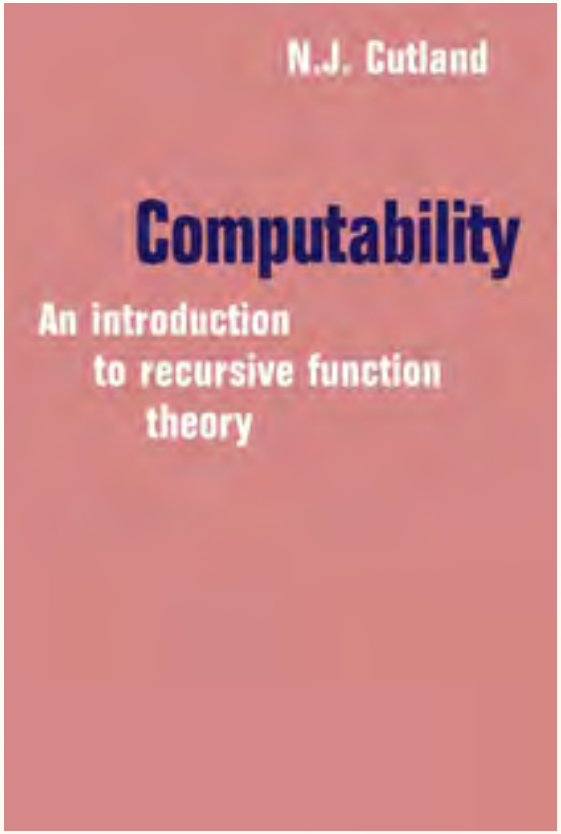
*The technique part is quite similar to puzzles of wise men.*

*So, please have a fun!*

*Intuition is extremely important!*

# Reference Book

- Computability: An Introduction to Recursive Function Theory.
  - Nigel J. Cutland
- plus extra reading materials.



# Scoring Policy

- 10% Attendance.
- 20% Assignments.
  - Four assignments.
  - Each one is 5 pts.
  - Work out individually.
- 70% Final exam.
- There are also several homework. The answer may be given in exercise lectures, two or three times.

# Special Requirements

A notebook and a pen.

Any questions?

# 0. Prologue

Effective Solutions

What problems can be solved by computers?

# Famous Problems

- Diophantine equations
- Shortest path problem
- Travelling salesman problem (TSP)
- Graph isomorphism problem (GI)

# Intuition

An effective procedure consists of a finite set of instructions which, given an input from some set of possible inputs, enables us to obtain an output through a systematic execution of the instructions that terminates in a finite number of steps.

# Intuition

Theorem proving is in general not effective.

Proof verification is effective.

Unbounded search is in general not effective.

Bounded search is effective.

# Representation of Problem

- How does a computer solve the GI problem or the TSP Problem?
- How is a problem instance (a graph) represented in a computer?
- How is the answer to a problem instance represented?
- How is an effective procedure formalized?

- Can every function from $\mathbb{N}$ to $\mathbb{N}$ be calculated by a C program?
  - Negative.

# Punchline

- In a formal theory of computability, every problem instance can be represented by a number and every number represents a problem instance.
- A problem is a function $f : \mathbb{N} \to \mathbb{N}$ from numbers to numbers.
- A problem is computable if it can be calculated by a program.

*Everything is number!*

*Pythagoras*

# Decision Problem

# Decision Problem

A problem $f : \mathbb{N} \to \mathbb{N}$ is a decision problem if the range $ran(f)$ of $f$ is $\{0, 1\}$, where $1$ denotes a 'yes' answer and $0$ a 'no' answer.

A decision problem $g$ can be identified with the set $\{n \mid g(n) = 1\}$.

Conversely a subset $A$ of $\mathbb{N}$ can be seen as a decision problem via the characteristic function of $A$:

$$c_A(n) \quad = \quad \begin{cases} 1, & \text{if } x \in A, \\ 0, & \text{otherwise.} \end{cases}$$

# Decision Problem as Predicate

A decision problem can be stated as a predicate $P(x)$ on number.

It relates to the problem-as-function viewpoint by the following characteristic function of $P(x)$:

$$c_P(n) = \begin{cases} 1, & \text{if } P(n) \text{ is valid,} \\ 0, & \text{otherwise.} \end{cases}$$

$$\begin{aligned} \text{Decision Problem} &\Leftrightarrow \text{Subset of } \mathbb{N} \\ &\Leftrightarrow \text{Predicate on } \mathbb{N} \end{aligned}$$

Several Problems

# Problem I

Is the function $tower(x)$ defined below computable?

$$tower(x) = \underbrace{2^{2^{\cdots^{2}}}}_{x}$$

Theoretically it is computable.

# Problem II

Consider the function $f$ defined as follows:

$$f(n) = \begin{cases} 1, & \text{if } n > 1 \text{ and } 2n \text{ is the sum of 2 primes,} \\ 0, & \text{otherwise.} \end{cases}$$

The Goldbach Conjecture remains unsolved. Is $f$ computable?

It is clearly computable even if we do not know what it is.

# Problem III

Consider the function $g$ defined as follows:

$$g(n) = \begin{cases} 1, & \text{if there is a run of exactly } n \text{ consecutive } 7's \\ & \text{in the decimal expansion of } \pi, \\ 0, & \text{otherwise.} \end{cases}$$

It is known that $\pi$ can be calculated by $4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots\right)$.
Is $g$ computable?

We do not know whether it is computable or not.

# Problem IV

Consider the function $h$ defined as follows:

$$h(n) = \begin{cases} 1, & \text{if } n \text{ is the machine code of a } C \text{ program that} \\ & \text{terminates in all inputs,} \\ 0, & \text{otherwise.} \end{cases}$$

This is the Halting Problem, a well known undecidable problem. In other words there does not exist any C program calculating $h$.

The only general approach to check if a function is defined on all numbers is to calculate it on all inputs.

# Problem V

Consider the function $i$ defined as follows:

$$i(x,n,t) = \begin{cases} 1, & \text{if on input } x, \text{ the machine coded by } n \\ & \text{terminates in } t \text{ steps,} \\ 0, & \text{otherwise.} \end{cases}$$

There could be a number of ways to interpret "$t$ steps".

The function $i$ is intuitively computable.

# Next Lecture

The examples try to suggest that in order to study computability one might as well look for a theory of computable functions.

We will begin with a machine model, register machine.

# Homework

- home reading: diagonal method.
- home reading: Presburger arithmetic.

# Computability Theory II

## Unlimited Register Machine

Guoqiang Li

Shanghai Jiao Tong University

Sep. 26, 2014

# Review Tips

# Computable Functions

- In a formal theory of computability, every problem instance can be represented by a number and every number represents a problem instance.
- A problem is a function $f : \mathbb{N} \to \mathbb{N}$ from numbers to numbers.
- A problem is computable if it can be calculated by a program.

# Decision Problem

A problem $f : \mathbb{N} \to \mathbb{N}$ is a decision problem if the range $ran(f)$ of $f$ is $\{0, 1\}$, where $1$ denotes a 'yes' answer and $0$ a 'no' answer.

A decision problem $g$ can be identified with the set $\{n \mid g(n) = 1\}$.

Conversely a subset $A$ of $\mathbb{N}$ can be seen as a decision problem via the characteristic function of $A$:

$$c_A(n) \quad = \quad \begin{cases} 1, & \text{if } x \in A, \\ 0, & \text{otherwise.} \end{cases}$$

# Decision Problem as Predicate

A decision problem can be stated as a predicate $P(x)$ on number.

It relates to the problem-as-function viewpoint by the following characteristic function of $P(x)$:

$$c_P(n) = \begin{cases} 1, & \text{if } P(n) \text{ is valid,} \\ 0, & \text{otherwise.} \end{cases}$$

$$\begin{aligned} \text{Decision Problem} &\Leftrightarrow \text{Subset of } \mathbb{N} \\ &\Leftrightarrow \text{Predicate on } \mathbb{N} \end{aligned}$$

# Register Machine

# Remark

Register Machines are more advanced than Turing Machines.

# Remark

Register Machine Models can be classified into three groups:

- CM (Counter Machine Model).
- RAM (Random Access Machine Model).
- RASP (Random Access Stored Program Machine Model).

# Synopsis

1. Unlimited Register Machine
2. Definability in URM

# Unlimited Register Machine

# Unlimited Register Machine Model

The Unlimited Register Machine Model belongs to the CM class.

Computability and Recursive Functions, by J. Shepherdson and H. Sturgis, in Journal of Symbolic Logic (**32**):1-63, 1965.

# Register

An Unlimited Register Machine (URM) has an infinite number of register labeled $R_1, R_2, R_3, \ldots$.


$R_1$ $R_2$ $R_3$ $R_4$ $R_5$ $R_6$ $R_7$ …

Every register can hold a natural number at any moment.

The registers can be equivalently written as for example

$$[r_1, r_2, r_3]_1^3 [r_4]_4^4 [r_5, r_6, r_7]_5^7 [0, 0, 0, \ldots]_8^\infty$$

or simply

$$[r_1, r_2, r_3]_1^3 [r_4]_4^4 [r_5, r_6, r_7]_5^7.$$

# Program

A URM also has a program, which is a finite list of instructions.

# Instruction

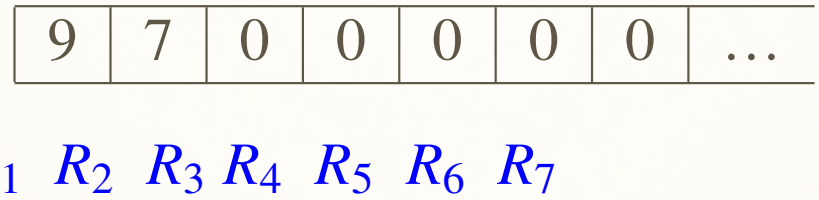
| Type | Instruction | Response of the URM |
|---|---|---|
| Zero | $Z(n)$ | Replace $r_n$ by $0$. |
| Successor | $S(n)$ | Add $1$ to $r_n$. |
| Transfer | $T(m,n)$ | Copy $r_m$ to $R_n$. |
| Jump | $J(m,n,q)$ | If $r_m = r_n$, go to the $q$-th instruction; otherwise go to the next instruction. |

# Program Rules

- $P = \{I_1, I_2, \cdots, I_s\} \rightarrow$ URM.
- URM starts by obeying instruction $I_1$.
- When URM finishes obeying $I_k$, it proceeds to the next instruction in the computation,
  - if $I_k$ is not a jump instruction, then the next instruction is $I_{k+1}$;
  - if $I_k = J(m,n,q)$ then next instruction is
    - $I_q$, if $r_m = r_n$; or
    - $I_{k+1}$, otherwise.
- Computation stops when the next instruction is $I_v$, where $v > s$.
  - if $k = s$, and $I_s$ is an arithmetic instruction;
  - if $I_k = J(m,n,q)$, $r_m = r_n$ and $q > s$;
  - if $I_k = J(m,n,q)$, $r_m \neq r_n$ and $k = s$.

# Computation

Registers:




Program:

$I_1 : J(1,2,6)$
$I_2 : S(2)$
$I_3 : S(3)$
$I_4 : J(1,2,6)$
$I_5 : J(1,1,2)$
$I_6 : T(3,1)$

# Configuration and Computation

Configuration: register contents + current instruction number.

Initial configuration, computation, final configuration.

# Some Notation

Suppose $P$ is the program of a URM and $a_1, a_2, a_3, \ldots$ are the numbers stored in the registers.

- $P(a_1, a_2, \ldots, a_m)$ is $P(a_1, a_2, \ldots, a_m, 0, 0, \ldots)$.
- $P(a_1, a_2, a_3, \ldots)$ is the initial configuration.
- $P(a_1, a_2, a_3, \ldots)\downarrow$ means that the computation converges.
- $P(a_1, a_2, a_3, \ldots)\uparrow$ means that the computation diverges.

# Definability in URM

# URM-Computable Function

Let $f(\widetilde{x})$ be an $n$-ary (partial) function.

What does it mean that a URM computes $f(\widetilde{x})$?

# URM-Computable Function

Suppose $P$ is the program of a URM and $a_1, \ldots, a_n, b \in \mathbb{N}$.

The computation $P(a_1, \ldots, a_n)$ converges to $b$ if $P(a_1, \ldots, a_n) \downarrow$ and $r_1 = b$ in the final configuration.

In this case we write $P(a_1, \ldots, a_n) \downarrow b$.

$P$ URM-computes $f$ if, for all $a_1, \ldots, a_n, b \in \mathbb{N}$, $P(a_1, \ldots, a_n) \downarrow b$ iff $f(a_1, \ldots, a_n) = b$.

The function $f$ is URM-definable if there is a program that URM-computes $f$.

We shall abbreviate “URM-computable” to “computable”.

Let

$\mathscr{C}$

be the set of computable functions and

$\mathscr{C}_n$

be the set of $n$-ary computable functions.

# Example of URM I

Construct a URM that computes $x + y$.

$I_1 : J(3, 2, 5)$
$I_2 : S(1)$
$I_3 : S(3)$
$I_4 : J(1, 1, 1)$

# Example of URM II

Construct a URM that computes $x \dot{-} 1 = \begin{cases} x - 1, & \text{if } x > 0, \\ 0, & \text{if } x = 0. \end{cases}$

$I_1 : J(1,4,8)$
$I_2 : S(3)$
$I_3 : J(1,3,7)$
$I_4 : S(2)$
$I_5 : S(3)$
$I_6 : J(1,1,3)$
$I_7 : T(2,1)$

# Example of URM III

Construct a URM that computes $x \div 2 = \begin{cases} x/2, & \text{if } x \text{ is even,} \\ \text{undefined}, & \text{if } x \text{ is odd.} \end{cases}$

$I_1 : J(1,2,6)$
$I_2 : S(3)$
$I_3 : S(2)$
$I_4 : S(2)$
$I_5 : J(1,1,1)$
$I_6 : T(3,1)$

# Example of URM IV

Construct a URM that computes $f(x) = \lfloor 3x/4 \rfloor$

$I_1$ Z(2)
$I_2$ Z(3)
$I_3$ Z(4)
$I_4$ J(1,2,10)
$I_5$ S(2)
$I_6$ S(3)
$I_7$ S(3)
$I_8$ S(3)
$I_9$ J(1,1,4)
$I_{10}$ Z(2)
$I_{11}$ J(2,3,21)
$I_{12}$ S(2)
$I_{13}$ J(2,3,21)
$I_{14}$ S(2)
$I_{15}$ J(2,3,21)
$I_{16}$ S(2)
$I_{17}$ J(2,3,21)
$I_{18}$ S(2)
$I_{19}$ S(4)
$I_{20}$ J(1,1,11)
$I_{21}$ T(4,1)

# Function Defined by Program

$$f_P^n(a_1, \ldots, a_n) = \begin{cases} b, & \text{if } P(a_1, \ldots, a_n) \downarrow b, \\ \text{undefined}, & \text{if } P(a_1, \ldots, a_n) \uparrow . \end{cases}$$

# Program in Standard Form

A program $P = I_1, \ldots, I_s$ is in standard form if, for every jump instruction $J(m, n, q)$ we have $q \leq s + 1$.

For every program there is a program in standard form that computes the same function.

We will focus exclusively on programs in standard form.

# Program Composition

Given Programs $P$ and $Q$, how do we construct the sequential composition $P; Q$?

The jump instructions of $P$ and $Q$ must be modified.

# Some Notations

Suppose the program $P$ computes $f$.

Let $\rho(P)$ be the least number $i$ such that the register $R_i$ is not used by the program $P$.

# Some Notations

The notation $P[l_1, \ldots, l_n \to l]$ stands for the following program

$$
\begin{array}{rcl}
I_1 & : & T(l_1, 1) \\
 & \vdots & \\
I_n & : & T(l_n, n) \\
I_{n+1} & : & Z(n+1) \\
 & \vdots & \\
I_{\rho(P)} & : & Z(\rho(P)) \\
\_ & : & P \\
\_ & : & T(1, l)
\end{array}
$$

# Computability Theory III

## Primitive Recursive Function

Guoqiang Li

Shanghai Jiao Tong University

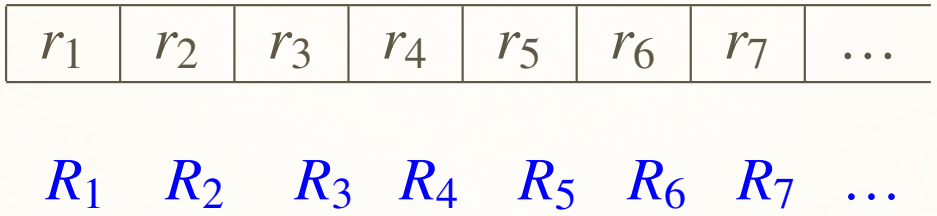
Oct. 10, 2014

*Assignment 1 is announced! (deadline Oct. 24)*

# Review Tips

# Register

An Unlimited Register Machine (URM) has an infinite number of register labeled $R_1, R_2, R_3, \ldots$.




Every register can hold a natural number at any moment.

The registers can be equivalently written as for example

$$[r_1, r_2, r_3]_1^3 [r_4]_4^4 [r_5, r_6, r_7]_5^7 [0, 0, 0, \ldots]_8^\infty$$

or simply

$$[r_1, r_2, r_3]_1^3 [r_4]_4^4 [r_5, r_6, r_7]_5^7.$$

# Instruction

| Type | Instruction | Response of the URM |
|---|---|---|
| Zero | $Z(n)$ | Replace $r_n$ by $0$. |
| Successor | $S(n)$ | Add $1$ to $r_n$. |
| Transfer | $T(m,n)$ | Copy $r_m$ to $R_n$. |
| Jump | $J(m,n,q)$ | If $r_m = r_n$, go to the $q$-th instruction; otherwise go to the next instruction. |

# Recursive Function

# Recursion Theory

Recursion Theory offers a mathematical model for the study of effective calculability.

1. All effective objects can be encoded by natural numbers.
2. All effective procedures can be modeled by functions from numbers to numbers.

# Synopsis

1. Primitive Recursive Function

# Primitive Recursive Function

# Basic Definitions

# Initial Function

1. The zero function
   - $\mathbf{0}$
   - $\mathbf{0}(\widetilde{x}) = 0$
2. The successor function
   - $s(x) = x + 1$
3. The projection function
   - $U_i^n(x_1, \ldots, x_n) = x_i$

# Composition

Suppose $f(y_1, \ldots, y_k)$ is a $k$-ary function and $g_1(\widetilde{x}), \ldots, g_k(\widetilde{x})$ are $n$-ary functions, where $\widetilde{x}$ abbreviates $x_1, \ldots, x_n$.

The composition function $h(\widetilde{x})$ is defined by

$$h(\widetilde{x}) \quad = \quad f(g_1(\widetilde{x}), \ldots, g_k(\widetilde{x})),$$

# Recursion

Suppose that $f(\widetilde{x})$ is an $n$-ary function and $g(\widetilde{x}, y, z)$ is an $(n+2)$-ary function.

The recursion function $h(\widetilde{x}, y)$ is defined by

$$h(\widetilde{x}, 0) = f(\widetilde{x}), \tag{1}$$

$$h(\widetilde{x}, y+1) = g(\widetilde{x}, y, h(\widetilde{x}, y)). \tag{2}$$

Clearly there is a unique function that satisfies (1) and (2).

# Primitive Recursive Recursion

The set of primitive recursive function is the least set generated from the initial functions, composition and recursion.

# Dummy Parameter

## Proposition

Suppose that $f(y_1, \ldots, y_k)$ is a primitive recursive and that $x_{i_1}, \ldots, x_{i_k}$ is a sequence of $k$ variables from $x_1, \ldots, x_n$ (possibly with repetition). Then the function $h$ given by

$$h(x_1, \ldots, x_n) \quad = \quad f(x_{i_1}, \ldots, x_{i_k})$$

is primitive recursive.

## Proof

$h(\widetilde{x}) = f(\mathsf{U}^n_{i_1}(\widetilde{x}), \ldots, \mathsf{U}^n_{i_k}(\widetilde{x}))$.

# Basic Arithmetic Function

# Basic Arithmetic Function

- $x + y$
-
$$\begin{aligned} x + 0 &= x, \\ x + (y + 1) &= s(x + y). \end{aligned}$$
- $xy$
-
$$\begin{aligned} x0 &= 0, \\ x(y + 1) &= xy + x. \end{aligned}$$
- $x^y$
-
$$\begin{aligned} x^0 &= 1, \\ x^{y+1} &= x^y x. \end{aligned}$$

# Quiz

$x + y + z$

# Basic Arithmetic Function

- $x \dot{-} 1$
-

$$\begin{aligned} 0 \dot{-} 1 &= 0, \\ (x+1) \dot{-} 1 &= x. \end{aligned}$$

- $x \dot{-} y \stackrel{\text{def}}{=} \begin{cases} x - y, & \text{if } x \geq y, \\ 0, & \text{otherwise.} \end{cases}$
-

$$\begin{aligned} x \dot{-} 0 &= x, \\ x \dot{-} (y+1) &= (x \dot{-} y) \dot{-} 1. \end{aligned}$$

# Basic Arithmetic Function

- $\mathsf{sg}(x) \stackrel{\text{def}}{=} \begin{cases} 0, & \text{if } x = 0, \\ 1, & \text{if } x \neq 0. \end{cases}$
-

$$\begin{aligned} \mathsf{sg}(0) &= 0, \\ \mathsf{sg}(x+1) &= 1. \end{aligned}$$

- $\overline{\mathsf{sg}}(x) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } x = 0, \\ 0, & \text{if } x \neq 0. \end{cases}$
-

$$\overline{\mathsf{sg}}(x) = 1 \dot{-} \mathsf{sg}(x).$$

# Basic Arithmetic Function

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

$$\begin{aligned} 0! &= 1, \\ (x+1)! &= x!(x+1). \end{aligned}$$

- $\min(x, y)$
- $\min(x, y) = x \dot{-} (x \dot{-} y).$
- $\max(x, y)$
- $\max(x, y) = x + (y \dot{-} x).$

# Basic Arithmetic Function

$\mathsf{rm}(x,y) \stackrel{\text{def}}{=}$ the remainder when $y$ is devided by $x$

$$\mathsf{rm}(x,y+1) \stackrel{\text{def}}{=} \begin{cases} \mathsf{rm}(x,y)+1 & \text{if } \mathsf{rm}(x,y)+1 < x, \\ 0, & \text{otherwise.} \end{cases}$$

The recursive definition is given by

$$\begin{aligned} \mathsf{rm}(x,0) &= 0, \\ \mathsf{rm}(x,y+1) &= (\mathsf{rm}(x,y)+1)\mathrm{sg}(x \dot{-} (\mathsf{rm}(x,y)+1)). \end{aligned}$$

# Basic Arithmetic Function

$\mathsf{qt}(x,y) \stackrel{\text{def}}{=}$ the quotient when $y$ is devided by $x$

$$\mathsf{qt}(x,y+1) \stackrel{\text{def}}{=} \begin{cases} \mathsf{qt}(x,y)+1, & \text{if } \mathsf{rm}(x,y)+1 = x, \\ \mathsf{qt}(x,y), & \text{if } \mathsf{rm}(x,y)+1 \neq x. \end{cases}$$

The recursive definition is given by

$$\begin{aligned} \mathsf{qt}(x,0) &= 0, \\ \mathsf{qt}(x,y+1), &= \mathsf{qt}(x,y) + \overline{\mathrm{sg}}(x - (\mathsf{rm}(x,y)+1)). \end{aligned}$$

# Basic Arithmetic Function

$$\mathsf{div}(x,y) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } x \text{ divides } y, \\ 0, & \text{otherwise.} \end{cases}$$

$$\mathsf{div}(x,y) = \overline{\mathrm{sg}}(\mathsf{rm}(x,y)).$$

# Bounded Minimalisation Operator

# Bounded Sum and Bounded Product

Bounded sum:

$$\begin{aligned} \sum_{y<0} f(\widetilde{x}, y) &= 0, \\ \sum_{y<z+1} f(\widetilde{x}, y) &= \sum_{y<z} f(\widetilde{x}, y) + f(\widetilde{x}, z). \end{aligned}$$

Bounded product:

$$\begin{aligned} \prod_{y<0} f(\widetilde{x}, y) &= 1, \\ \prod_{y<z+1} f(\widetilde{x}, y) &= \left(\prod_{y<z} f(\widetilde{x}, y)\right) \cdot f(\widetilde{x}, z). \end{aligned}$$

# Bounded Sum and Bounded Product

By composition the following functions are also primitive recursive if $k(\widetilde{x}, \widetilde{w})$ is primitive recursive:

$$\sum_{z<k(\widetilde{x},\widetilde{w})} f(\widetilde{x}, z)$$

and

$$\prod_{z<k(\widetilde{x},\widetilde{w})} f(\widetilde{x}, z).$$

# Bounded Minimization Operator

Bounded search:

$$\mu z{<}y(f(\widetilde{x},z)=0) \stackrel{\text{def}}{=} \begin{cases} \text{the least } z<y, & \text{such that } f(\widetilde{x},z)=0; \\ y, & \text{if there is no such } z. \end{cases}$$

**Proposition**

If $f(\widetilde{x},z)$ is primitive recursive, then so is $\mu z{<}y(f(\widetilde{x},z)=0)$

**Proof**

$\mu z{<}y(f(\widetilde{x},z)=0) = \sum_{v<y}(\prod_{u<v+1} \mathsf{sg}(f(\widetilde{x},u)))$

# Bounded Minimization Operator

If $f(\widetilde{x}, z)$ and $k(\widetilde{x}, \widetilde{w})$ are primitive recursive functions, then so is the function

$$\mu z{<}k(\widetilde{x}, \widetilde{w})(f(\widetilde{x}, z) = 0).$$

# Primitive Recursive Predicate

# Primitive Recursive Predicate

Suppose $M(x_1, \ldots, x_n)$ is an $n$-ary predicate of natural numbers. The characteristic function $c_M(\widetilde{x})$, where $\widetilde{x} = x_1, \ldots, x_n$, is

$$c_M(a_1, \ldots, a_n) = \begin{cases} 1, & \text{if } M(a_1, \ldots, a_n) \text{ holds,} \\ 0, & \text{if otherwise.} \end{cases}$$

The predicate $M(\widetilde{x})$ is primitive recursive if $c_M$ is primitive recursive.

# Closure Property

## Proposition

The following statements are valid:

- If $R(\widetilde{x})$ is a primitive recursive predicate, then so is $\neg R(\widetilde{x})$.
- If $R(\widetilde{x})$, $S(\widetilde{x})$ are primitive recursive predicates, then the following predicates are primitive recursive:
  - $R(\widetilde{x}) \wedge S(\widetilde{x})$;
  - $R(\widetilde{x}) \vee S(\widetilde{x})$.
- If $R(\widetilde{x}, y)$ is a primitive recursive predicate, then the following predicates are primitive recursive:
  - $\forall z < y.R(\widetilde{x}, z)$;
  - $\exists z < y.R(\widetilde{x}, z)$.

## Proof

For example $c_{\forall z<y.R(\widetilde{x},z)}(\widetilde{x}, y) = \prod_{z<y} c_R(\widetilde{x}, z)$.

# Definition by Case

## Proposition

Suppose that $f_1(\widetilde{x}), \ldots, f_k(\widetilde{x})$ are primitive recursive functions, and $M_1(\widetilde{x}), \ldots, M_k(\widetilde{x})$ are primitive recursive predicates, such that for every $\widetilde{x}$ exactly one of $M_1(\widetilde{x}), \ldots, M_k(\widetilde{x})$ holds. Then the function $g(\widetilde{x})$ given by

$$g(\widetilde{x}) = \begin{cases} f_1(\widetilde{x}), & \text{if } M_1(\widetilde{x}) \text{ holds,} \\ f_2(\widetilde{x}), & \text{if } M_2(\widetilde{x}) \text{ holds,} \\ \vdots & \\ f_k(\widetilde{x}), & \text{if } M_k(\widetilde{x}) \text{ holds.} \end{cases}$$

is primitive recursive.

## Proof

$g(\widetilde{x}) = c_{M_1}(\widetilde{x}) f_1(\widetilde{x}) + \ldots + c_{M_k}(\widetilde{x}) f_k(\widetilde{x})$

# More Arithmetic Functions

# More Arithmetic Functions

The following functions are primitive recursive.

1. $D(x) =$ the number of divisors of $x$;
2. $Pr(x) = \begin{cases} 1, & \text{if } x \text{ is prime,} \\ 0, & \text{if } x \text{ is not prime.} \end{cases}$
3. $\mathsf{p}_x =$ the $x$-th prime number;
4. $(x)_y = \begin{cases} k, & k \text{ is the exponent of } p_y \text{ in the prime} \\ & \text{factorisation of } x, \text{ for } x, y > 0, \\ 0, & \text{if } x = 0 \text{ or } y = 0. \end{cases}$

# More Arithmetic Functions

Proof

1. $D(x) = \sum_{y<x+1} \mathrm{div}(y, x)$.
2. $Pr(x) = \overline{\mathsf{sg}}(|D(x) - 2|)$.
3. $\mathsf{p}_x$ can be recursively defined as follows:

$$\begin{aligned} \mathsf{p}_0 &= 0, \\ \mathsf{p}_{x+1} &= \mu z < (1 + \mathsf{p}_x!)\left(1 \dot{-} (z \dot{-} \mathsf{p}_x) Pr(z) = 0\right). \end{aligned}$$

4. $(x)_y = \mu z{<}x(\mathrm{div}(\mathsf{p}_y^{z+1}, x) = 0)$.

# Encoding a Finite Sequence

Suppose $s = (a_1, a_2, \ldots, a_n)$ is a finite sequence of numbers. It can be coded by the following number

$$b = \mathsf{p}_1^{a_1+1} \mathsf{p}_2^{a_2+1} \ldots \mathsf{p}_n^{a_n+1}.$$

Then the length of $s$ can be recovered from

$$\mu z{<}b((b)_{z+1} = 0),$$

and the $i$-th component can be recovered from

$$(b)_i \dot{-} 1.$$

# Not all Computable Functions are Primitive Recursive

Using the fact that all primitive recursive functions are total, a diagonalisation argument shows that non-primitive recursive computable functions must exist.

The same diagonalisation argument applies to all finite axiomatizations of computable total function.

Onward to the partial functions!

# Computability Theory IV

## Recursive Function

Guoqiang Li

Shanghai Jiao Tong University

Oct. 17, 2014

# Review Tips

# Initial Function

1. The zero function
   - $\mathbf{0}$
   - $\mathbf{0}(\widetilde{x}) = 0$
2. The successor function
   - $s(x) = x + 1$
3. The projection function
   - $U_i^n(x_1, \ldots, x_n) = x_i$

# Composition

Suppose $f(y_1, \ldots, y_k)$ is a $k$-ary function and $g_1(\widetilde{x}), \ldots, g_k(\widetilde{x})$ are $n$-ary functions, where $\widetilde{x}$ abbreviates $x_1, \ldots, x_n$.

The composition function $h(\widetilde{x})$ is defined by

$$h(\widetilde{x}) \quad = \quad f(g_1(\widetilde{x}), \ldots, g_k(\widetilde{x})),$$

# Recursion

Suppose that $f(\widetilde{x})$ is an $n$-ary function and $g(\widetilde{x}, y, z)$ is an $(n+2)$-ary function.

The recursion function $h(\widetilde{x}, y)$ is defined by

$$h(\widetilde{x}, 0) = f(\widetilde{x}), \tag{1}$$

$$h(\widetilde{x}, y+1) = g(\widetilde{x}, y, h(\widetilde{x}, y)). \tag{2}$$

Clearly there is a unique function that satisfies (1) and (2).

# Quiz

$$LCM(x,y)$$

Sol. $LCM(x,y) = \mu z < xy + 1(div(x,z)div(y,z) = 1)$.

$$HCF(x,y))$$

Sol. $HCF(x,y) = \frac{xy}{LCM(x,y)}$.

# Synopsis

1. Recursive Function
2. Ackermann Function
3. Definability in URM

# Recursive Function

# An Example

$$g(x) = \begin{cases} \sqrt{x} & \text{if } x \text{ is a pefect square.} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

# Minimization Operator, or Search Operator

Minimization function, or $\mu$-function, or search function:

$$\mu y(f(\widetilde{x}, y) = 0) \quad \simeq \quad \begin{cases} \text{the least } y \text{ such that} \\ \quad f(\widetilde{x}, z) \text{ is defined for all } z \leq y, \text{ and} \\ \quad f(\widetilde{x}, y) = 0, \\ \text{undefined if otherwise.} \end{cases}$$

Here $\simeq$ is the computational equality.

- The recursion operation is a well-founded going-down procedure.
- The search operation is a possibly divergent going-up procedure.

# An Example

$$g(x) = \begin{cases} \sqrt{x} & \text{if } x \text{ is a pefect square.} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

$$f(x,y) = |x - y^2|$$

# Recursive Function

The set of recursive functions is the least set generated from the initial functions, composition, recursion and minimization.

# Decidable Predicate

A predicate $R(\widetilde{x})$ is decidable if its characteristic function

$$c_R(\widetilde{x}) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } R(\widetilde{x}) \text{ is true,} \\ 0, & \text{otherwise.} \end{cases}$$

is a recursive function. The predicate $R(\widetilde{x})$ is partially decidable if its partial characteristic function

$$\chi_R(\widetilde{x}) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } R(\widetilde{x}) \text{ is true,} \\ \uparrow, & \text{otherwise.} \end{cases}$$

is a recursive function.

# Closure Property

The following statements are valid:

- If $R(\widetilde{x})$ is decidable, then so is $\neg R(\widetilde{x})$.
- If $R(\widetilde{x}), S(\widetilde{x})$ are (partially) decidable, then the following predicates are (partially) decidable:
  - $R(\widetilde{x}) \wedge S(\widetilde{x})$;
  - $R(\widetilde{x}) \vee S(\widetilde{x})$.
- If $R(\widetilde{x}, y)$ is (partially) decidable, then the following predicates are (partially) decidable:
  - $\forall z < y . R(\widetilde{x}, y)$;
  - $\exists z < y . R(\widetilde{x}, y)$.

# Definition by Cases

Suppose $f_1(\widetilde{x}), \ldots, f_k(\widetilde{x})$ are recursive and $M_1(\widetilde{x}), \ldots, M_k(\widetilde{x})$ are partially decidable. For every $\widetilde{x}$ at most one of $M_1(\widetilde{x}), \ldots, M_k(\widetilde{x})$ holds. Then the function $g(\widetilde{x})$ given by

$$g(\widetilde{x}) \simeq \begin{cases} f_1(\widetilde{x}), & \text{if } M_1(\widetilde{x}) \text{ holds,} \\ f_2(\widetilde{x}), & \text{if } M_2(\widetilde{x}) \text{ holds,} \\ \vdots & \\ f_k(\widetilde{x}), & \text{if } M_k(\widetilde{x}) \text{ holds.} \end{cases}$$

is recursive.

# Minimization via Decidable Predicate

Suppose $R(x, y)$ is a partially decidable predicate. The function

$$
\begin{aligned}
g(x) &= \mu y R(\widetilde{x}, y) \\
&= \begin{cases} \text{the least } y \text{ such that } R(\widetilde{x}, y) \text{ holds}, & \text{if there is such a } y \\ \text{undefined}, & \text{otherwise.} \end{cases}
\end{aligned}
$$

is recursive.

Proof

$g(\widetilde{x}) = \mu y(\overline{\mathsf{sg}}(\chi_R(\widetilde{x}, y)) = 0)$.

# Comment

The $\mu$-operator allows one to define partial functions.

The diagonalisation argument does not apply to the set $\mathfrak{F}$ of recursive functions.

Using the $\mu$-operator, one may define total functions that are not primitive recursive.

# Minimization Operator is a Search Operator

It is clear from the above proof why the minimization operator is sometimes called a search operator.

# Definable Function

A function is definable if there is a recursive function calculating it.

Ackermann function

# Ackermann Function

The Ackermann function [1928] is defined as follows:

$$\begin{aligned} \psi(0, y) &\simeq y + 1, \\ \psi(x + 1, 0) &\simeq \psi(x, 1), \\ \psi(x + 1, y + 1) &\simeq \psi(x, \psi(x + 1, y)). \end{aligned}$$

The equations clearly define a total function.

# Ackermann is not Primitive Recursive

Lemma 1.
$\psi(1, m) = m + 2$ and $\psi(2, m) = 2m + 3$

Lemma 2.
$\psi(n, m) \geq m + 1$

# Ackermann is not Primitive Recursive

Lemma 3.

The Ackermann function is monotone:

$$\begin{aligned} \psi(n,m) &< \psi(n,m+1), \\ \psi(n,m) &< \psi(n+1,m). \end{aligned}$$

Lemma 4.

The Ackermann function grows faster on the first parameter:

$$\psi(n,m+1) \leq \psi(n+1,m)$$

# Ackermann is not Primitive Recursive

Lemma 5.
$\psi(n,m)+C$ is dominated by $\psi(J,m)$ for some large enough $J$:

$$\begin{aligned} \psi(n,m)+\psi(n',m) &< \psi(\max(n,n')+4,m), \\ \psi(n,m)+m &< \psi(n+4,m). \end{aligned}$$

# Ackermann is not Primitive Recursive

Lemma 6.
Let $f(\widetilde{x})$ be a $k$-ary primitive recursive function. Then there exists some $J$ such that for all $n_1, \ldots, n_k$ we have that

$$f(n_1, \ldots, n_k) < \psi(J, \sum_{i=1}^{k} n_k).$$

*Proof.* The proof is by structural induction.

(i) $f$ is one of the initial functions. In this case take $J$ to be 1.

# Ackermann is not Primitive Recursive

(ii) $f$ is the composition function $h(g_1(\widetilde{x}), \ldots, g_m(\widetilde{x}))$. Then

$$\begin{aligned}
f(\widetilde{n}) &= h(g_1(\widetilde{n}), \ldots, g_m(\widetilde{n})) \\
&< \psi(J_0, \sum_{i=1}^{m} g_i(\widetilde{n})) < \psi(J_0, \sum_{i=1}^{m} \psi(J_i, \sum_{j=1}^{k} n_j)) \\
&< \psi(J_0, \psi(J^*, \sum_{j=1}^{k} n_j)) < \psi(J^*, \psi(J^*+1, \sum_{j=1}^{k} n_j)) \\
&= \psi(J^*+1, \sum_{j=1}^{k} n_j + 1) \leq \psi(J^*+2, \sum_{j=1}^{k} n_j).
\end{aligned}$$

Now set $J = J^* + 2$.

# Ackermann is not Primitive Recursive

(iii) Suppose $f$ is defined by the recursion:

$$\begin{aligned} f(\widetilde{x},0) &\simeq h(\widetilde{x}), \\ f(\widetilde{x},y+1) &\simeq g(\widetilde{x},y,f(\widetilde{x},y)). \end{aligned}$$

Then $h(\widetilde{n}) < \psi(J_h, \sum \widetilde{n})$ and $g(\widetilde{n},m,p) < \psi(J_g, \sum \widetilde{n} + m + p)$.

It is easy to prove

$$f(n_1,\ldots,n_k,m) < \psi(J, \sum_{i=1}^{k} n_k + m)$$

by induction on $m$.

# Ackermann is not Primitive Recursive

Now suppose $\psi(x, y)$ was primitive recursive.
By composition $\psi(x, x)$ would be primitive recursive.

According to the Lemma 6

$$\psi(n, n) < \psi(J, n)$$

for some $J$ and all $n$, which would lead to the contradiction

$$\psi(J, J) < \psi(J, J).$$

# Ackermann is not Primitive Recursive

Theorem

The Ackermann function grows faster than every primitive recursive function.

# Ackermann Function is Recursive

Theorem

The Ackermann function is recursive.

# Ackermann Function is Recursive

A finite set $S$ of triples is said to be suitable if the followings hold:
(i) if $(0, y, z) \in S$ then $z = y + 1$;
(ii) if $(x + 1, 0, z) \in S$ then $(x, 1, z) \in S$;
(iii) if $(x + 1, y + 1, z) \in S$ then $\exists u.((x + 1, y, u) \in S \wedge (x, u, z) \in S)$.

A triple $(x, y, z)$ can be coded up by $2^x 3^y 5^z$.
A set $\{u_1, \ldots, u_k\}$ can be coded up by $p_{u_1} \cdots p_{u_k}$.

Let $R(x, y, v)$ be "$v$ is a legal code and $\exists z < v.(x, y, z) \in S_v$".

The Ackermann function $\psi(x, y) \simeq \mu z((x, y, z) \in S_{\mu v R(x,y,v)})$.

# Definability in URM

# Definability of Initial Function

**Fact**. The initial functions are URM-definable.

# Definability of Composition

**Fact**. If $f(y_1, \ldots, y_k)$ and $g_1(\widetilde{x}), \ldots, g_k(\widetilde{x})$ are URM-definable, then the composition function $h(\widetilde{x})$ given by

$$h(\widetilde{x}) \quad \simeq \quad f(g_1(\widetilde{x}), \ldots, g_k(\widetilde{x}))$$

is URM-definable.

# Some Notations

Suppose the program $P$ computes $f$.

Let $\rho(P)$ be the least number $i$ such that the register $R_i$ is not used by the program $P$.

# Some Notations

The notation $P[l_1, \ldots, l_n \to l]$ stands for the following program

$$
\begin{array}{rcl}
I_1 & : & T(l_1, 1) \\
 & \vdots & \\
I_n & : & T(l_n, n) \\
I_{n+1} & : & Z(n+1) \\
 & \vdots & \\
I_{\rho(P)} & : & Z(\rho(P)) \\
\_ & : & P \\
\_ & : & T(1, l)
\end{array}
$$

# Definability of Composition

Let $F, G_1, \ldots, G_k$ be programs that compute $f, g_1, \ldots, g_k$.

Let $m$ be $\max\{n, k, \rho(F), \rho(G_1), \ldots, \rho(G_k)\}$.

Registers:

$$[\ldots]_1^m [\widetilde{x}]_{m+1}^{m+n} [g_1(\widetilde{x})]_{m+n+1}^{m+n+1} \cdots [g_k(\widetilde{x})]_{m+n+k}^{m+n+k}$$

# Definability of Composition

The program for $h$:

$$
\begin{array}{rcl}
I_1 & : & T(1, m+1) \\
 & \vdots & \\
I_n & : & T(n, m+n) \\
I_{n+1} & : & G_1[m+1, m+2, \ldots, m+n \ \rightarrow \ m+n+1] \\
 & \vdots & \\
I_{n+k} & : & G_k[m+1, m+2, \ldots, m+n \ \rightarrow \ m+n+k] \\
I_{n+k+1} & : & F[m+n+1 \ldots, m+n+k \ \rightarrow \ 1]
\end{array}
$$

# Definability of Recursion

**Fact**. Suppose $f(\widetilde{x})$ and $g(\widetilde{x}, y, z)$ are URM-definable.
The recursion function $h(\widetilde{x}, y)$ defined by the following recursion

$$\begin{aligned} h(\widetilde{x}, 0) &\simeq f(\widetilde{x}), \\ h(\widetilde{x}, y+1) &\simeq g(\widetilde{x}, y, h(\widetilde{x}, y)) \end{aligned}$$

is URM-definable.

# Definability of Recursion

Let $F$ compute $f$ and $G$ compute $g$. Let $m$ be $\max\{n, \rho(F), \rho(G)\}$.

Registers: $[\ldots]_1^m [\widetilde{x}]_{m+1}^{m+n} [y]_{m+n+1}^{m+n+1} [k]_{m+n+2}^{m+n+2} [h(\widetilde{x}, k)]_{m+n+3}^{m+n+3}$.

Program:

$$
\begin{array}{rcl}
I_1 & : & T(1, m+1) \\
 & & \vdots \\
I_{n+1} & : & T(n+1, m+n+1) \\
I_{n+2} & : & F[1, 2, \ldots, n \ \rightarrow \ m+n+3] \\
I_{n+3} & : & J(m+n+2, m+n+1, n+7) \\
I_{n+4} & : & G[m+1, \ldots, m+n, m+n+2, m+n+3 \ \rightarrow \ m+n+3] \\
I_{n+5} & : & S(m+n+2) \\
I_{n+6} & : & J(1, 1, n+3) \\
I_{n+7} & : & T(m+n+3, 1)
\end{array}
$$

# Definability of Minimization

**Fact**. If $f(\widetilde{x}, y)$ is URM-definable, then the minimization function $\mu y(f(\widetilde{x}, y) = 0)$ is URM-definable.

# Definability of Minimization

Suppose $F$ computes $f(\widetilde{x}, y)$. Let $m$ be $\max\{n+1, \rho(F)\}$.

Registers: $[\ldots]_1^m [\widetilde{x}]_{m+1}^{m+n} [k]_{m+n+1}^{m+n+1} [0]_{m+n+2}^{m+n+2}$.

Program:

$$
\begin{array}{rcl}
I_1 & : & T(1, m+1) \\
 & & \vdots \\
I_n & : & T(n, m+n) \\
I_{n+1} & : & F[m+1, m+2, \ldots, m+n+1 \ \rightarrow \ 1] \\
I_{n+2} & : & J(1, m+n+2, n+5) \\
I_{n+3} & : & S(m+n+1) \\
I_{n+4} & : & J(1, 1, n+1) \\
I_{n+5} & : & T(m+n+1, 1)
\end{array}
$$

# Main Result

**Theorem**. All recursive functions are URM-definable.

# Homework

- Read the proof that Ackermann function is not primitive.
- Try to solve the exercises in Chapter 1 & 2 as many as possible.

# Computability Theory V

## Turing Machine

Guoqiang Li

Shanghai Jiao Tong University

Oct. 24, 2014

*Assignment 2 is announced! (deadline Nov. 7)*

Turing Machine

# Alan Turing

**Alan Turing** (23Jun.1912-7Jun.1954), an English student of Church, introduced a machine model for effective calculation in

"On Computable Numbers, with an Application to the Entsheidungs problem",

Proc. of the London Mathematical Society, **42**:230-265, 1936.

Turing Machine, Halting Problem, Turing Test

British Prime Minister Gordon Brown:

"…I am pleased to have the chance to say how deeply sorry I and we all are for what happened to him ... So on behalf of the British government, and all those who live freely thanks to Alan's work, I am very proud to say: we're sorry, you deserved so much better."

# Motivation

What are necessary for a machine to calculate a function?

- The machine should be able to interpret numbers;
- The machine must be able to operate and manipulate numbers according to a set of predefined instructions;

and

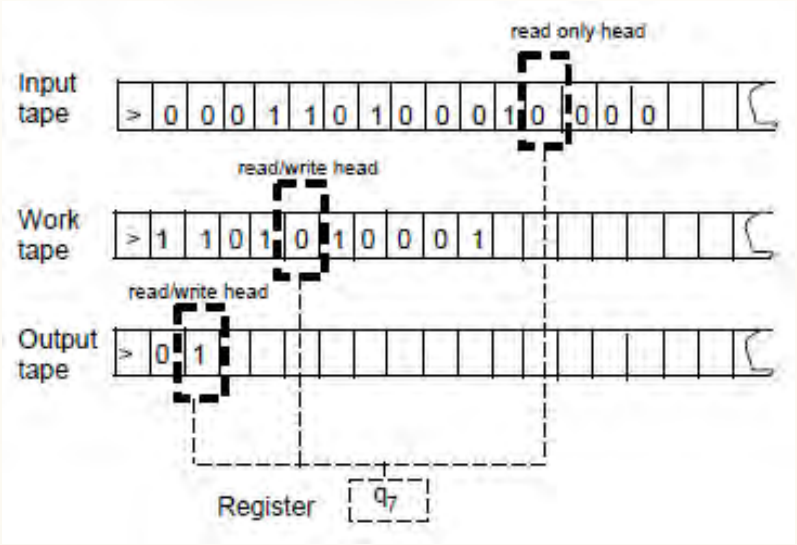
- The input number has to be stored in an accessible place;
- There should be an accessible place for the machine to store the intermediate results;
- The output number has to be put in an accessible place.

# Turing Machine

A $k$-tape Turing Machine $\mathbb{M}$ has $k$-tapes such that

- The first tape is the read-only input tape.
- The other $k-1$ tapes are the read/write work tapes.
- The $k$-th tape is also used as the output tape.

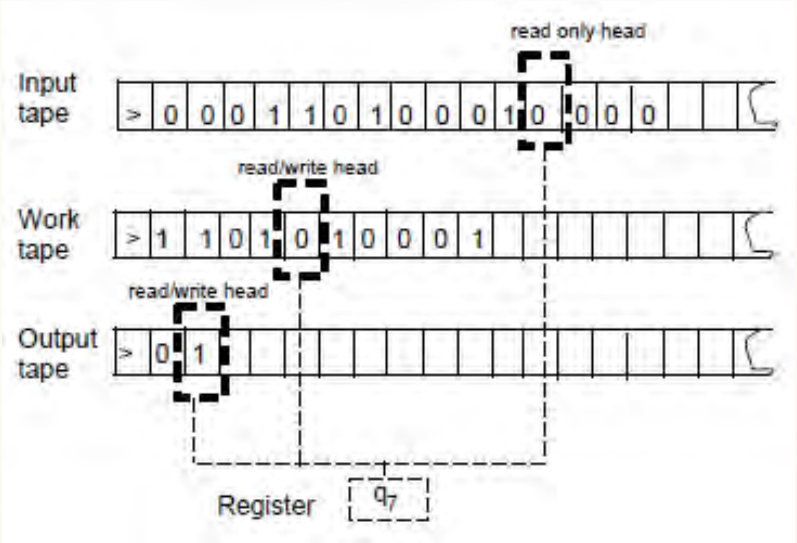
Every tape comes with a read/write head.



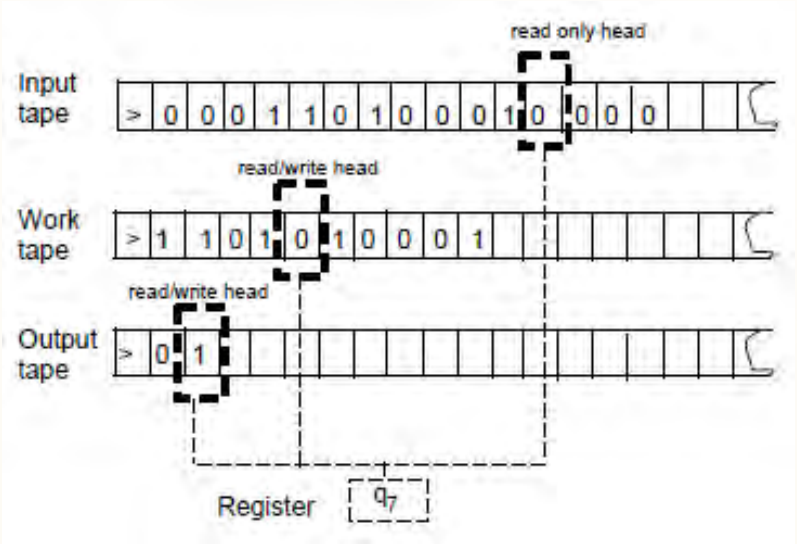
# Turing Machine

The machine is described by a tuple $(\Gamma, Q, \delta)$ containing

- A finite set $\Gamma$, called alphabet, of symbols. It contains a blank symbol $\square$, a start symbol $\triangleright$, and the digits $0$ and $1$.
- A finite set $Q$ of states. It contains a start state $q_{\mathrm{s}}$ and a halting state $q_{\mathrm{h}}$.
- A transition function $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\leftarrow, -, \rightarrow\}^k$, describing the rules of each computation step.



# Computation and Configuration




Configuration, initial configuration, final configuration, computation step

# An Example

$Q = \{q_s, q_h, q_1, q_2, q_3\}$, $\Gamma = \{0, 1, \square, \triangleright\}$,
and $\delta$ is as follows:

| $p \in Q$ | $\sigma \in \Gamma$ | $\delta(p, \sigma)$ |
|---|---|---|
| $q_s$ | $0$ | $(q_s, 0, \rightarrow)$ |
| $q_s$ | $1$ | $(q_s, 1, \rightarrow)$ |
| $q_s$ | $\square$ | $(q_1, \square, \leftarrow)$ |
| $q_s$ | $\triangleright$ | $(q_s, \triangleright, \rightarrow)$ |
| $q_1$ | $0$ | $(q_2, \square, \rightarrow)$ |
| $q_1$ | $1$ | $(q_3, \square, \rightarrow)$ |
| $q_1$ | $\square$ | $(q_1, \square, -)$ |
| $q_1$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_2$ | $0$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $1$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\square$ | $(q_s, 0, \leftarrow)$ |
| $q_2$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |
| $q_3$ | $0$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $1$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\square$ | $(q_s, 1, \leftarrow)$ |
| $q_3$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |

Start the machine with input 010

- $q_s, \underline{\triangleright}010$
- $q_s, \triangleright\underline{0}10$
- $q_s, \triangleright0\underline{1}0$
- $q_s, \triangleright01\underline{0}$
- $q_s, \triangleright010\underline{\square}$
- $q_1, \triangleright01\underline{0}\square$
- $q_2, \triangleright01\square\underline{\square}$
- $q_s, \triangleright01\underline{\square}0$
- $q_1, \triangleright0\underline{1}\square0$
- $q_3, \triangleright0\square\underline{\square}0$
- $q_s, \triangleright0\underline{\square}10$
- $q_1, \triangleright\underline{0}\square10$
- $q_2, \triangleright\square\underline{\square}10$
- $q_0, \triangleright\underline{\square}010$
- $q_1, \underline{\triangleright}\square010$
- $q_h, \triangleright\underline{\square}010$

# The Second Example

$Q = \{q_s, q_h, q_1\}$, $\Gamma = \{0, 1, \square, \triangleright\}$, and $\delta$ is as follows:

| $p \in Q$ | $\sigma \in \Gamma$ | $\delta(p, \sigma)$ |
|---|---|---|
| $q_s$ | $0$ | $(q_s, 0, \rightarrow)$ |
| $q_s$ | $1$ | $(q_s, 1, \rightarrow)$ |
| $q_s$ | $\square$ | $(q_1, \square, \leftarrow)$ |
| $q_s$ | $\triangleright$ | $(q_s, \triangleright, \rightarrow)$ |
| $q_1$ | $0$ | $(q_h, 1, -)$ |
| $q_1$ | $1$ | $(q_1, 0, \leftarrow)$ |
| $q_1$ | $\triangleright$ | $(q_h, \triangleright, \rightarrow)$ |

# The Third Example

$Q = \{q_s, q_h, q_c, q_l, q_t\}$; $\Gamma = \{\square, \triangleright, 0, 1\}$; two work tapes.

$\langle q_s, \triangleright, \triangleright, \triangleright\rangle \to \langle q_c, \triangleright, \triangleright, \to, \to, \to\rangle$

$\langle q_c, 0, \square, \square\rangle \to \langle q_c, 0, \square, \to, \to, -\rangle$
$\langle q_c, 1, \square, \square\rangle \to \langle q_c, 1, \square, \to, \to, -\rangle$
$\langle q_c, \square, \square, \square\rangle \to \langle q_l, \square, \square, \leftarrow, -, -\rangle$

$\langle q_l, 0, \square, \square\rangle \to \langle q_l, \square, \square, \leftarrow, -, -\rangle$
$\langle q_l, 1, \square, \square\rangle \to \langle q_l, \square, \square, \leftarrow, -, -\rangle$
$\langle q_l, \triangleright, \square, \square\rangle \to \langle q_t, \square, \square, \to, \leftarrow, -\rangle$

$\langle q_t, \square, \triangleright, \square\rangle \to \langle q_h, \triangleright, 1, -, -, -\rangle$
$\langle q_t, 0, 1, \square\rangle \to \langle q_h, 1, 0, -, -, -\rangle$
$\langle q_t, 1, 0, \square\rangle \to \langle q_h, 0, 0, -, -, -\rangle$
$\langle q_t, 0, 0, \square\rangle \to \langle q_t, 0, \square, \to, \leftarrow, -\rangle$
$\langle q_t, 1, 1, \square\rangle \to \langle q_t, 1, \square, \to, \leftarrow, -\rangle$

# $\{0, 1, \square, \triangleright\}$ vs. Larger Alphabets

Suppose $\mathbb{M}$ has $k$ tapes with the alphabet $\Gamma$.

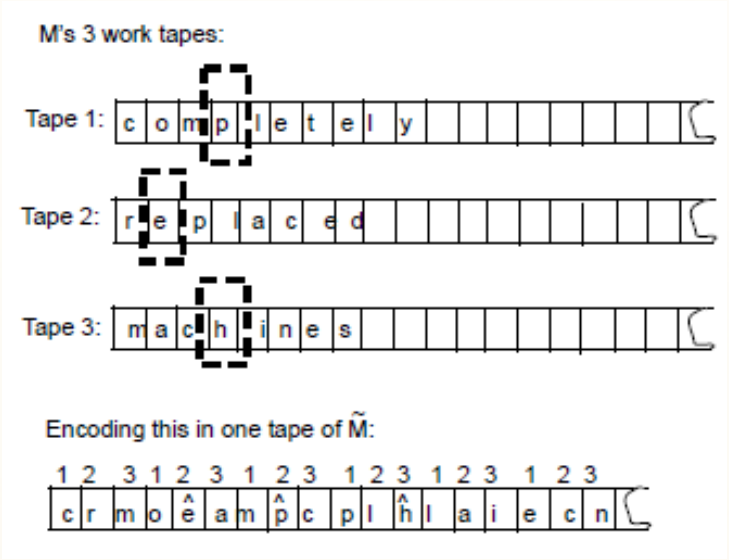
A symbol of $\mathbb{M}$ is encoded by a string $\sigma \in \{0,1\}^*$ of length $\log|\Gamma|$.

States: A state $q$ is turned into states $q, \langle q, \sigma_1^1, \ldots, \sigma_1^k \rangle$ where $|\sigma_1^1| = \ldots = |\sigma_1^k| = 1$, ..., $\langle q, \sigma_{\log|\Gamma|}^1, \ldots, \sigma_{\log|\Gamma|}^k \rangle$ where $|\sigma_{\log|\Gamma|}^1| = \ldots = |\sigma_{\log|\Gamma|}^k| = \log|\Gamma|$.

A computation step of $\mathbb{M}$ is simulated in $\widetilde{\mathbb{M}}$ by $\log|\Gamma|$ steps to read, $\log|\Gamma|$ steps to write, and $\log|\Gamma|$ steps to relocate the heads.

# One Tape vs. Many Tapes

The basic idea is to interleave $k$ tapes into one tape.
The first $n + 1$ cells are reserved for the input.




Every symbol $a$ of $\mathbb{M}$ is turned into two symbols $a, \widehat{a}$ in $\widetilde{\mathbb{M}}$, with $\widehat{a}$ used to indicate head position.

# One Tape vs. Many Tapes

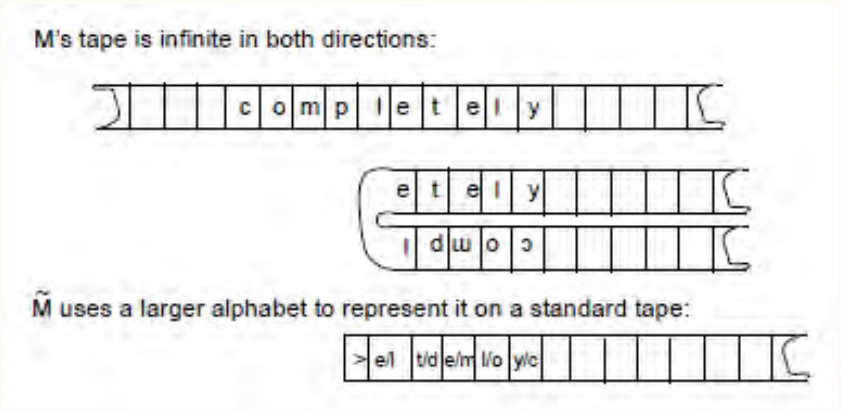
The machine $\widetilde{\mathbb{M}}$ copies the input bits to the first imaginary tape. The head then moves left to the $(n+2)$-th cell.

Sweeping the tape cells from left to right. Record in the register the $k$ symbols marked with the hat $\widehat{\_}$.

Sweeping the tape cells from right to left to update using the transitions of $\mathbb{M}$.

# One Unidirectional vs. Bidirectional Tape

The idea is that $\widetilde{\mathbb{M}}$ makes use of the alphabet $\Gamma \times \Gamma$.




Every state $q$ of $\mathbb{M}$ is turned into $\overline{q}$ and $\underline{q}$.

# Simulation of TM by URM

# Simulating TM by URM

Suppose $\mathbb{M}$ is a 3-tape TM with the alphabet $\{0, 1, \square, \triangleright\}$.

The URM that simulates $\mathbb{M}$ can be designed as follows:

- Suppose that $R_m$ is the right most register that is used by a program calculating $x \dot{-} 1$.
- The head positions are stored in $R_{m+1}, R_{m+2}, R_{m+3}$.
- The three binary strings in the tapes are stored respectively in
  $R_{m+4}, R_{m+7}, R_{m+10}, \ldots$,
  $R_{m+5}, R_{m+8}, R_{m+11}, \ldots$,
  $R_{m+6}, R_{m+9}, R_{m+12}, \ldots$.
- The states of $\mathbb{M}$ are encoded by the states of the URM.
- The transition function of $\mathbb{M}$ can be easily simulated by the program of the URM.

# Homework

Encode the addition function by $k$-tape Turing machine two nature number partitioned by $\sharp$ on the input tape, for example, 11010$\#$1001, and then try to encode the function by 1-tape Turing machine.

# Computability Theory VI

## Church-Turing Thesis

Guoqiang Li

Shanghai Jiao Tong University

Oct. 31, 2014

# Church-Turing Thesis

# Fundamental Question

How do computation models characterize the informal notion of effective computability?

# Fundamental Result

**Theorem**. The set of functions definable (the Turing Machine Model, the URM Model) is precisely the set of functions definable in the Recursive Function Model.

*Proof*.
We showed that
$\mu$-definable $\Rightarrow$ $\lambda$-definable $\Rightarrow$ Turing definable $\Rightarrow$ URM-definable.

We will show that URM-definable $\Rightarrow$ $\mu$-definable.

# Church-Turing Thesis

**Church-Turing Thesis**.

The functions definable in all computation models are the same. They are precisely the computable functions.

1. Church believed that all computable functions are $\lambda$-definable.
2. Kleene termed it Church Thesis.
3. Gödel accepted it only after he saw Turing's equivalence proof.
4. Church-Turing Thesis is now universally accepted.

# Computable Function

Let $\mathcal{C}$ be the set of all computable functions.

Let $\mathcal{C}_n$ be the set of all $n$-ary computable functions.

# Power of Church-Turing Thesis

**No** one has come up with a computable function that is not in $\mathcal{C}$.

When you are convincing people of your model of computation, you are constructing an effective translation from your model to a well-known computation model.

# Use of Church-Turing Thesis

Church-Turing Thesis allows us to give an informal argument for the computability of a function.

We will make use of a computable function without explicitly defining it.

# Comment on Church-Turing Thesis

CTT and Physical Implementation

- Deterministic Turing Machines are physically implementable. This is the well-known von Neumann Architecture.
- Are quantum computers physically implementable? Can a quantum computer compute more? Can it compute more efficiently?

CTT, is it a Law of Nature or a Wisdom of Human?

# Synopsis

1. Gödel Encoding (section 4.1)
2. URM is Recursive (Appendix of chapter 5)

# Gödel Encoding

*Everything is number!*

# Godel's Insight

The set of syntactical objects of a formal system is denumerable.

More importantly, every syntactical object can be coded up effectively by a number in such a way that a unique syntactical object can be recovered from the number.

This is the crucial technique Gödel used in his proof of the Incompleteness Theorem.

# Enumeration

An enumeration of a set $X$ is a surjection $g : \mathbb{N} \to X$;
this is often represented by writing $\{x_0, x_1, x_2, \ldots\}$.

It is an enumeration without repetition if $g$ is injective.

# Denumeration

A set $X$ is denumerable if there is a bijection $f : X \to \mathbb{N}$.
(denumerate = denote + enumerate)

Let $X$ be a set of "finite objects".

Then $X$ is effectively denumerable if there is a bijection $f : X \to \mathbb{N}$ such that both $f$ and $f^{-1}$ are computable.

# Effective Denumerable Set

**Fact**. $\mathbb{N} \times \mathbb{N}$ is effectively denumerable.

*Proof*. A bijection $\pi : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is defined by

$$\begin{aligned} \pi(m,n) &\stackrel{\text{def}}{=} 2^m(2n+1) - 1, \\ \pi^{-1}(l) &\stackrel{\text{def}}{=} (\pi_1(l), \pi_2(l)), \end{aligned}$$

where

$$\begin{aligned} \pi_1(x) &\stackrel{\text{def}}{=} (x+1)_1, \\ \pi_2(x) &\stackrel{\text{def}}{=} ((x+1)/2^{\pi_1(x)} - 1)/2. \end{aligned}$$

# Effective Denumerable Set

**Fact**. $\mathbb{N}^+ \times \mathbb{N}^+ \times \mathbb{N}^+$ is effectively denumerable.

*Proof*. A bijection $\zeta : \mathbb{N}^+ \times \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}$ is defined by

$$\begin{aligned} \zeta(m,n,q) &\stackrel{\text{def}}{=} \pi(\pi(m-1,n-1),q-1), \\ \zeta^{-1}(l) &\stackrel{\text{def}}{=} (\pi_1(\pi_1(l))+1, \pi_2(\pi_1(l))+1, \pi_2(l)+1). \end{aligned}$$

# Effective Denumerable Set

**Fact**. $\bigcup_{k>0} \mathbb{N}^k$ is effectively denumerable.

*Proof.* A bijection $\tau : \bigcup_{k>0} \mathbb{N}^k \to \mathbb{N}$ is defined by

$$\begin{aligned}\tau(a_1, \ldots, a_k) \stackrel{\text{def}}{=} \; & 2^{a_1} + 2^{a_1+a_2+1} + 2^{a_1+a_2+a_3+2} + \ldots \\ & + 2^{a_1+a_2+a_3+\ldots,a_k+k-1} - 1.\end{aligned}$$

Now given $x$ it is easy to find $b_1 < b_2 < \ldots < b_k$ such that

$$2^{b_1} + 2^{b_2} + 2^{b_3} + \ldots + 2^{b_k} = x + 1.$$

It is then clear how to calculate $a_1, a_2, a_3, \ldots, a_k$. Details are next.

# Effective Denumerable Set

A number $x \in \mathbb{N}$ has a unique expression as

$$x = \sum_{i=0}^{\infty} \alpha_i 2^i,$$

where $\alpha_i$ is either $0$ or $1$ for all $i \geq 0$.

1. The function $\alpha(i,x) = \alpha_i$ is primitive recursive:

$$\alpha(i,x) = \mathsf{rm}(2, \mathsf{qt}(2^i, x)).$$

2. The function $\ell(x) = \textit{if } x > 0 \textit{ then } k \textit{ else } 0$ is primitive recursive:

$$\ell(x) = \sum_{i<x} \alpha(i,x).$$

# Effective Denumerable Set

3. If $x > 0$ then it has a unique expression as

$$x = 2^{b_1} + 2^{b_2} + \ldots + 2^{b_k},$$

where $1 \leq k$ and $0 \leq b_1 < b_2 < \ldots < b_k$.

The function $\mathsf{b}(i,x) = \mathit{if}\ (x > 0) \wedge (1 \leq i \leq \ell(x))\ \mathit{then}\ b_i\ \mathit{else}\ 0$ is primitive recursive:

$$\mathsf{b}(i,x) = \begin{cases} \mu y{<}x \left( \sum_{k \leq y} \alpha(k,x) = i \right), & \text{if } (x > 0) \wedge (1 \leq i \leq \ell(x)); \\ 0, & \text{otherwise.} \end{cases}$$

# Effective Denumerable Set

4. If $x > 0$ then it has a unique expression as

$$x = 2^{a_1} + 2^{a_1+a_2+1} + \ldots + 2^{a_l+a_2+\ldots+a_k+k-1}.$$

The function $\mathsf{a}(i,x) = a_i$ is primitive recursive:

$$\begin{aligned} \mathsf{a}(i,x) &= \mathsf{b}(i,x), \text{ if } i = 0 \text{ or } i = 1, \\ \mathsf{a}(i+1,x) &= (\mathsf{b}(i+1,x) \dot{-} \mathsf{b}(i,x)) \dot{-} 1, \text{ if } i \geq 1. \end{aligned}$$

We conclude that $a_1, a_2, a_3, \ldots, a_k$ can be calculated by primitive recursive functions.

# Encoding Program

Let $\mathcal{I}$ be the set of all instructions.

Let $\mathcal{P}$ be the set of all programs.

The objects in $\mathcal{I}$, and $\mathcal{P}$ as well, are “finite objects”.

# Encoding Program

**Theorem**. $\mathcal{I}$ is effectively denumerable.

*Proof*. The bijection $\beta : \mathcal{I} \to \mathbb{N}$ is defined as follows:

$$\begin{aligned}
\beta(Z(n)) &= 4(n-1), \\
\beta(S(n)) &= 4(n-1)+1, \\
\beta(T(m,n)) &= 4\pi(m-1,n-1)+2, \\
\beta(J(m,n,q)) &= 4\zeta(m,n,q)+3.
\end{aligned}$$

The converse $\beta^{-1}$ is easy.

# Encoding Program

**Theorem**. $\mathcal{P}$ is effectively denumerable.

*Proof*. The bijection $\gamma : \mathcal{P} \to \mathbb{N}$ is defined as follows:

$$\gamma(P) \quad = \quad \tau(\beta(I_1), \ldots, \beta(I_s)),$$

assuming $P = I_1, \ldots, I_s$.

The converse $\gamma^{-1}$ is obvious.

# Gödel Number of Program

The value $\gamma(P)$ is called the Gödel number of $P$.

$$\begin{aligned} P_n &= \text{the programme with Godel index } n \\ &= \gamma^{-1}(n) \end{aligned}$$

We shall fix this particular encoding function $\gamma$ throughout.

# Example

Let $P$ be the program $T(1,3), S(4), Z(6)$.

$\beta(T(1,3)) = 18$
$\beta(S(4)) = 13$
$\beta(Z(6)) = 20$

$\gamma(P) = 2^{18} + 2^{32} + 2^{53} - 1$

# Example

Consider $P_{4127}$.

$4127 = 2^5 + 2^{12} - 1$.

$\beta(I_1) = 4 + 1$
$\beta(I_2) = 4\pi(1,0) + 2$

So $P_{4127}$ is $S(2); T(2,1)$.

# URM is Recursive

# Kleene's Proof

Kleene demonstrated how to prove that machine computable functions are recursive functions.

# Proof in Detail

The states of the computation of the program $P_e(\widetilde{x})$ can be described by a configuration and an instruction number.

A state can be coded up by the number

$$\sigma = \pi(c, j),$$

where $c$ is the configuration that codes up the current values in the registers

$$c = 2^{r_1} 3^{r_2} \ldots = \prod_{i \geq 1} p_i^{r_i},$$

and $j$ is the next instruction number.

# Proof in Detail

To describe the changes of the states of $P_e(\widetilde{x})$, we introduce three $(n+2)$-ary functions:

$$
\begin{aligned}
\mathsf{c}_n(e,\widetilde{x},t) &= \text{the configuration after } t \text{ steps of } P_e(\widetilde{x}),\\
\mathsf{j}_n(e,\widetilde{x},t) &= \text{the number of the next instruction after } t \text{ steps}\\
&\quad \text{of } P_e(\widetilde{x}) \text{ (it is 0 if } P_e(\widetilde{x}) \text{ stops in } t \text{ or less steps)},\\
\sigma_n(e,\widetilde{x},t) &= \pi(\mathsf{c}_n(e,\widetilde{x},t),\mathsf{j}_n(e,\widetilde{x},t)).
\end{aligned}
$$

If $\sigma_n$ is primitive recursive, then $\mathsf{c}_n, \mathsf{j}_n$ are primitive recursive!

# Proof in Detail

If the computation of $P_e(\widetilde{x})$ stops, it does so in

$$\mu t(\mathsf{j}_n(e, \widetilde{x}, t) = 0)$$

steps.

Then the final configuration is

$$\mathsf{c}_n(e, \widetilde{x}, \mu t(\mathsf{j}_n(e, \widetilde{x}, t) = 0)).$$

We conclude that the value of the computation $P_e(\widetilde{x})$ is

$$(\mathsf{c}_n(e, \widetilde{x}, \mu t(\mathsf{j}_n(e, \widetilde{x}, t) = 0)))_1.$$

# Proof in Detail

The function $\sigma_n$ can be defined as follows:

$$\begin{aligned} \sigma_n(e,\widetilde{x},0) &= \pi(2^{x_1}3^{x_2}\ldots p_n^{x_n},1), \\ \sigma_n(e,\widetilde{x},t+1) &= \pi(\mathsf{config}(e,\sigma_n(e,\widetilde{x},t)),\mathsf{next}(e,\sigma_n(e,\widetilde{x},t))), \end{aligned}$$

where $\mathsf{config}(e,\pi(c,j))$ is the new configuration, and $\mathsf{next}(e,\pi(c,j))$ is the number of the next instruction, after the $j$-th instruction has been executed upon $c$.

# Proof in Detail

$$\mathsf{ln}(e) = \text{the number of instructions in } P_e;$$

$$\mathsf{gn}(e,j) = \begin{cases} \text{the code of } I_j \text{ in } P_e, & \text{if } 1 \le j \le \mathsf{ln}(e), \\ 0, & \text{otherwise.} \end{cases}$$

$$\mathsf{ch}(c,z) = \text{the resulting configuration when the configuration } c \text{ is operated on by the instruction with code number } z.$$

$$\mathsf{v}(c,j,z) = \begin{cases} \text{the number } j' \text{ of the next instruction when the configuration } c \text{ is operated on by the } j\text{th instruction with code } z, & \text{if } j > 0, \\ 0, & \text{if } j = 0. \end{cases}$$

# Proof in Detail

$$\mathsf{config}(e,\sigma) \quad = \quad \begin{cases} \mathsf{ch}(\pi_1(\sigma), \mathsf{gn}(e, \pi_2(\sigma))), & \text{if } 1 \leq \pi_2(\sigma) \leq \mathsf{ln}(e), \\ \pi_1(\sigma), & \text{otherwise.} \end{cases}$$

$$\mathsf{next}(e,\sigma) \quad = \quad \begin{cases} \mathsf{v}(\pi_1(\sigma), \pi_2(\sigma), \mathsf{gn}(e, \pi_2(\sigma))), & \text{if } 1 \leq \pi_2(\sigma) \leq \mathsf{ln}(e), \\ 0, & \text{otherwise.} \end{cases}$$

# Proof in Detail ($\mathsf{ln}, \mathsf{gn}$)

$$\mathsf{ln}(e) = \text{the number of instructions in } P_e;$$

$$\mathsf{gn}(e,j) = \begin{cases} \text{the code of } I_j \text{ in } P_e, & \text{if } 1 \le j \le \mathsf{ln}(e), \\ 0, & \text{otherwise.} \end{cases}$$

Both functions are primitive recursive since

$$\begin{aligned} \mathsf{ln}(e) &= \ell(e+1), \\ \mathsf{gn}(e,j) &= \mathsf{a}(j, e+1). \end{aligned}$$

# Proof in Detail ($\mathsf{ch}$)

The following function

$$\mathsf{ch}(c, z) \quad = \quad \text{the resulting configuration when the configuration } c \text{ is operated on by the instruction with code number } z.$$

is primitive recursive if

$$\mathsf{ch}(c, z) \quad = \quad \begin{cases} \mathsf{zero}(c, \mathsf{u}(z)), & \text{if } \mathsf{rm}(4, z) = 0, \\ \mathsf{succ}(c, \mathsf{u}(z)), & \text{if } \mathsf{rm}(4, z) = 1, \\ \mathsf{tran}(c, \mathsf{u}_1(z), \mathsf{u}_2(z)), & \text{if } \mathsf{rm}(4, z) = 2, \\ c, & \text{if } \mathsf{rm}(4, z) = 3. \end{cases}$$

# Proof in Detail (ch)

$\mathsf{u}(z) = m$ whenever $z = \beta(Z(m))$ or $z = \beta(S(m))$:

$$\mathsf{u}(z) = \mathsf{qt}(4, z) + 1.$$

$\mathsf{u}_1(z) = m_1$ and $\mathsf{u}_2(z) = m_2$ whenever $z = \beta(T(m_1, m_2))$:

$$\begin{aligned} \mathsf{u}_1(z) &= \pi_1(\mathsf{qt}(4, z)) + 1, \\ \mathsf{u}_2(z) &= \pi_2(\mathsf{qt}(4, z)) + 1. \end{aligned}$$

# Proof in Detail (ch)

The change in the configuration $c$ effected by instruction $Z(m)$:

$$\mathsf{zero}(c,m) = \mathsf{qt}(p_m^{(c)_m}, c).$$

The change in the configuration $c$ effected by instruction $S(m)$:

$$\mathsf{succ}(c,m) = p_m c.$$

The change in the configuration $c$ effected by instruction $T(m,n)$:

$$\mathsf{tran}(c,m,n) = \mathsf{qt}(p_n^{(c)_n}, p_n^{(c)_m} c).$$

# Proof in Detail ($\mathsf{v}$)

The following function

$$\mathsf{v}(c,j,z) \quad = \quad \begin{cases} \text{the number } j' \text{ of the next instruction} & \\ \text{when the configuration } c \text{ is operated} & \text{if } j > 0, \\ \text{on by the } j\text{th instruction with code } z, & \\ 0, & \text{if } j = 0. \end{cases}$$

is primitive recursive if

$$\mathsf{v}(c,j,z) \quad = \quad \begin{cases} j+1, & \text{if } \mathsf{rm}(4,z) \neq 3, \\ j+1, & \text{if } \mathsf{rm}(4,z) = 3 \ \wedge \ (c)_{\mathsf{v}_1(z)} \neq (c)_{\mathsf{v}_2(z)}, \\ \mathsf{v}_3(z), & \text{if } \mathsf{rm}(4,z) = 3 \ \wedge \ (c)_{\mathsf{v}_1(z)} = (c)_{\mathsf{v}_2(z)}. \end{cases}$$

# Proof in Detail ($\mathsf{v}$)

$\mathsf{v}_1(z) = m_1$ and $\mathsf{v}_2(z) = m_2$ and $\mathsf{v}_3(z) = q$ if $z = \beta(J(m_1, m_2, q))$:

$$\begin{aligned}
\mathsf{v}_1(z) &= \pi_1(\pi_1(\mathsf{qt}(4, z))) + 1, \\
\mathsf{v}_2(z) &= \pi_2(\pi_1(\mathsf{qt}(4, z))) + 1, \\
\mathsf{v}_3(z) &= \pi_2(\mathsf{qt}(4, z)) + 1.
\end{aligned}$$

# Proof in Detail

We can now define the function $\mathsf{config}(\_,\_)$ by

$$\mathsf{config}(e,\sigma) \quad = \quad \begin{cases} \mathsf{ch}(\pi_1(\sigma), \mathsf{gn}(e, \pi_2(\sigma))), & \text{if } 1 \leq \pi_2(\sigma) \leq \mathsf{ln}(e), \\ \pi_1(\sigma), & \text{otherwise.} \end{cases}$$

and the function $\mathsf{next}(\_,\_)$ by

$$\mathsf{next}(e,\sigma) \quad = \quad \begin{cases} \mathsf{v}(\pi_1(\sigma), \pi_2(\sigma), \mathsf{gn}(e, \pi_2(\sigma))), & \text{if } 1 \leq \pi_2(\sigma) \leq \mathsf{ln}(e), \\ 0, & \text{otherwise.} \end{cases}$$

# Proof in Detail

We conclude that the functions $\mathsf{c}_n, \mathsf{j}_n, \sigma_n$ are primitive recursive.

# Computability Theory VII

## S-M-N Theorem

Guoqiang Li

Shanghai Jiao Tong University

Nov. 14, 2014

# Problem Index

# Motivation

By Church-Turing Thesis one may study computability theory using any of the computation models.

It is much more instructive however to carry out the study in a model independent manner.

The first step is to assign index to computable function.

# Review Tips

# Effective Denumerable Set

$\mathbb{N} \times \mathbb{N}$

$\mathbb{N}^+ \times \mathbb{N}^+ \times \mathbb{N}^+$

$\bigcup_{k>0} \mathbb{N}^k$

$\mathcal{I}$ is effectively denumerable.

$\mathcal{P}$ is effectively denumerable.

$\gamma(P) = \tau(\beta(I_1), \ldots, \beta(I_s))$

The value $\gamma(P)$ is called the Gödel number of $P$.

# Synopsis

1. Gödel Index
2. S-m-n Theorem

# Gödel Index

# Basic Idea

We see a number as an index for a problem/function if it is the Gödel number of a programme that solves/calculates the problem/function.

# Definition

Suppose $a \in \mathbb{N}$ and $n \geq 1$.

$$
\begin{aligned}
\phi_a^{(n)} &= \text{the } n \text{ ary function computed by } P_a \\
&= f_{P_a}^{(n)}, \\
W_a^{(n)} &= \text{the domain of } \phi_a^{(n)} = \{(x_1, \ldots, x_n) \mid P_a(x_1, \ldots, x_n)\downarrow\}, \\
E_a^{(n)} &= \text{the range of } \phi_a^{(n)}.
\end{aligned}
$$

The super script $(n)$ is omitted when $n = 1$.

# Example

Let $a = 4127$. Then $P_{4127} = S(2); T(2,1)$.

If the program is seen to calculate a unary function, then

$$\begin{aligned} \phi_{4127}(x) &= 1, \\ W_{4127} &= \mathbb{N}, \\ E_{4127} &= \{1\}. \end{aligned}$$

If the program is seen to calculate an $n$-ary function, then

$$\begin{aligned} \phi^{(n)}_{4127}(x_1, \ldots, x_n) &= x_2 + 1, \\ W^n_{4127} &= \mathbb{N}^n, \\ E^n_{4127} &= \mathbb{N}^+. \end{aligned}$$

# Gödel Index for Computable Function

Suppose $f$ is an $n$-ary computable function..

A number $a$ is an index for $f$ if $f = \phi_a^{(n)}$.

# Padding Lemma

## Padding Lemma

Every computable function has infinite indices. Moreover for each $x$ we can effectively find an infinite recursive set $A_x$ of indices for $\phi_x$.

## Proof

Systematically add useless instructions to $P_x$.

# Enumeration of Computable Function

**Proposition**

$\mathcal{C}_n$, and $\mathcal{C}$ as well, is denumerable.

We may list for example all the elements of $\mathcal{C}_n$ as $\phi_0^{(n)}, \phi_1^{(n)}, \phi_2^{(n)}, \ldots$.

# Diagonal Method

**Fact**

There is a total unary function that is not computable.

**Proof**

Suppose $\phi_0, \phi_1, \phi_2, \ldots$ is an enumeration of $\mathcal{C}$. Define

$$f(n) \quad = \quad \begin{cases} \phi_n(n) + 1, & \text{if } \phi_n(n) \text{ is defined,} \\ 0, & \text{if } \phi_n(n) \text{ is undefined.} \end{cases}$$

By Church-Turing Thesis the function $f(n)$ is not computable.

Is the following function computable?

$$f(n) \quad \simeq \quad \begin{cases} \phi_n(n) + 1, & \text{if } \phi_n(n) \text{ is defined,} \\ \uparrow, & \text{if } \phi_n(n) \text{ is undefined.} \end{cases}$$

# Diagonal Method

Suppose there is a sequence $f_0, f_1, \ldots, f_n, \ldots$

Diagonalize out of $f_0, f_1, \ldots$ by making $f$ differ from $f_n$ at $n$.

# S-m-n Theorem

# Motivation

*How do different indexing systems relate?*

# S-m-n Theorem, the Unary Case

Given a binary function $f(x, y)$, we get a unary computable function $f(a, y)$ by fixing a value $a$ for $x$.

Let $e$ be an index for $f(a, y)$. Then

$$f(a, y) \simeq \phi_e(y)$$

S-m-n Theorem states that the index $e$ can be computed from $a$.

# S-m-n Theorem, the Unary Case

**Fact**

Suppose that $f(x, y)$ is a computable function. There is a primitive recursive function $k(x)$ such that

$$f(x, y) \simeq \phi_{k(x)}(y).$$

# S-m-n Theorem, the Unary Case

Proof

Let $F$ be a program that computes $f$. Consider the following program

$$
\begin{array}{l}
T(1,2) \\
Z(1) \\
\left.\begin{array}{l} S(1) \\ \vdots \\ S(1) \end{array}\right\} a \text{ times} \\
F
\end{array}
$$

The above program can be effectively constructed from $a$.

Let $k(a)$ be the Gödel number of the above program. It can be effectively computed from the above program.

# Examples

Let $f(x,y) = y^x$.

Then $\phi_{k(x)}(y) = y^x$. For each fixed $n$, $k(n)$ is an index for $y^n$.

Let $f(x,y) \simeq \begin{cases} y, & \text{if } y \text{ is a multiple of } x, \\ \uparrow, & \text{otherwise.} \end{cases}$.

Then $\phi_{k(n)}(y)$ is defined if and only if $y$ is a multiple of $n$.

# S-m-n Theorem

**S-m-n Theorem**

For $m, n$, there is an injective primitive recursive $(m+1)$-function $s_n^m(x, \widetilde{x})$ such that for all $e$ the following holds:

$$\phi_e^{(m+n)}(\widetilde{x}, \widetilde{y}) \simeq \phi_{s_n^m(e,\widetilde{x})}^{(n)}(\widetilde{y})$$

S-m-n Theorem is also called **Parameter Theorem**.

# S-m-n Theorem

Proof

Given $e, x_1, \ldots, x_m$, we can effectively construct the following program and its index

$$
\begin{aligned}
&T(n, m+n)\\
&\vdots\\
&T(1, m+1)\\
&Q(1, x_1)\\
&\vdots\\
&Q(m, x_m)\\
&P_e
\end{aligned}
$$

where $Q(i, x)$ is the program $Z(i), \underbrace{S(i), \ldots, S(i)}_{x \text{ times}}$.

The injectivity is achieved by padding enough useless instructions.

# Exercise I

Show that there is a total computable function $k$ such that for each $n$, $k(n)$ is an index of the function $\lceil \sqrt[n]{x} \rceil$.

# Exercise II

Show that there is a total computable function $k$ such that for each $n$, $W_{k(n)} =$ the set of perfect $n$th power.

# Exercise III

Show that there is a total computable function $k$ such that

$$W^{(m)}_{k(n)} = \{(y_1, \ldots, y_m) : y_1 + y_2 + \ldots + y_m = n\}$$

suppose $m \geq 1$.

# Computability Theory VIII

## Universal Program

Guoqiang Li

Shanghai Jiao Tong University

Nov. 28, 2014

# Enumeration Theorem

# General Remark

There are universal programs that embody all the programs.

A program is universal if upon receiving the Gödel number of a program it simulates the program indexed by the number.

# Intuition

Consider the function $\psi(x, y)$ defined as follows

$$\psi(x, y) \simeq \phi_x(y)$$

In an obvious sense $\psi(x, \_)$ is a universal function for the unary functions

$$\phi_0, \phi_1, \phi_2, \phi_3, \ldots.$$

# Universal Function

The universal function for $n$-ary computable functions is the $(n+1)$-ary function $\psi_U^{(n)}$ defined by

$$\psi_U^{(n)}(e, x_1, \ldots, x_n) \simeq \phi_e^{(n)}(x_1, \ldots, x_n).$$

We write $\psi_U$ for $\psi_U^{(1)}$.

Question: Is $\psi_U^{(n)}$ computable?

# Enumeration Theorem

## Enumeration Theorem

For each $n$, the universal function $\psi_U^{(n)}$ is computable.

## Proof

Given a number $e$, decode the number to get the program $P_e$; and then simulate the program $P_e$. If the simulation ever terminates, then return the number in $R_1$. By Church-Turing Thesis, $\psi_U^{(n)}$ is computable.

# Undecidability

**Proposition**

The problem '$\phi_x$ is total' is undecidable.

Proof

If '$\phi_x$ is total' were decidable, then by Church-Turing Thesis

$$f(x) = \begin{cases} \psi_U(x,x) + 1, & \text{if } \phi_x \text{ is total,} \\ 0, & \text{if } \phi_x \text{ is not total.} \end{cases}$$

would be a total computable function that differs from every total computable function.

# Effectiveness of Function Operation

**Proposition**

There is a total computable function $s(x,y)$ such that $\phi_{s(x,y)} = \phi_x \phi_y$ for all $x, y$.

Proof

Let $f(x,y,z) \simeq \phi_x(z)\phi_y(z) \simeq \psi_U(x,z)\psi_U(y,z)$.

By S-m-n Theorem there is a total function $s(x,y)$ such that $\phi_{s(x,y)}(z) \simeq f(x,y,z)$.

# Effectiveness of Set Operation

**Proposition**

There is a total computable function $s(x,y)$ such that $W_{s(x,y)} = W_x \cup W_y$.

Proof

Let

$$f(x,y,z) = \begin{cases} 1, & \text{if } z \in W_x \text{ or } z \in W_y, \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

By S-m-n Theorem there is a total function $s(x,y)$ such that $\phi_{s(x,y)}(z) \simeq f(x,y,z)$. Clearly $W_{s(x,y)} = W_x \cup W_y$.

# Effectiveness of Recursion

Consider $f$ defined by the following recursion

$$f(e_1, e_2, \widetilde{x}, 0) \simeq \phi_{e_1}^{(n)}(\widetilde{x}) \simeq \psi_U^{(n)}(e_1, \widetilde{x})$$

and

$$\begin{aligned} f(e_1, e_2, \widetilde{x}, y+1) &\simeq \phi_{e_2}^{(n+2)}(\widetilde{x}, y, f(e_1, e_2, \widetilde{x}, y)) \\ &\simeq \psi_U^{(n+2)}(e_2, \widetilde{x}, y, f(e_1, e_2, \widetilde{x}, y)). \end{aligned}$$

By S-m-n Theorem, there is a total computable function $r(e_1, e_2)$ such that

$$\phi_{r(e_1,e_2)}^{(n+1)}(\widetilde{x}, y) \simeq f(e_1, e_2, \widetilde{x}, y)$$

# Non-Primitive Recursive Total Function

**Theorem**

There is a total computable function that is not primitive recursive.

Proof

1. The primitive recursive functions have a universal function.
2. Such a function cannot be primitive recursive by diagonalisation.

# Recursion Theorem

# Recursion Theorem

## Recursion Theorem

Let $f$ be a total unary computable function. Then there is a number $n$ such that $\phi_{f(n)} = \phi_n$.

### Proof

By S-m-n Theorem there is an injective primitive recursive function $s(x)$ such that for all $x$

$$\phi_{s(x)}(y) \simeq \begin{cases} \phi_{\phi_x(x)}(y), & \text{if } \phi_x(x)\downarrow; \\ \uparrow, & \text{otherwise.} \end{cases}$$

Let $v$ be such that $\phi_v = s \circ f$. Obviously $\phi_v$ is total and $\phi_v(v)\downarrow$.

$$\phi_{s(v)} = \phi_{\phi_v(v)} = \phi_{f(s(v))}$$

We are done by letting $n$ be $s(v)$.

# Exercise I

Show that there is a total computable function $k$ such that for each $n$, $E_{k(n)} = W_n$.

# Exercise II

Show that there is a total computable function $k(x, y)$ such that for each $x, y$, $E_{k(x,y)} = E_x \cup E_y$.

# Exercise III

Suppose $f(n)$ is computable, show that there is a total computable function $k(n)$ such that for each $n$, $W_{k(n)} = f^{-1}(W_n)$.

# Computability Theory IX

## Undecidability

Guoqiang Li

Shanghai Jiao Tong University

Nov. 28, 2014

*Assignment 3 was announced!*

*The deadline is Dec. 12.*

# Undecidability

# Decidability and Undecidability

A predicate $M(\mathbf{x})$ is decidable if its characteristic function $c_M(\mathbf{x})$ given by

$$c_M(\mathbf{x}) = \begin{cases} 1, & \text{if } M(\mathbf{x}) \text{ holds,} \\ 0, & \text{if } M(\mathbf{x}) \text{ does not hold.} \end{cases}$$

is computable.

The predicate $M(\mathbf{x})$ is undecidable if it is not decidable.

# Undecidability Result

**Theorem**

The problem '$x \in W_x$' is undecidable.

*Proof*

The characteristic function of this problem is given by

$$c(x) = \begin{cases} 1, & \text{if } x \in W_x, \\ 0, & \text{if } x \notin W_x. \end{cases}$$

Suppose $c(x)$ was computable. Then the function $g(x)$ defined below would also be computable.

$$g(x) = \begin{cases} 0, & \text{if } c(x) = 0, \\ \text{undefined}, & \text{if } c(x) = 1. \end{cases}$$

Let $m$ be an index for $g$. Then

$$m \in W_m \text{ iff } c(m) = 0 \text{ iff } m \notin W_m.$$

# Undecidability Result

**Corollary**

There is a computable function $h$ such that both '$x \in Dom(h)$' and '$x \in Ran(h)$' are undecidable.

*Proof*

Let

$$h(x) = \begin{cases} x, & \text{if } x \in W_x, \\ \text{undefined}, & \text{if } x \notin W_x. \end{cases}$$

Clearly $x \in Dom(h)$ iff $x \in W_x$ iff $x \in Ran(h)$.

# Undecidability Result

**Theorem**
The problem '$\phi_x(y)$ is defined' is undecidable.

*Proof*
If $y \in W_x$ were decidable then $x \in W_x$ would be decidable.

This is known as Halting Problem.

In this proof we have reduced the problem '$x \in W_x$' to the problem '$y \in W_x$'. The reduction shows that the latter is at least as hard as the former.

# Undecidability Result

**Theorem**

The problem '$\phi_x = \mathbf{0}$' is undecidable.

*Proof*

Consider the function $f$ defined by

$$f(x,y) = \begin{cases} 0, & \text{if } x \in W_x, \\ \text{undefined}, & \text{if } x \notin W_x. \end{cases}$$

By s-m-n theorem there is some total computable function $k(x)$ such that $\phi_{k(x)}(y) \simeq f(x,y)$.

It is clear that $\phi_{k(x)} = \mathbf{0}$ iff $x \in W_x$.

# Undecidability Result

**Corollary**

The problem '$\phi_x \simeq \phi_y$' is undecidable.

# Undecidability Result

**Theorem**

Let $c$ be any number. The followings are undecidable.

(a) Acceptance Problem: '$c \in W_x$',

(b) Printing Problem: '$c \in E_x$'.

*Proof*

Consider the function $f$ defined by

$$f(x,y) = \begin{cases} y, & \text{if } x \in W_x, \\ \text{undefined}, & \text{if } x \notin W_x. \end{cases}$$

By s-m-n theorem there is some total computable function $k(x)$ such that $\phi_{k(x)}(y) \simeq f(x,y)$.

It is clear that $c \in W_{k(x)}$ iff $x \in W_x$ iff $c \in E_{k(x)}$.

# More on Undecidability

# Exercise I

$x \in E_x$

# Exercise II

$W_x = W_y$

# Exercise III

$\phi_x(y) = 0$

# Exercise IV

$E_x$ is infinite.

# Exercise V

‘$\phi_x = g$’, where $g$ is any fixed computable function.

# Computability Theory X

## Recursive Set

Guoqiang Li

Shanghai Jiao Tong University

Dec. 5, 2014

# Decision Problem, Predicate, Number Set

The following emphasizes the importance of number set:

$$\text{Decision Problem} \Leftrightarrow \text{Predicate on Number} \Leftrightarrow \text{Set of Number}$$

A central theme of recursion theory is to look for sensible classification of number sets.

Classification is often done with the help of reduction.

# Synopsis

1. Reduction
2. Recursive Set
3. Rice Theorem

# Reduction

# Reduction between Problems

A reduction is a way of defining a solution of a problem with the help of a solution of another problem.

In recursion theory we are only interested in reductions that are computable.

# Reduction

There are several ways of reducing a problem to another.

The differences between different reductions from $A$ to $B$ consists in the manner and the extent to which information about $B$ is allowed to settle questions about $A$.

# Many-One Reduction

The set $A$ is many-one reducible, or m-reducible, to the set $B$ if there is a total computable function $f$ such that

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

for all $x$. We shall write $A \leq_m B$ or more explicitly $f : A \leq_m B$.

If $f$ is injective, then it is a one-one reducibility, denoted by $\leq_1$.

# An Example

Suppose $G$ is a finite graph and $k$ is a natural number.

- The Independent Set Problem (IS) asks if there are $k$ vertices of $G$ with every pair of which unconnected.
- The Clique Problem asks if there is a $k$-complete subgraph of $G$.

There is a simple one-one reduction from IS to Clique.

# Many-One Reduction

- $\leq_m$ is reflexive and transitive.
- $A \leq_m B$ iff $\overline{A} \leq_m \overline{B}$.
- $A \leq_m \mathbb{N}$ iff $A = \mathbb{N}$; $A \leq_m \emptyset$ iff $A = \emptyset$.
- $\mathbb{N} \leq_m A$ iff $A \neq \emptyset$; $\emptyset \leq_m A$ iff $A \neq \mathbb{N}$.

# m-Degree

Two sets $A, B$ are many-one equivalent, notation $A \equiv_m B$, if $A \leq_m B$ and $B \leq_m A$.

Similarly $A \equiv_1 B$ if $A \leq_1 B$ and $B \leq_1 A$.

Clearly both $\equiv_m$ and $\equiv_1$ an equivalence relation.

Let $d_m(A)$ be $\{B \mid A \equiv_m B\}$.

The class $d_m(A)$ is called the m-degree represented by $A$.

# m-Degree

The set of m-degrees is ranged over by $\mathbf{a}, \mathbf{b}, \mathbf{c}, \ldots$.

$\mathbf{a} \leq_m \mathbf{b}$ iff $A \leq_m B$ for some $A \in \mathbf{a}$ and $B \in \mathbf{b}$.

$\mathbf{a} <_m \mathbf{b}$ iff $\mathbf{a} \leq_m \mathbf{b}$ and $\mathbf{b} \not\leq_m \mathbf{a}$.

The relation $\leq_m$ is a partial order.

# The Structure of m-Degree

**Proposition**

The m-degrees form a distributive lattice.

# The Restriction of m-Reduction

Suppose $G$ is a finite directed weighted graph and $m$ is a number.

- The Travelling Salesman Problem (TSP) asks for the overall weight of a cycle with minimum weight if there are circles.
- A decision problem version asks given a budget $b$, whether there exists a cycle that passes through every vertex exactly once, of total cost $b$ or less - or to report that no such tour exists.

# Recursive Set

# Definition of Recursive Set

Let $A$ be a subset of $\mathbb{N}$. The characteristic function of $A$ is given by

$$c_A(x) = \begin{cases} 1, & \text{if } x \in A, \\ 0, & \text{if } x \notin A. \end{cases}$$

$A$ is recursive if $c_A(x)$ is computable.

# Fact about Recursive Set

**Fact**. If $A$ is recursive then $\overline{A}$ is recursive.

**Fact**. If $A$ is recursive and $B \neq \emptyset, \mathbb{N}$, then $A \leq_m B$.

**Fact**. If $A, B$ are recursive and $A, B, \overline{A}, \overline{B}$ are infinite then $A \equiv B$.

**Fact**. If $A \leq_m B$ and $B$ is recursive, then $A$ is recursive.

**Fact**. If $A \leq_m B$ and $A$ is not recursive, then $B$ is not recursive.

# A Characterization of Recursive Set

**Theorem**. An infinite set is recursive iff it is the range of a total increasing computable function.

*Proof*. Suppose $A$ is recursive and infinite. Then $A$ is range of the increasing function $f$ given by

$$\begin{aligned} f(0) &= \mu y(y \in A), \\ f(n+1) &= \mu y(y \in A \text{ and } y > f(n)). \end{aligned}$$

The function is total, increasing and computable.

Conversely suppose $A$ is the range of a total increasing computable function $f$. Obviously $y = f(n)$ implies $n \leq y$. Hence

$$y \in A \Leftrightarrow y \in Ran(f) \Leftrightarrow \exists n \leq y(f(n) = y).$$

# Unsolvable Problem

A decision problem $f : \mathbb{N} \to \{0, 1\}$ is solvable if it is computable and $dom(f)$ is recursive.

It is unsolvable if it is not solvable.

Non-recursive $\Leftrightarrow$ Unsolvable $\Leftrightarrow$ Undecidable

# Some Important Undecidable Sets

Here are some important undecidable sets:

$$
\begin{aligned}
K &= \{x \mid x \in W_x\}, \\
K_0 &= \{\pi(x,y) \mid x \in W_y\}, \\
K_1 &= \{x \mid W_x \neq \emptyset\}, \\
\mathit{Fin} &= \{x \mid W_x \text{ is finite}\}, \\
\mathit{Inf} &= \{x \mid W_x \text{ is infinite}\}, \\
\mathit{Con} &= \{x \mid \phi_x \text{ is total and constant}\}, \\
\mathit{Tot} &= \{x \mid \phi_x \text{ is total}\}, \\
\mathit{Cof} &= \{x \mid W_x \text{ is cofinite}\}, \\
\mathit{Rec} &= \{x \mid W_x \text{ is recursive}\}, \\
\mathit{Ext} &= \{x \mid \phi_x \text{ is extensible to a total recursive function}\}.
\end{aligned}
$$

# Rice Theorem

# Rice Theorem

**Henry Rice**.

Classes of Recursively Enumerable Sets and their Decision Problems. Transactions of the American Mathematical Society, **77**:358-366, 1953.

# Rice Theorem

**Rice Theorem**. (1953)

If $\emptyset \subsetneq \mathcal{B} \subsetneq \mathcal{C}$, then $\{x \mid \phi_x \in \mathcal{B}\}$ is not recursive.

*Proof*. Suppose $f_\emptyset \notin \mathcal{B}$ and $g \in \mathcal{B}$. Let the function $f$ be defined by

$$f(x,y) = \begin{cases} g(y), & \text{if } x \in W_x, \\ \uparrow, & \text{if } x \notin W_x. \end{cases}$$

By S-m-n Theorem there is some injective primitive recursive function $k(x)$ such that $\phi_{k(x)}(y) \simeq f(x,y)$.

It is clear that $k$ is a one-one reduction from $K$ to $\{x \mid \phi_x \in \mathcal{B}\}$.

# Applying Rice Theorem

Assume that $f(x) \simeq \phi_x(x) + 1$ could be extended to a total computable function say $g(x)$. Let $e$ be an index of $g(x)$. Then $\phi_e(e) = g(e) = f(e) = \phi_e(e) + 1$. Contradiction.

So we may use Rice Theorem to conclude that

$$Ext \quad = \quad \{x \mid \phi_x \text{ is extensible to a total recursive function}\}$$

is not recursive.

**Comment**: Not every partial recursive function can be obtained by restricting a total recursive function.

# Remark on Rice Theorem

Rice Theorem deals with programme independent properties.

It talks about classes of computable functions rather than classes of programmes.

All non-trivial semantic problems are algorithmically undecidable.

# Computability Theory XI

## Recursively Enumerable Set

Guoqiang Li

Shanghai Jiao Tong University

Dec. 12&19, 2013

# Assignment

*Assignment 4 was announced!*

*The deadline is Dec. 26!*

# An Exercise

Let $A, B \subseteq \mathbb{N}$. Define sets of $A \oplus B$ and $A \otimes B$ by

$$\begin{aligned} A \oplus B &= \{2x \mid x \in A\} \cup \{2x+1 \mid x \in B\} \\ A \otimes B &= \{\pi(x,y) \mid x \in A \wedge y \in B\} \end{aligned}$$

1. $A \oplus B$ is recursive iff $A$ and $B$ are both recursive.
2. if $A, B \neq \emptyset$, then $A \otimes B$ is recursive iff $A$ and $B$ are both recursive.

We have seen that many sets, although not recursive, can be effectively generated in the sense that, for any such set, there is an effective procedure that produces the elements of the set in a non-stop manner.

We shall formalize this intuition in this lecture.

# Synopsis

1. Recursively Enumerable Set
2. Characterization of R.E. Set
3. Rice-Shapiro Theorem

# Recursively Enumerable Set

# The Definition of R.E. Set

The partial characteristic function of a set $A$ is given by

$$\chi_A(x) = \begin{cases} 1, & \text{if } x \in A, \\ \uparrow, & \text{if } x \notin A. \end{cases}$$

$A$ is recursively enumerable if $\chi_A$ is computable.

We shall often abbreviate 'recursively enumerable set' to 'r.e. set'.

# Partially Decidable Problem

A problem $f : \mathbb{N} \to \{0, 1\}$ is partially decidable if $dom(f)$ is r.e.

# Partially Decidable Predicate

A predicate $M(\widetilde{x})$ of natural number is partially decidable if its partial characteristic function

$$\chi_M(\widetilde{x}) = \begin{cases} 1, & \text{if } M(\widetilde{x}) \text{ holds,} \\ \uparrow, & \text{if } M(\widetilde{x}) \text{ does not hold,} \end{cases}$$

is computable.

Partially Decidable Problem $\Leftrightarrow$ Partially Decidable Predicate

$\Leftrightarrow$ Recursively Enumerable Set

# Example

The halting problem is partially decidable. Its partial characteristic function is given by

$$\chi_H(x,y) = \begin{cases} 1, & \text{if } P_x(y)\downarrow, \\ \uparrow, & \text{otherwise.} \end{cases}$$

$K, K_0, K_1$ are r.e.. But none of $\overline{K}, \overline{K_0}, \overline{K_1}$ is r.e..

# Index for Recursively Enumerable Set

A set is r.e. iff it is the domain of a unary computable function.

So $W_0, W_1, W_2, \ldots$ is an enumeration of all r.e. sets.

Every r.e. set has an infinite number of indexes.

# Closure Property

**Union Theorem**. The recursively enumerable sets are closed under union and intersection uniformly and effectively.

*Proof*. According to S-m-n Theorem there are primitive recursive functions $r(x,y), s(x,y)$ such that

$$\begin{aligned} W_{r(x,y)} &= W_x \cup W_y, \\ W_{s(x,y)} &= W_x \cap W_y. \end{aligned}$$

# The Most Hard Recursively Enumerable Set

**Fact**. If $A \leq_m B$ and $B$ is r.e. then $A$ is r.e..

**Theorem**. $A$ is r.e. iff $A \leq_1 K$.

*Proof*. Suppose $A$ is r.e. Let $f(x,y)$ be defined by

$$f(x,y) = \begin{cases} 1, & \text{if } x \in A, \\ \uparrow, & \text{if } x \notin A. \end{cases}$$

By S-m-n Theorem there is an injective primitive recursive function $s(x)$ s.t. $f(x,y) = \phi_{s(x)}(y)$. It is clear that $x \in A$ iff $s(x) \in K$.

**Comment**. No r.e. set is more difficult than $K$.

# Characterization of R.E. Set

# Normal Form Theorem

**Normal Form Theorem**. $M(\widetilde{x})$ is partially decidable iff there is a primitive recursive predicate $R(\widetilde{x}, y)$ such that $M(\widetilde{x})$ iff $\exists y.R(\widetilde{x}, y)$.

*Proof*. If $R(\widetilde{x}, y)$ is primitive recursive and $M(\widetilde{x}) \Leftrightarrow \exists y.R(\widetilde{x}, y)$, then the computable function '**if** $\mu y R(\widetilde{x}, y)$ **then 1 else** $\uparrow$' is the partial characteristic function of $M(\widetilde{x})$.

Conversely suppose $M(\widetilde{x})$ is partially decided by $P$. Let $R(\widetilde{x}, y)$ be

$$P(\widetilde{x}) \downarrow \text{ in } y \text{ steps.}$$

Then $R(\widetilde{x}, y)$ is primitive recursive and $M(\widetilde{x}) \Leftrightarrow \exists y.R(\widetilde{x}, y)$.

# Quantifier Contraction Theorem

**Quantifier Contraction Theorem**. If $M(\widetilde{x}, y)$ is partially decidable, so is $\exists y.M(\widetilde{x}, y)$.

*Proof*. Let $R(\widetilde{x}, y, z)$ be a primitive recursive predicate such that

$$M(\widetilde{x}, y) \Leftrightarrow \exists z.R(\widetilde{x}, y, z)$$

Then $\exists y.M(\widetilde{x}, y) \Leftrightarrow \exists y.\exists z.R(\widetilde{x}, y, z) \Leftrightarrow \exists u.R(\widetilde{x}, (u)_0, (u)_1)$.

# Examples

The following predicates are partially decidable:

$x \in E_y^{(n)}$

$W_x \neq \emptyset$

# Uniformisation Theorem

**Uniformisation Theorem**. If $R(x, y)$ is partially decidable, then there is a computable function $c(x)$ such that $c(x) \downarrow$ iff $\exists y.R(x, y)$ and $c(x) \downarrow$ implies $R(x, c(x))$.

We may think of $c(x)$ as a choice function for $R(x, y)$. The theorem states that the choice function is computable.

$A$ is r.e. iff there is a partially decidable predicate $R(x, y)$ such that $x \in A$ iff $\exists y.R(x, y)$.

# Complementation Theorem

**Complementation Theorem**. $A$ is recursive iff $A$ and $\overline{A}$ are r.e.

*Proof*. Suppose $A$ and $\overline{A}$ are r.e. Then some primitive recursive predicates $R(x,y), S(x,y)$ exist such that

$$\begin{aligned} x \in A &\Leftrightarrow \exists y R(x,y), \\ x \in \overline{A} &\Leftrightarrow \exists y S(x,y). \end{aligned}$$

Now let $f(x)$ be $\mu y(R(x,y) \vee S(x,y))$.

Then $f(x)$ is total and computable, and

$$x \in A \Leftrightarrow R(x, f(x))$$

# Applying Complementation Theorem

**Fact**. $\overline{K}$ is not r.e.

**Comment**. If $\overline{K} \leq_m A$ then $A$ is not r.e. either.

# Applying Complementation Theorem

**Fact**. If $A$ is r.e. but not recursive, then $\overline{A} \not\leq_m A \not\leq_m \overline{A}$.

**Comment**. However $A$ and $\overline{A}$ are intuitively equally difficult.

# Graph Theorem

**Graph Theorem**. Let $f(x)$ be a partial function. Then $f(x)$ is computable iff the predicate '$f(x) \simeq y$' is partially decidable iff $\{\pi(x,y) \mid f(x) \simeq y\}$ is r.e.

*Proof*. If $f(x)$ is computable by $P(x)$, then

$$f(x) \simeq y \Leftrightarrow \exists t.(P(x) \downarrow y \text{ in } t \text{ steps})$$

The predicate '$P(x) \downarrow y$ in $t$ steps' is primitive recursive.

Conversely let $R(x,y,t)$ be such that

$$f(x) \simeq y \Leftrightarrow \exists t.R(x,y,t).$$

Now $f(x) = \mu y.R(x,y,\mu t.R(x,y,t))$.

# Listing Theorem

**Listing Theorem**. $A$ is r.e. iff either $A = \emptyset$ or $A$ is the range of a unary total computable function.

*Proof*. Suppose $A$ is nonempty and its partial characteristic function is computed by $P$. Let $a$ be a member of $A$. The total function $g(x,t)$ given by

$$g(x,t) = \begin{cases} x, & \text{if } P(x)\downarrow \text{ in } t \text{ steps,} \\ a, & \text{otherwise.} \end{cases}$$

is computable. Clearly $A$ is the range of $h(z) = g((z)_1, (z)_2)$.

Conversely, $x \in A$ iff $\exists y.h(y) = x$, $\exists y.h(y) = x$ is partially decidable.

# Listing Theorem

The theorem gives rise to the terminology ‘recursively enumerable’.

# Implication of Listing Theorem

A set is r.e. iff it is the range of a computable function.

# Implication of Listing Theorem

**Corollary**. For each infinite nonrecursive r.e. $A$, there is an injective total recursive function $f$ such that $ran(f) = A$.

**Corollary**. Every infinite r.e. set has an infinite recursive subset.

*Proof*. Suppose $A = ran(f)$. An infinite recursive subset is enumerated by the total increasing computable function $g$ given by

$$\begin{aligned} g(0) &= f(0), \\ g(n+1) &= f(\mu y(f(y) > g(n))). \end{aligned}$$

# Applying Listing Theorem

**Fact**. The set $\{x \mid \phi_x \text{ is total}\}$ is not r.e.

*Proof*. If $\{x \mid \phi_x \text{ is total}\}$ were a r.e. set, then there would be a total computable function $f$ whose range is the r.e. set.

The function $g(x)$ given by $g(x) = \phi_{f(x)}(x) + 1$ would be total and computable.

# Rice-Shapiro Theorem

# Rice-Shapiro Theorem

**Rice-Shapiro Theorem**. Suppose that $\mathcal{A}$ is a set of unary computable functions such that the set $\{x \mid \phi_x \in \mathcal{A}\}$ is r.e.
Then for any unary computable function $f$, $f \in \mathcal{A}$ iff there is a finite function $\theta \subseteq f$ with $\theta \in \mathcal{A}$.

**Comment**. Intuitively a set of recursive functions is r.e. iff it is effectively generated by an r.e. set of finite functions.

# Applications of the Rice-Shapiro Theorem

Both $Tot$ and $\overline{Tot}$ are not r.e.

$$Tot = \{x \mid \phi_x \text{ is total}\}$$

Proof

We apply the Rice-Shapiro theorem on $Tot$. For no $f \in Tot$ is there a finite $\theta \subseteq f$ with $\theta \in Tot$.

If $f$ is any total computable function, $f \notin \overline{Tot}$; but every finite function $\theta \subseteq f$ in $\overline{Tot}$.

# What Rice-Shapiro Theorem Can Do

Can we apply Rice-Shapiro Theorem to show that any of the following sets is non-r.e.:

$$
\begin{aligned}
\mathit{Fin} &= \{x \mid W_x \text{ is finite}\},\\
\mathit{Inf} &= \{x \mid W_x \text{ is infinite}\},\\
\mathit{Tot} &= \{x \mid \phi_x \text{ is total}\},\\
\mathit{Con} &= \{x \mid \phi_x \text{ is total and constant}\},\\
\mathit{Cof} &= \{x \mid W_x \text{ is cofinite}\},\\
\mathit{Rec} &= \{x \mid W_x \text{ is recursive}\},\\
\mathit{Ext} &= \{x \mid \phi_x \text{ is extensible to a total recursive function}\}.
\end{aligned}
$$

# Proof of Rice-Shapiro Theorem

Suppose $A = \{x \mid \phi_x \in \mathcal{A}\}$ is r.e.

($\Rightarrow$): Suppose $f \in \mathcal{A}$ but for all finite $\theta \subseteq f.\theta \notin \mathcal{A}$.

Let $P$ be a partial characteristic function of $K$. Define the computable function $g(z,t)$ by

$$g(z,t) \simeq \begin{cases} f(t), & \text{if } P(z) \not\downarrow \text{ in } t \text{ steps,} \\ \uparrow, & \text{otherwise.} \end{cases}$$

According to S-m-n Theorem, there is an injective primitive recursive function $s(z)$ such that $g(z,t) \simeq \phi_{s(z)}(t)$.

By construction $\phi_{s(z)} \subseteq f$ for all $z$.

$z \in K \Rightarrow \phi_{s(z)}$ is finite $\Rightarrow s(z) \notin A$;
$z \notin K \Rightarrow \phi_{s(z)} = f \Rightarrow s(z) \in A$.

# Proof of Rice-Shapiro Theorem

($\Leftarrow$): Suppose $f$ is a computable function and there is a finite $\theta \in \mathcal{A}$ such that $\theta \subseteq f$ and $f \notin \mathcal{A}$.

Define the computable function $g(z,t)$ by

$$g(z,t) \simeq \begin{cases} f(t), & \text{if } t \in Dom(\theta) \vee z \in K, \\ \uparrow, & \text{otherwise.} \end{cases}$$

According to S-m-n Theorem, there is an injective primitive recursive function $s(z)$ such that $g(z,t) \simeq \phi_{s(z)}(t)$.

$z \in K \Rightarrow \phi_{s(z)} = f \Rightarrow s(z) \notin A$;
$z \notin K \Rightarrow \phi_{s(z)} = \theta \Rightarrow s(z) \in A$.

# Reversing Rice-Shapiro Theorem

$\{x \mid \phi_x \in \mathcal{A}\}$ is r.e. if the following hold:

1. $\Theta = \{e(\theta) \mid \theta \in \mathcal{A} \text{ and } \theta \text{ is finite}\}$ is r.e., where $e$ is a canonical effective encoding of the finite functions.
2. $\forall f \in \mathcal{A}.\exists \text{ finite } \theta \in \mathcal{A}.\theta \subseteq f$.

**Comment**. We cannot take $e$ as the Gödel encoding function of the recursive functions. Why? How would you define $e$?

# Homework

- Homework 6: Exercise 6.14, pp. 119 of the textbook.

# Computability Theory XII

## Creative Set

Guoqiang Li

Shanghai Jiao Tong University

Dec. 26, 2014

Creative Set

# Most Difficult Semi-Decidable Problems?

An r.e. set is very difficult if it is very non-recursive.

An r.e. set is very non-recursive if its complement is very non-r.e..

A set is very non-r.e. if it is easy to distinguish it from any r.e. set.

These sets are creative respectively productive.

# Synopsis

1. Productive Set
2. Creative Set
3. Simple Sets

# Productive Set

# Productive Set

Suppose $W_x \subseteq \overline{K}$. Then $x \in \overline{K} \setminus W_x$.

So $x$ witnesses the strict inclusion $W_x \subseteq \overline{K}$.

In other words the identity function is an effective proof that $\overline{K}$ differs from every r.e. set.

# Productive Set

A set $A$ is productive if there is a total computable function $p$ such that whenever $W_x \subseteq A$, then $p(x) \in A \setminus W_x$. The function $p$ is called a productive function for $A$.

A productive set is not r.e. by definition.

# Productive Set

$\overline{K}$ is productive.

$\{x \mid c \notin W_x\}$ is productive.

$\{x \mid c \notin E_x\}$ is productive.

$\{x \mid \phi_x(x) \neq 0\}$ is productive.

# Productive Set

Suppose $A = \{x \mid \phi_x(x) \neq 0\}$.

By S-m-n Theorem one gets a primitive recursive function $p(x)$ such that $\phi_{p(x)}(y) = 0$ if and only if $\phi_x(y)$ is defined. Then

$$p(x) \in W_x \Leftrightarrow p(x) \notin A.$$

So if $W_x \subseteq A$ we must have $p(x) \in A \setminus W_x$.
Thus $p$ is a productive function for $A$.

# Productive Set

**Lemma**. If $A \leq_m B$ and $A$ is productive, then $B$ is productive.

# Productive Set

**Theorem**. Suppose that $\mathcal{B}$ is a set of unary computable functions with $f_\emptyset \in \mathcal{B}$ and $\mathcal{B} \neq \mathcal{C}$. Then $B = \{x \mid \phi_x \in \mathcal{B}\}$ is productive.

*Proof*. Suppose $g \notin \mathcal{B}$. Consider the function $f$ defined by

$$f(x,y) = \begin{cases} g(y), & \text{if } x \in W_x, \\ \uparrow, & \text{if } x \notin W_x. \end{cases}$$

By S-m-n Theorem there is a primitive recursive function $k(x)$ such that $\phi_{k(x)}(y) \simeq f(x,y)$.

Clearly $x \notin W_x$ iff $\phi_{k(x)} = f_\emptyset$ iff $\phi_{k(x)} \in \mathcal{B}$ iff $k(x) \in B$.

Hence $k : \overline{K} \leq_m B$.

# Property of Productive Set

**Lemma**. Suppose that $g$ is a total computable function. Then there is a primitive recursive function $p$ such that for all $x$,
$W_{p(x)} = W_x \cup \{g(x)\}$.

*Proof*. Using S-m-n Theorem, take $p(x)$ to be a primitive recursive function such that

$$\phi_{p(x)}(y) = \begin{cases} 1, \text{if } y \in W_x \vee y = g(x), \\ \uparrow, \text{otherwise} \end{cases}$$

# Property of Productive Set

**Theorem**. A productive set contains an infinite r.e. subset.

*Proof*. Suppose $p$ is a production function for $A$.

Take $e_0$ to be some index for $\emptyset$. Then $p(e_0) \in A$ by definition.

By the Lemma there is a primitive recursive function $k$ such that for all $x$, $W_{k(x)} = W_x \cup \{p(x)\}$.

Apparently $\{e_0, \ldots, k^n(e_0), \ldots\}$ is r.e.

Consequently $\{p(e_0), \ldots, p(k^n(e_0)), \ldots\}$ is a r.e. subset of $A$, which must be infinite by the definition of $k$.

# Productive Function via a Partial Function

**Proposition**. A set $A$ is productive iff there is a partial recursive function $p$ such that

$$\forall x.(W_x \subseteq A \Rightarrow (p(x)\downarrow \wedge p(x) \in A \setminus W_x)). \tag{1}$$

*Proof*. Suppose $p$ is a partial recursive function satisfying (1). Let $s$ be a primitive recursive function such that

$$\phi_{s(x)}(y) \quad = \quad \begin{cases} y, & p(x)\downarrow \wedge\, y \in W_x, \\ \uparrow, & \text{otherwise.} \end{cases}$$

A productive function $q$ can be defined by running $p(x)$ and $p(s(x))$ in parallel and stops when either terminates.

# Productive Function Made Injective

**Proposition**. A productive set has an injective productive function.

*Proof*. Suppose $p$ is a productive function of $A$. Let

$$W_{h(x)} = W_x \cup \{p(x)\}$$

Clearly

$$W_x \subseteq A \Rightarrow W_{h(x)} \subseteq A$$

Define $q(0) = p(0)$.

If there is a least $y \in \{p(x+1), ph(x+1), ph^2(x+1), \ldots\}$ such that $y \notin \{q(0), \ldots, q(x)\}$, let $q(x+1)$ be $y$;

otherwise let $q(x+1)$ be $\mu y.y \notin \{q(0), \ldots, q(x)\}$.

It is easily seen that $q$ is an injective production function for $A$.

# Myhill's Characterization of Productive Set

**Fact**. $\overline{K} \leq_m A$ iff $\overline{K} \leq_1 A$.

# Myhill's Characterization of Productive Set

**Theorem**. (Myhill, 1955) $A$ is productive iff $\overline{K} \leq_1 A$ iff $\overline{K} \leq_m A$.

Creative Set

# Creative Set

A set $A$ is creative if it is r.e. and its complement $\overline{A}$ is productive.

Intuitively a creative set $A$ is effectively non-recursive in the sense that the non-recursiveness of $\overline{A}$, hence the non-recursiveness of $A$, can be effectively demonstrated.

# Creative Set

$K$ is creative.

$\{x \mid c \in W_x\}$ is creative.

$\{x \mid c \in E_x\}$ is creative.

$\{x \mid \phi_x(x) = 0\}$ is creative.

# Creative Set

**Theorem**. Suppose that $\mathcal{A} \subseteq \mathcal{C}$ and let $A = \{x \mid \phi_x \in \mathcal{A}\}$. If $A$ is r.e. and $A \neq \emptyset, \mathbb{N}$, then $A$ is creative.

*Proof*. Suppose $A$ is r.e. and $A \neq \emptyset, \mathbb{N}$. If $f_\emptyset \in \mathcal{A}$, then $A$ is productive by a previous theorem. This is a contradiction.

So $\overline{A}$ is productive by the same theorem. Hence $A$ is creative.

# Creative Set

The set $K_0 = \{x \mid W_x \neq \emptyset\}$ is creative. It corresponds to the set $\mathcal{A} = \{f \in \mathcal{C} \mid f \neq f_\emptyset\}$.

# Discussion

**Question**. Are all non-recursive r.e. sets creative?

The answer is negative. By a special construction we can obtain r.e.sets that are neither recursive nor creative.

# Simple Sets

# Simple Sets

A set $A$ is simple if

1. $A$ is r.e.,
2. $\overline{A}$ is infinite,
3. $\overline{A}$ contains no infinite r.e. subset.

# Simple Sets

**Theorem**. A simple set is neither recursive nor creative.

*Proof*. Since $\overline{A}$ can not be r.e., $A$ can not be recursive.

(3) implies that $A$ can not be creative.

# Simple Sets

**Theorem**. There is a simple set.

*Proof*. Define $f(x) = \phi_x(\mu z(\phi_x(z) > 2x))$. Let $A$ be $Ran(f)$.

1. $A$ is r.e.
2. $\overline{A}$ is infinite. This is because $A \cap \{0, 1, \ldots, 2n\}$ contains at most the elements $\{f(0), f(1), \ldots, f(n-1)\}$.
3. Suppose $B$ is an infinite r.e. set. Then there is a total computable function $\phi_b$ such that $B = E_b$. Since $\phi_b$ is total, $f(b)$ is defined and $f(b) \in A$. Hence $B \not\subseteq \overline{A}$.

# Assignment (I)

*Due*: Oct. 24, 2014

## Problem 1

Let $f(0) = 0$, $f(1) = 1$, $f(2) = 2^2$, $f(3) = 3^{3^3}$, etc. In general, $f(n)$ is written as a stack $n$ high, of $n$'s as exponents. Show that $f$ is primitive recursive.

## Problem 2

Show $f(x) = x^2$ is URM-computable by devising a URM program to compute it.

## Problem 3

Show that the *Euler's function* $\phi(x)$ is primitive recursive, where $\phi(x)$ is defined by "$\phi(x)$ = the number of positive integers less than $x$ which are relatively prime to $x$".

## Problem 4

Let

$$\begin{aligned} h_1(x,0) &= f_1(x), \\ h_2(x,0) &= f_2(x), \\ h_1(x,t+1) &= g_1(x, h_1(x,t), h_2(x,t)), \\ h_2(x,t+1) &= g_2(x, h_1(x,t), h_2(x,t)). \end{aligned}$$

Prove that if $f_1, f_2, g_1, g_2$ are recursive, then $h_1$, $h_2$ are also.

## Problem 5

Construct a total computable function that is non-primitive recursive by diagonalization.

# Assignment (II)

*Due*: Nov. 7, 2014

## Problem 1

Design a one tap Turing Machine that accepts the language $L = \{a^n b^n c^n : n \geq 1\}$.

## Problem 2

Describe an algorithm that transforms a TM to a URM.

## Problem 3

Suppose that $f(x)$ and $g(x)$ are effectively computable functions. Prove, using Church-Turing Thesis, that the function $h$ given by

$$h(x) = \begin{cases} 1 & \text{if } x \in Dom(f) \cap Dom(g), \\ \uparrow & \text{o.w.} \end{cases}$$

is computable.

## Problem 4

Suppose that $f$ is a total unary computable function. Prove by Church-Turing Thesis, that the following function $h$ is computable.

$$h(x) = \begin{cases} 1 & \text{if } x \in Ran(f), \\ \uparrow & \text{o.w.} \end{cases}$$

## Problem 5

Let $f$ be a unary computable function and $Dom(f) \neq \emptyset$, show that there is a total computable function $g$ s.t. $Ran(g) = Dom(f)$.

# Assignment (III)

*Due*: Dec. 12, 2014

## Problem 1

a). Show that the set of all functions form $\mathbb{N}$ to $\mathbb{N}$ is not denumerable. b). Show that the set of all non-computable total functions from $\mathbb{N}$ to $\mathbb{N}$ is not denumerable.

## Problem 2

Show that there is a total computable function $s(x, y)$ s.t. for all $x$, $y$, $E_{s(x,y)} = E_x \cup E_y$.

## Problem 3

Show that for each $m$ there is a total $(m+1)$-ary computable function $s^m$ such that for all $n$

$$\phi_e^{m+n}(\mathbf{x}, \mathbf{y}) \simeq \phi_{s^m(e,\mathbf{x})}(\mathbf{y})$$

where $\mathbf{x}$, $\mathbf{y}$ are $m$- and $n$-tuples respectively.

## Problem 4

Suppose that $f(x, y)$ is a computable function. Show that there is a total computable function $k(x)$ such that $f(x, y) \simeq \phi_{k(x)}(y)$ and $k(x) > x$ for all $x$.

## Problem 5

Show that there is a partial computable function $\psi(x)$ such that there is no total computable function $\tau(x, y)$ that validates the equality $\psi(x) \simeq \mu y.(\tau(x, y) = 1)$.

# Assignment (IV)

*Due*: Dec. 27, 2013

## Problem 1

Suppose that $f$ is a total computable function, A a recursive set and $B$ an r.e. set. Prove that $f^{-1}(A)$ is recursive and that $f(A)$, $f(B)$ and $f^{-1}(B)$ are r.e. but not necessary recursive. What extra information about these sets can be obtained if $f$ is a bijection?

## Problem 2

Suppose that $A$ is an r.e. set. Show that the sets $\bigcup_{x\in A} W_x$ and $\bigcup_{x\in A} E_x$ are both r.e. Show that $\bigcap_{x\in A} W_x$ is not necessary r.e.

## Problem 3

Let $f$ be unary function. Prove tat $f$ is computable iff the set $\{2^x 3^{f(x)} \mid x \in \mathrm{Dom}(f)\}$ is r.e.

## Problem 4

Show by diagonalization the set $A$ defined as follows is not r.e.

$$A = \{x \mid \phi_x \text{ is a total increasing function}\}$$

## Problem 5

Define set $A$ and $B$ as follows, show that $A$ is r.e. and $B$ is recursive.

1. $A = \{x \mid$ there is a run of *exactly* $x$ consecutive 7s in the decimal expansion of $\pi\}$.

2. $B = \{x \mid$ there is a run of *at least* $x$ consecutive 7s in the decimal expansion of $\pi\}$.

# Assignments

November 7, 2014

Assignment I

Assignment II

# Problem 1

Let $f(0) = 0$, $f(1) = 1$, $f(2) = 2^2$, $f(3) = 3^{3^3}$, etc.
In general, $f(n)$ is written as a stack $n$ high, of $n$'s as exponents.
Show that $f$ is primitive recursive.

# Problem 1

Let $f(0) = 0$, $f(1) = 1$, $f(2) = 2^2$, $f(3) = 3^{3^3}$, etc.
In general, $f(n)$ is written as a stack $n$ high, of $n$'s as exponents.
Show that $f$ is primitive recursive.

## Solution

*Define $g(x, y)$ as follows*

$$\begin{aligned} g(x, 0) &= 1 \\ g(x, y+1) &= x^{g(x,y)} \end{aligned}$$

# Problem 1

Let $f(0) = 0$, $f(1) = 1$, $f(2) = 2^2$, $f(3) = 3^{3^3}$, etc.
In general, $f(n)$ is written as a stack $n$ high, of $n$'s as exponents.
Show that $f$ is primitive recursive.

## Solution

*Define $g(x, y)$ as follows*

$$\begin{aligned} g(x, 0) &= 1 \\ g(x, y+1) &= x^{g(x,y)} \end{aligned}$$

$f(x) = sg(x)(g(x, x))$ *is primitive recursive.*

# Problem 2

Show $f(x) = x^2$ is URM-computable by devising a URM program to compute it.

# Problem 2

Show $f(x) = x^2$ is URM-computable by devising a URM program to compute it.

## Solution

1. **J(1,2,8)** *: outer loop,* $R_2$ *from* 0 *to* $x$
2. **S(2)**
3. **Z(3)** *:* $R_3$ *is temp*
4. **J(1,3,1)** *: inner loop,* $R_3$ *from* 0 *to* $x$
5. **S(3)**
6. **S(4)**
7. **J(1,1,4)**
8. **T(4,1)**

# Problem 2

Show $f(x) = x^2$ is URM-computable by devising a URM program to compute it.

## Solution

1. **J(1,2,8)** *: outer loop,* $R_2$ *from* 0 *to* $x$
2. **S(2)**
3. **Z(3)** *:* $R_3$ *is temp*
4. **J(1,3,1)** *: inner loop,* $R_3$ *from* 0 *to* $x$
5. **S(3)**
6. **S(4)**
7. **J(1,1,4)**
8. **T(4,1)**

*Test the program on input* 0 *and* 1.

# Problem 3

Show that the Euler's function $\phi(x)$ is primitive recursive, where $\phi(x)$ is defined by "$\phi(x)$ = the number of positive integers less than $x$ which are relatively prime to $x$".

# Problem 3

Show that the Euler's function $\phi(x)$ is primitive recursive, where $\phi(x)$ is defined by "$\phi(x)$ = the number of positive integers less than $x$ which are relatively prime to $x$".

## Solution

- $gcd(x, y) = x \dot{-} \mu z < (x + 1)(div(x - z, x) \wedge div(x - z, y))$

# Problem 3

Show that the Euler's function $\phi(x)$ is primitive recursive, where $\phi(x)$ is defined by "$\phi(x)$ = the number of positive integers less than $x$ which are relatively prime to $x$".

## Solution

- $gcd(x, y) = x \dot{-} \mu z < (x + 1)(div(x - z, x) \wedge div(x - z, y))$
- $rp(x, y) = \overline{sg}(|gcd(x, y) - 1|)$

# Problem 3

Show that the Euler's function $\phi(x)$ is primitive recursive, where $\phi(x)$ is defined by "$\phi(x)$ = the number of positive integers less than $x$ which are relatively prime to $x$".

## Solution

- $gcd(x, y) = x \dot{-} \mu z < (x + 1)(div(x - z, x) \wedge div(x - z, y))$
- $rp(x, y) = \overline{sg}(|gcd(x, y) - 1|)$
- $\phi(x) = \sum_{y \leq x} rp(x, y)$

# Problem 4

Let

$$\begin{aligned} h_1(x,0) &= f_1(x), \\ h_2(x,0) &= f_2(x), \\ h_1(x,t+1) &= g_1(x, h_1(x,t), h_2(x,t)), \\ h_2(x,t+1) &= g_2(x, h_1(x,t), h_2(x,t)). \end{aligned}$$

Prove that if $f_1, f_2, g_1, g_2$ are recursive, then $h_1$, $h_2$ are also.

# Problem 4

Let

$$\begin{aligned} h_1(x,0) &= f_1(x), \\ h_2(x,0) &= f_2(x), \\ h_1(x,t+1) &= g_1(x, h_1(x,t), h_2(x,t)), \\ h_2(x,t+1) &= g_2(x, h_1(x,t), h_2(x,t)). \end{aligned}$$

Prove that if $f_1, f_2, g_1, g_2$ are recursive, then $h_1$, $h_2$ are also.

## Solution

*Define a new function $H(x,t)$ as follows.*

$$\begin{aligned} H(x,0) &= 2^{f_1(x)} 3^{f_2(x)} \\ H(x,t+1) &= 2^{g_1(x,(H(x,t))_1,H(x,t)_2)} 3^{g_2(x,(H(x,t))_1,H(x,t)_2)} \end{aligned}$$

# Problem 4

Let

$$\begin{aligned} h_1(x,0) &= f_1(x), \\ h_2(x,0) &= f_2(x), \\ h_1(x,t+1) &= g_1(x,h_1(x,t),h_2(x,t)), \\ h_2(x,t+1) &= g_2(x,h_1(x,t),h_2(x,t)). \end{aligned}$$

Prove that if $f_1, f_2, g_1, g_2$ are recursive, then $h_1$, $h_2$ are also.

## Solution

*Define a new function $H(x,t)$ as follows.*

$$\begin{aligned} H(x,0) &= 2^{f_1(x)}3^{f_2(x)} \\ H(x,t+1) &= 2^{g_1(x,(H(x,t))_1,H(x,t)_2)}3^{g_2(x,(H(x,t))_1,H(x,t)_2)} \end{aligned}$$

*$2^x3^y$, $(x)_1$, $(x)_2$, $f_1, f_2, g_1, g_2$ are recursive implies $H(x,t)$ is recursive.*

*$h_1(x,t) = (H(x,t))_1$ and $h_2(x,t) = (H(x,t))_2$ are recursive.*

# Problem 5

Construct a total computable function that is non primitive recursive by diagonalization.

# Problem 5

Construct a total computable function that is non primitive recursive by diagonalization.

## Solution

*We can effectively enumerate all primitive recursive functions. Let $f$ be that enumerator and $f_i$ be the $i$-th function enumerated by $f$, then define a new function $g$ s.t.*

$$g(x) = f_x(x) + 1$$

*$g$ is total computable but $\forall i.g(i) \neq f_i(i)$.*

# Problem 1

Design a one tap Turing Machine that accepts the language $L = \{a^n b^n c^n : n \geq 1\}$.

## Solution

*We define a one tap Turing Machine* $\mathcal{M} = \{\Gamma, Q, \delta\}$ *such that whenever a word* $S \in \{a, b, c\}^*$ *is written on the tap,* $\mathcal{M}$ *stops in finite steps. If* $S = a^n b^n c^n$ *for some* $n \geq 1$*, then when* $\mathcal{M}$ *stops it writes* $1$ *on the tap. Otherwise it writes* $0$ *on the tap.*

- $\Gamma = (\triangleright, \triangleleft, 0, 1, a, b, c)$
- $Q = \{s, p_{\triangleright}, p_{\triangleleft}, p_a, p'_a, p_b, p'_b, p_c, p'_c, p_f, p'_f, p_{\square}, p'_{\square}, p^0_h, p^1_h, h\}$
- Definition of $\delta$ is given as follows.

- Set Boundary

| $(*, *)$ | $\delta(*, *)$ |
|---|---|
| $(s, \triangleright)$ | $(p_{\triangleleft}, \triangleright, \rightarrow)$ |
| $(p_{\triangleleft}, a)$ | $(p_{\triangleleft}, a, \rightarrow)$ |
| $(p_{\triangleleft}, b)$ | $(p_{\triangleleft}, b, \rightarrow)$ |
| $(p_{\triangleleft}, c)$ | $(p_{\triangleleft}, c, \rightarrow)$ |
| $(p_{\triangleleft}, \square)$ | $(p_c, \triangleleft, \leftarrow)$ |

- Do counting.

| $(p_c, *)$ | $\delta(p_c, *)$ | $(p'_c, *)$ | $\delta(p'_c, *)$ |
|---|---|---|---|
| $(p_c, a)$ | $(p_f, \square, \uparrow)$ | $(p'_c, a)$ | $(p_f, \square, \uparrow)$ |
| $(p_c, b)$ | $(p_f, \square, \uparrow)$ | $(p'_c, b)$ | $(p_b, b, \uparrow)$ |
| $(p_c, c)$ | $(p'_c, \square, \leftarrow)$ | $(p'_c, c)$ | $(p'_c, c, \leftarrow)$ |
| $(p_c, \square)$ | $(p_c, \square, \leftarrow)$ | $(p'_c, \square)$ | $(p'_c, \square, \leftarrow)$ |
| $(p_c, \triangleright)$ | $(p_f, \triangle, \uparrow)$ | $(p'_c, \triangleright)$ | $(p_f, \triangleright, \uparrow)$ |
| $(p_b, *)$ | $\delta(p_b, *)$ | $(p'_b, *)$ | $\delta(p'_b, *)$ |
| $(p_b, a)$ | $(p_f, \square, \uparrow)$ | $(p'_b, a)$ | $(p_a, a, \uparrow)$ |
| $(p_b, b)$ | $(p'_b, \square, \leftarrow)$ | $(p'_b, b)$ | $(p_b, b, \leftarrow)$ |
| $(p_b, c)$ | $(p_f, \square, \uparrow)$ | $(p'_b, c)$ | $(p_f, \square, \uparrow)$ |
| $(p_b, \square)$ | $(p_b, \square, \uparrow)$ | $(p'_b, \square)$ | $(p'_b, \square, \leftarrow)$ |
| $(p_b, \triangleright)$ | $(p_f, \triangle, \uparrow)$ | $(p'_b, \triangleright)$ | $(p_f, \triangleright, \uparrow)$ |
| $(p_a, *)$ | $\delta(p_a, *)$ | $(p'_a, *)$ | $\delta(p'_a, *)$ |
| $(p_a, a)$ | $(p'_a, \square, \leftarrow)$ | $(p'_a, a)$ | $(p'_a, a, \leftarrow)$ |
| $(p_a, b)$ | $(p_f, \square, \uparrow)$ | $(p'_a, b)$ | $(p_f, \square, \uparrow)$ |
| $(p_a, c)$ | $(p_f, \square, \uparrow)$ | $(p'_a, c)$ | $(p_f, \square, \uparrow)$ |
| $(p_a, \square)$ | $(p_a, \square, \leftarrow)$ | $(p'_a, \square)$ | $(p'_a, \square, \leftarrow)$ |
| $(p_a, \triangleright)$ | $(p_f, \triangle, \uparrow)$ | $(p'_a, \triangleright)$ | $(p_\square, \triangleright, \rightarrow)$ |

- Clean up

| $(p_f, *)$ | $\delta(p_f, *)$ | $(p'_f, *)$ | $\delta(p'_f, *)$ |
|---|---|---|---|
| $(p_f, a)$ | $(p_f, \square, \rightarrow)$ | $(p'_f, a)$ | $(p'_f, \square, \leftarrow)$ |
| $(p_f, b)$ | $(p_f, \square, \rightarrow)$ | $(p'_f, b)$ | $(p'_f, \square, \leftarrow)$ |
| $(p_f, c)$ | $(p_f, \square, \rightarrow)$ | $(p'_f, c)$ | $(p'_f, \square, \leftarrow)$ |
| $(p_f, \square)$ | $(p_f, \square, \rightarrow)$ | $(p'_f, \square)$ | $(p'_f, \square, \leftarrow)$ |
| $(p_f, \triangleleft)$ | $(p_f, \triangleleft, \rightarrow)$ | $(p'_f, \triangleright)$ | $(p^0_h, \triangleright, \rightarrow)$ |
| $(p_f, \triangleright)$ | $(p'_f, \square, \leftarrow)$ | $(p'_f, \triangleleft)$ | $(p'_f, \square, \leftarrow)$ |
| $(p_\square, *)$ | $\delta(p_\square, *)$ | $(p'_\square, *)$ | $\delta(p'_\square, *)$ |
| $(p_\square, a)$ | $(p_\triangleleft, a, \rightarrow)$ | $(p'_\square, a)$ | $(p_\square, \square, \leftarrow)$ |
| $(p_\square, b)$ | $(p_\triangleleft, b, \rightarrow)$ | $(p'_\square, b)$ | $(p_\square, \square, \leftarrow)$ |
| $(p_\square, c)$ | $(p_\triangleleft, c, \rightarrow)$ | $(p'_\square, c)$ | $(p_\square, \square, \leftarrow)$ |
| $(p_\square, \square)$ | $(p_\square, \square, \rightarrow)$ | $(p'_\square, \square)$ | $(p_\square, \square, \leftarrow)$ |
| $(p_\square, \triangleleft)$ | $(p'_\square, \square, \leftarrow)$ | $(p'_\square, \triangleright)$ | $(p^1_h, \triangleright, \rightarrow)$ |

- Clean up

| $(p_f, *)$ | $\delta(p_f, *)$ | $(p'_f, *)$ | $\delta(p'_f, *)$ |
|---|---|---|---|
| $(p_f, a)$ | $(p_f, \square, \rightarrow)$ | $(p'_f, a)$ | $(p'_f, \square, \leftarrow)$ |
| $(p_f, b)$ | $(p_f, \square, \rightarrow)$ | $(p'_f, b)$ | $(p'_f, \square, \leftarrow)$ |
| $(p_f, c)$ | $(p_f, \square, \rightarrow)$ | $(p'_f, c)$ | $(p'_f, \square, \leftarrow)$ |
| $(p_f, \square)$ | $(p_f, \square, \rightarrow)$ | $(p'_f, \square)$ | $(p'_f, \square, \leftarrow)$ |
| $(p_f, \triangleleft)$ | $(p_f, \triangleleft, \rightarrow)$ | $(p'_f, \triangleright)$ | $(p^0_h, \triangleright, \rightarrow)$ |
| $(p_f, \triangleright)$ | $(p'_f, \square, \leftarrow)$ | $(p'_f, \triangleleft)$ | $(p'_f, \square, \leftarrow)$ |
| $(p_\square, *)$ | $\delta(p_\square, *)$ | $(p'_\square, *)$ | $\delta(p'_\square, *)$ |
| $(p_\square, a)$ | $(p_\triangleleft, a, \rightarrow)$ | $(p'_\square, a)$ | $(p_\square, \square, \leftarrow)$ |
| $(p_\square, b)$ | $(p_\triangleleft, b, \rightarrow)$ | $(p'_\square, b)$ | $(p_\square, \square, \leftarrow)$ |
| $(p_\square, c)$ | $(p_\triangleleft, c, \rightarrow)$ | $(p'_\square, c)$ | $(p_\square, \square, \leftarrow)$ |
| $(p_\square, \square)$ | $(p_\square, \square, \rightarrow)$ | $(p'_\square, \square)$ | $(p_\square, \square, \leftarrow)$ |
| $(p_\square, \triangleleft)$ | $(p'_\square, \square, \leftarrow)$ | $(p'_\square, \triangleright)$ | $(p^1_h, \triangleright, \rightarrow)$ |

- Output

| $(*, *)$ | $\delta(*, *)$ |
|---|---|
| $(p^0_h, \square)$ | $(h, 0, \uparrow)$ |
| $(p^1_h, \square)$ | $(h, 1, \uparrow)$ |

# Problem 2

Describe an algorithm that transforms a TM to a URM.

# Problem 2

Describe an algorithm that transforms a TM to a URM.

## Solution

*Assume that the Turing Machine is a one tap TM.* $R_1$ *encodes the tape,* $R_2$ *encodes the head position and* $R_3$ *encodes the content in the head position.*
*The URM program is series of implementations of* $\delta$ *function of TM.*

# Problem 3

Suppose that $f(x)$ and $g(x)$ are effectively computable functions. Prove, using Church-Turing Thesis, that the function $h$ given by

$$h(x) = \begin{cases} 1 & \text{if } x \in Dom(f) \cap Dom(g), \\ \uparrow & \text{o.w.} \end{cases}$$

is computable.

# Problem 3

Suppose that $f(x)$ and $g(x)$ are effectively computable functions. Prove, using Church-Turing Thesis, that the function $h$ given by

$$h(x) = \begin{cases} 1 & \text{if } x \in Dom(f) \cap Dom(g), \\ \uparrow & \text{o.w.} \end{cases}$$

is computable.

## Solution

*We compute h as follows:*

- *Let $P_f$ and $P_g$ computes f and g respectively.*
- *Run $P_f$ and $P_g$ on input x simultaneously.*
- *If $P_f$ and $P_g$ stops then output 1.*

*By Church-Turing Thesis, h is computable.*

# Problem 4

Suppose that $f$ is a total unary computable function. Prove by Church-Turing Thesis, that the following function $h$ is computable.

$$h(x) = \begin{cases} 1 & \text{if } x \in Ran(f), \\ \uparrow & \text{o.w.} \end{cases}$$

# Problem 4

Suppose that $f$ is a total unary computable function. Prove by Church-Turing Thesis, that the following function $h$ is computable.

$$h(x) = \begin{cases} 1 & \text{if } x \in Ran(f), \\ \uparrow & \text{o.w.} \end{cases}$$

## Solution

*Let $P$ be a program that compute $f$, we can compute $h$ as follows. Run $P(0)$, $P(1)$, ... in turn. If $P(i)$ stops with result $x$ then output $1$ otherwise continue to run $P(i+1)$. By Church-Turing Thesis, $h$ is computable.*

# Problem 5

Let $f$ be a unary computable function and $Dom(f) \neq \emptyset$, show that there is a total computable function $g$ s.t. $Ran(g) = Dom(f)$.

# Problem 5

Let $f$ be a unary computable function and $Dom(f) \neq \emptyset$, show that there is a total computable function $g$ s.t. $Ran(g) = Dom(f)$.

## Solution

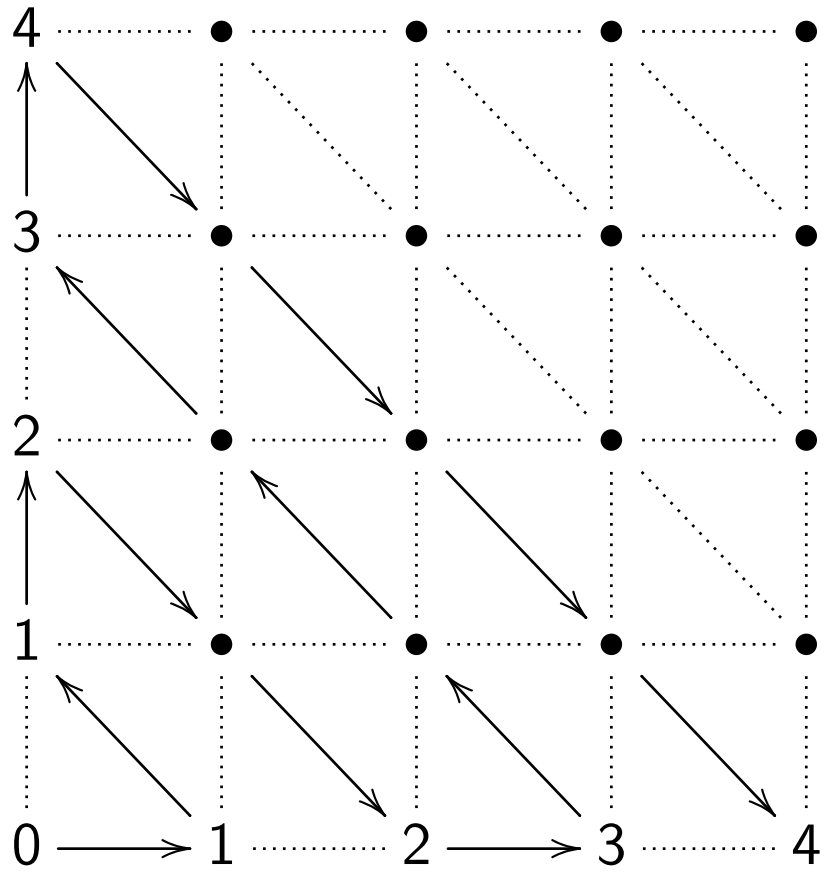



Figure: An effective bijection between $\mathbb{N}$ and $\mathbb{N} \times \mathbb{N}$

Let $n$ be an element in $Dom(f)$ ($Dom(f) \neq \emptyset$).
$Q(x)$ is defined as follows.

---

1. Find a pair $(i,j)$ s.t. $m(x) = (i,j)$.
2. Simulate $P(i)$ $j$ steps.
3. If $P(i)$ stops in $j$ steps then output $i$ else ouput $n$.

---

Let $g$ be the function $Q$ computes, then $Ran(g) = Dom(f)$.

- Clearly $Ran(g) \subseteq Dom(f)$ since every output of $Q$ is defined by $f$.
- For every $i \in Dom(f)$, there is a least number $j$ s.t. $P(i)$ stops in $j$ steps. Let $x = m(i,j)$, then by construction $Q(x)$ will stop and output $i$.

# Assignments

November 7, 2014

## Assignment III

## Assignment IV

# Problem 1

a). Show that the set of all functions form $\mathbb{N}$ to $\mathbb{N}$ is not denumerable.
b). Show that the set of all non-computable total functions from $\mathbb{N}$ to $\mathbb{N}$ is not denumerable.

# Problem 1

a). Show that the set of all functions form $\mathbb{N}$ to $\mathbb{N}$ is not denumerable.
b). Show that the set of all non-computable total functions from $\mathbb{N}$ to $\mathbb{N}$ is not denumerable.

## Solution

*a)& b) Counting. The cardinality of the set* $\{f|f : \mathbb{N} \to \mathbb{N}\}$ *is* $2^{\mathbb{N}}$

# Problem 2

Show that there is a total computable function $s(x, y)$ s.t. for all $x$, $y$, $E_{s(x,y)} = E_x \cup E_y$.

# Problem 2

Show that there is a total computable function $s(x, y)$ s.t. for all $x$, $y$, $E_{s(x,y)} = E_x \cup E_y$.

## Solution

*Let*

$$f(x, y, z) = \begin{cases} z & z \in E_x \vee z \in E_y, \\ \uparrow & o.w. \end{cases}$$

*By s-m-n theorem, there exists a total computable function* $s(x, y)$ *such that* $\phi_{s(x,y)}(z) = f(x, y, z)$.

# Problem 3

Show that for each $m$ there is a total $(m+1)$-ary computable function $s^m$ such that for all $n$

$$\phi_e^{m+n}(\mathbf{x}, \mathbf{y}) \simeq \phi_{s^m(e,\mathbf{x})}(\mathbf{y})$$

where $\mathbf{x}$, $\mathbf{y}$ are $m$- and $n$-tuples respectively.

# Problem 3

Show that for each $m$ there is a total $(m+1)$-ary computable function $s^m$ such that for all $n$

$$\phi_e^{m+n}(\mathbf{x}, \mathbf{y}) \simeq \phi_{s^m(e,\mathbf{x})}(\mathbf{y})$$

where $\mathbf{x}$, $\mathbf{y}$ are $m$- and $n$-tuples respectively.

## Solution

*The only way in which n was used was in determining how many of the* $r_1, r_2, \ldots$ *to* $R_{m+1}, R_{m+2}, \ldots$. *Now recall that the effect of* $P_e$ *depends only on the original contents of* $R_1, \ldots, R_{\rho(P_e)}$.

# Problem 4

Suppose that $f(x, y)$ is a computable function. Show that there is a total computable function $k(x)$ such that $f(x, y) \simeq \phi_{k(x)}(y)$ and $k(x) > x$ for all $x$.

# Problem 4

Suppose that $f(x, y)$ is a computable function. Show that there is a total computable function $k(x)$ such that $f(x, y) \simeq \phi_{k(x)}(y)$ and $k(x) > x$ for all $x$.

## Solution

$k(x) > x$ *by construction.*

# Problem 5

Show that there is a partial computable function $\psi(x)$ such that there is no total computable function $\tau(x, y)$ that validates the equality $\psi(x) \simeq \mu y.(\tau(x, y) = 1)$.

# Problem 5

Show that there is a partial computable function $\psi(x)$ such that there is no total computable function $\tau(x, y)$ that validates the equality $\psi(x) \simeq \mu y.(\tau(x, y) = 1)$.

## Solution

*Let*

$$\psi(x) = \begin{cases} 1 & \phi_x(x) \downarrow \\ \uparrow & o.w. \end{cases}$$

*Suppose there is a total computable function* $\tau(x, y)$ *s.t* $\psi(x) \simeq \mu y.(\tau(x, y) = 1)$. $\tau(x, 1) = 1$ *iff* $\psi(x) = 1$ *iff* $x \in K$, *while* $K$ *is not total computable.*

# Problem 1

Suppose that $f$ is a total computable function, A a recursive set and $B$ an r.e. set. Prove that $f^{-1}(A)$ is recursive and that $f(A)$, $f(B)$ and $f^{-1}(B)$ are r.e. but not necessary recursive. What extra information about these sets can be obtained if $f$ is a bijection?

# Problem 1

Suppose that $f$ is a total computable function, A a recursive set and $B$ an r.e. set. Prove that $f^{-1}(A)$ is recursive and that $f(A)$, $f(B)$ and $f^{-1}(B)$ are r.e. but not necessary recursive. What extra information about these sets can be obtained if $f$ is a bijection?

## Solution

$x \in f^{-1}(A)$ *iff* $f(x) \in A$ *is recursive.*
$x \in f(B)$ *iff* $\exists y \in B.x = f(y)$ *is r.e.*
$x \in f^{-1}(B)$ *iff* $f(x) \in B$ *is r.e..*

# Problem 2

Suppose that $A$ is an r.e. set. Show that the sets $\bigcup_{x \in A} W_x$ and $\bigcup_{x \in A} E_x$ are both r.e. Show that $\bigcap_{x \in A} W_x$ is not necessary r.e.

# Problem 2

Suppose that $A$ is an r.e. set. Show that the sets $\bigcup_{x \in A} W_x$ and $\bigcup_{x \in A} E_x$ are both r.e. Show that $\bigcap_{x \in A} W_x$ is not necessary r.e.

## Solution

*Church Turing Thesis*

# Problem 2

Suppose that $A$ is an r.e. set. Show that the sets $\bigcup_{x\in A} W_x$ and $\bigcup_{x\in A} E_x$ are both r.e. Show that $\bigcap_{x\in A} W_x$ is not necessary r.e.

## Solution

*Church Turing Thesis*

*For any $t$, let $K_t = \{x | P_x(x) \downarrow in\ t\ steps\}$, which is recursive.*

*$K = \bigcup_{t\in\mathbb{N}} K_t$ and $\bar{K} = \bigcap_{t\in\mathbb{N}} \bar{K}_t$.*

*There is a computable function $s$ such that $\bar{K}_t = W_s(t)$ for every $t$.*

# Problem 3

Let $f$ be unary function. Prove that $f$ is computable iff the set $\{2^x 3^{f(x)} \mid x \in \mathrm{Dom}(f)\}$ is r.e.

# Problem 3

Let $f$ be unary function. Prove that $f$ is computable iff the set $\{2^x 3^{f(x)} \mid x \in \mathrm{Dom}(f)\}$ is r.e.

## Solution

*"$\Rightarrow$" is trivial.*

*"$\Leftarrow$"*

*If $A = \{2^x 3^{f(x)} \mid x \in Dom(f)\}$ is r.e., then $y \in A$ is computable.$y \in A$ iff $y_1 = x$ and $y_2 = f(x)$. $f(x)$ is computable.*

# Problem 4

Show by diagonalization the set $A$ defined as follows is not r.e.

$$A = \{x \mid \phi_x \text{ is a total increasing function}\}$$

# Problem 4

Show by diagonalization the set $A$ defined as follows is not r.e.

$$A = \{x \mid \phi_x \text{ is a total increasing function}\}$$

## Solution

*Suppose to the contrary that $f$ is a total unary computable function such that enumerates this set;i.e. $\phi_{f(0)}, \phi_{f(1)}, \ldots$*
*Let $g(x) = \phi_{f(x)}(x) + g(x-1) + 1$, which is a total increasing function. But $g \neq \phi_{f(m)}$ for every $m$.*

# Problem 5

Define set $A$ and $B$ as follows, show that $A$ is r.e. and $B$ is recursive.

1. $A = \{x \mid$ there is a run of exactly $x$ consecutive 7s in the decimal expansion of $\pi\}$.
2. $B = \{x \mid$ there is a run of at least $x$ consecutive 7s in the decimal expansion of $\pi\}$.

# Problem 5

Define set $A$ and $B$ as follows, show that $A$ is r.e. and $B$ is recursive.

1. $A = \{x \mid$ there is a run of exactly $x$ consecutive 7s in the decimal expansion of $\pi\}$.
2. $B = \{x \mid$ there is a run of at least $x$ consecutive 7s in the decimal expansion of $\pi\}$.

## Solution

*1. Church Turing Thesis*

# Problem 5

Define set $A$ and $B$ as follows, show that $A$ is r.e. and $B$ is recursive.

1. $A = \{x \mid$ there is a run of exactly $x$ consecutive 7s in the decimal expansion of $\pi\}$.
2. $B = \{x \mid$ there is a run of at least $x$ consecutive 7s in the decimal expansion of $\pi\}$.

## Solution

1. *Church Turing Thesis*
2. *$B$ is finite or $\mathbb{N}$*

# Lab1-Computable Functions

October 17, 2014

1. Show that the following functions are computable by devising programs that computes them.

   (a) $f(x) = 5$;

   (b) $f(x,y) = \begin{cases} 0 & \text{if } x = y, \\ 1 & \text{if } x \neq y; \end{cases}$

   (c) $f(x,y) = \begin{cases} 0 & \text{if } x \leq y, \\ 1 & \text{if } x > y; \end{cases}$ If $f(x,y)$ is the $c_M$ of predicate $M$, what is $M$?

   (d) $f(x) = [3x/4]$ ($[z]$ denotes the greatest integer $\leq z$).

2. Suppose $P$ is a program without any jump instructions. Show that

   (a) there is a number $m$ such that either $\forall x : f_P^{(1)}(x) = m$, or $\forall x : f_P^{(1)}(x) = x + m$.

   (b) not every computable function is computable in this sense.

3. Show that for each transfer instruction $T(m,n)$ there is a program without any transfer instructions that has exactly the same effect as $T(m,n)$ on any configuration of the URM (Thus transfer instructions are really redundant in the formulation of our URM; it is nevertheless natural and convenient to have transfer as a basic facility of the URM).

4. Show that the following predicates are decidable.

   (a) '$x$ is even' on $\mathbb{Z}$ (Firstly design an URM to '$x$ is even' on $\mathbb{N}$, then give coding and $f^*$).

   (b) $x$ is a power of a prime number,

   (c) $x$ is a perfect cube.

   (d) $M(x,y) \equiv$ '$g(x) = y$', if $g(x)$ is a total computable function.

5. Without writing any programs, show that for all $m \in \mathbb{N}$ the following functions are computable:

   (a) $\mathbf{m}$ (recall that $\mathbf{m}(x) = m$, for all $x$).

   (b) $mx$.

   (c) $f(x,m)$, if $f(x,y)$ is computable.

6. Show that the following functions are computable:

   (a) Any polynomial function $a_0 + a_1x + \ldots + a_nx^n$, where $a_0, a_1, \ldots, a_n \in \mathbb{N}$,

   (b) $LCM(x,y) =$ the least common multiple of $x$ and $y$,

   (c) $HCF(x,y) =$ the highest common factor of $x$ and $y$,

   (d) $f(x) =$ the number of prime divisors of $x$,

   (e) $\phi(x) =$ the number of positive integers less than $x$ which are relatively prime to $x$. (*Euler's function*)(We say that $x, y$ are *relatively prime* if $HCF(x,y) = 1$.)

7. Let $\pi(x,y) = 2^x(2y+1) - 1$. Show that $\pi$ is a computable bijection from $\mathbb{N}^2$ to $\mathbb{N}$, and that the functions $\pi_1$, $\pi_2$ such that $\pi(\pi_1(z), \pi_2(z)) = z$ for all $z$ are computable.

8. Suppose $f(x)$ is defined by $\begin{cases} f(0) = 1, \\ f(1) = 1, \\ f(x+2) = f(x) + f(x+1). \end{cases}$ ($f(x)$ is the *Fibonacci* sequence)
Show that $f$ is computable (*Hint*: first show that the function $g(x) = 2^{f(x)}3^{f(x+1)}$ is computable, using recursion).

9. Any number $x \in \mathbb{N}$ has a unique expression as

   (a) $x = \sum_{i=0}^{\infty} \alpha_i 2^i$, with $\alpha_i = 0$ or $1$, all $i$. Hence, if $x > 0$, there are unique expressions for $x$ in the forms;

   (b) $x = 2^{b_1} + 2^{b_2} + \ldots + 2^{b_1}$, with $0 \leq b_1 < b_2 < ... < b_l$ and $l \geq 1$. And

   (c) $x = 2^{a_1} + 2^{a_1+a_2+1} + \ldots + 2^{a_1+a_2+\ldots+a_k+k-1}$.

   Putting

   $\alpha(i, x) = \alpha_i$ as in the expression (a);

   $$l(x) = \begin{cases} l \text{ as in (b)}, & \text{if } x > 0, \\ 0 & \text{otherwise;} \end{cases}$$

   $$b(i, x) = \begin{cases} b_i \text{ as in (b)}, & \text{if } x > 0 \text{ and } 1 \leq i \leq l, \\ 0 & \text{otherwise;} \end{cases}$$

   $$a(i, x) = \begin{cases} a_i \text{ as in (c)}, & \text{if } x > 0 \text{ and } 1 \leq i \leq l, \\ 0 & \text{otherwise;} \end{cases}$$

   Show that each of the functions $\alpha$, $l$, $b$, $a$ is computable (The expression (c) is a way of regarding $x$ as coding the sequence $(a_1, a_2, \ldots, a_l)$ of numbers).

10. Gadgets

In order to construct URM to perform complex operations, it is useful to build it from smaller components that we'll call *gadgets*, which perform specific operations. A gadget will be defined by a series of instructions and will operate on registers that are specified in the gadget's name. For instance, the gadget "predecessor $r_n$" denoted by $P(n)$ will subtract 1 from the contents of register $R_n$ if it is non-zero. It can be represented by an instruction sequence shown in the right block. For simplicity, when we obey gadget function $P(l)$, we by default obey $P^{-1}[l_1, \cdots, l_n \to l]$, meaning we will use registers $R_{l_1}, \cdots, R_{l_n}$ ($l_i > \rho(P)$, $\forall 1 \leq i \leq n$) and place the result in $R_l$, without any interference to the next instructions. Now answer the following questions:

Gadget $P(1)$

| | |
|---|---|
| $I_1$ | J(1,4,9) |
| $I_2$ | S(3) |
| $I_3$ | J(1,3,7) |
| $I_4$ | S(2) |
| $I_5$ | S(3) |
| $I_6$ | J(1,1,3) |
| $I_7$ | T(2,1) |

URM $Q$

| | |
|---|---|
| $I'_1$ | J(1,2,6) |
| $I'_2$ | S(2) |
| $I'_3$ | T(2,3) |
| $I'_4$ | M(2,3,4) |
| $I'_5$ | G(1,4,2) |
| $I'_6$ | H(2) |

   (a) Define a gadget "greater than $r_m > r_n$" denoted by $G(m, n, q)$, which determines whether the initial value of $R_m$ is greater than that of $R_n$. If yes, jump to the $q$th instruction, otherwise go on to the next instruction.

   (b) Define a gadget "halt with $r_n$" denoted by $H(n)$, which leaves $R_n$ with its initial value, and overwrites the initial values of other registers into 0 (write the instruction sequences).

   (c) Define a gadget "multiply $r_m$ by $r_n$ to $R_p$" denoted by $M(m, n, p)$, which multiplies $r_m$ by $r_n$ and stores the result in $R_p$.

   (d) Describe the function of one argument $f(x)$ computed by the program $Q$. (What is $f_Q^{(1)}$?)

# Lab2-Church's Thesis

November 13, 2014

1. Show the computability of the following functions by minimalisation.

   (a) $f^{-1}(x)$, if $f(x)$ is a total injective computable function.

   (b) $f(a) = \begin{cases} \text{least non-negative integral root of } p(x) - a (a \in \mathbb{N}), \\ \text{undefined if there's no such root,} \end{cases}$

   where $p(x)$ is a polynomial with integer coefficients.

   (c) $f(x, y) = \begin{cases} x/y & \text{if } y \neq 0 \text{ and } y|x, \\ \text{undefined} & \text{otherwise,} \end{cases}$

2. Devise Turing machines that will Turing-compute the functions: (a) $x \dot{-} 1$, and (b) $2x$.

3. Suggest natural definition of computability on domain (a) $3 \times 3$ matrices; and (b) $\mathbb{Q}$.

4. Prove that the following functions are URM-computable by Church's Thesis.

   (a) $h(x) = \begin{cases} x & \text{if } x \in Dom(f) \cap Dom(g), \\ \text{undefined} & \text{otherwise.} \end{cases}$

   where $f(x)$ and $g(x)$ are effectively computable functions.

   (b) $h(x) = \begin{cases} 1 & \text{if } x \in Ran(f), \\ \text{undefined} & \text{otherwise.} \end{cases}$ if $f$ is a total unary computable function.

   (c) Ackermann function $\psi(x, y)$.

   (d) $g(n) = n^{th}$ digit in the decimal expansion of $e$ ($e$ is the basis for natural logarithms).

5. Find the Gödel encoding (a) $\beta(J(3, 4, 2))$, (b) $\beta^{-1}(503)$, (c) $P_{100}$, and (d) The code number of program $P = \{T(3, 4), S(3), Z(1)\}$.

6. Let $f(x, y)$ be a total computable function. $\forall m$, let $g_m$ be the computable function given by $g_m(y) = f(m, y)$. Construct a total computable function $h$ such that for each $m$, $h \neq g_m$.

7. Let $f_0, f_1, \ldots$ be an enumeration of partial functions from $\mathbb{N}$ to $\mathbb{N}$. Construct a function $g$ from $\mathbb{N}$ to $\mathbb{N}$ such that $Dom(g) \neq Dom(f_i)$ for each $i$.

8. Let $f$ be a partial function from $\mathbb{N}$ to $\mathbb{N}$, and let $m \in \mathbb{N}$. Construct a non-computable function $g$ such that $g(x) \simeq f(x)$ for $x \leq m$.

9. (a) (Cantor) Show that the set of all functions from $\mathbb{N}$ to $\mathbb{N}$ is not denumerable.

   (b) Show that the set of all non-computable total functions from $\mathbb{N}$ to $\mathbb{N}$ is not denumerable.

10. Palindrome Recognization

    Assume $M$ is a $k$-tape Turing Machine, in which the first tape is the read-only input tape, the other $k - 1$ tapes are read/write work tapes, and the $k^{th}$ tape is also used as the output tape. Every tape comes with a read/write head. We define $L$, $R$, and $S$ as the action 'move left', 'move right', and 'stay' for a head. $M$ works on $k$ tapes simultaneously, and thus one instruction should be written as $\langle q_i, s_1, \cdots, s_k \rangle \rightarrow \langle q_l, s'_2, \cdots, s'_k, A_1, \cdots, A_k \rangle$, where each $s_i$ is the scanned symbol on the $i^{th}$ tape, $s'_i$ is the replaced symbol, and $A_i$ is the corresponding action. E.g., $\langle q_2, a, a \rangle \rightarrow \langle q_3, b, R, S \rangle$ works for $k = 2$. Give the specification of $M$ with $k = 3$ to recognize palindromes on symbol set $\{0, 1, \triangleright, \triangleleft, \square\}$ . (A palindrome is a word that reads the same both forwards and backwards. Example: anna, madam, nitalarbralatin. To recognize palindrome we need to check the input string, output 1 if the string is a palindrome, and 0 otherwise. Initially the head on each tape reads $\triangleright$ as the starting point, with state $q_S$).

# Lab3-Denumerability

December 8, 2014

1. Show that there is a total computable function $k$ such that for each $n$,
   (a) $k(n)$ is an index of the function $[\sqrt[y]{x}]$.
   (b) $W^{(m)}_{k(n)} = \{(y_1, \ldots, y_m) : y_1 + y_2 + \ldots + y_m = n\}$ $(m \geq 1)$.
   (c) $E_{k(n)} = W_n$.
   (d) $W_{k(n)} = f^{-1}(W_n)$, if $f(n)$ is computable.

2. (a) Show that for each $m$ there is a total $(m+1)$-ary computable function $s^m$ such that for all $n$, $\phi^{(m+n)}_e(\mathbf{x}, \mathbf{y}) \simeq \phi^{(n)}_{s^m(e,\mathbf{x})}(\mathbf{y})$, where $\mathbf{x}, \mathbf{y}$ are $m$- and $n$-tuples respectively.
   (*Hint.* Consider the definition of $s^m_n(e, \mathbf{x})$ given in the proof of Chapter 4-theorem 4.3. The only way in which $n$ was used was in determining how many of the $r_1, r_2, \ldots$ to transfer to $R_{m+1}, R_{m+2}, \ldots$. Now recall that the effect of $P_e$ depends only on the original contents of $R_1, \ldots, R_{\rho(P_e)}$, where $\rho$ is the function defined in Chapter 2 § 2; $\rho(P_e)$ is independent of $n$.)
   (b) Show further that there is such a function $s^m$ that is primitive recursive.

3. (a) Show that there is a decidable predicate $Q(x, y, z)$ such that
      i. $y \in E_x$ if and only if $\exists z\ Q(x, y, z)$
      ii. if $y \in E_x$ and $Q(x, y, z)$, then $\phi_x((z)_1) = y$.
   (b) Deduce that there is a computable function $g(x, y)$ such that
      i. $g(x, y)$ is defined if and only if $y \in E_x$.
      ii. if $y \in E_x$, then $g(x, y) \in W_x$ and $\phi_x(g(x, y)) = y$; i.e.$g(x, y) \in \phi^{-1}_x(\{y\})$.
      If $y \in E_x$, then we get from definition that $g(x, y)$ is defined and $g(x, y) = (\mu z\ Q(x, y, z))_1$. Furthermore, from the definition of $Q$, we get $(\mu z\ Q(x, y, z))_1 = g(x, y) \in W_x$ and $\phi_x((\mu z\ Q(x, y, z))_1) = \phi_x(g(x, y)) = y$.
   (c) Deduce that if $f$ is a computable injective function (not necessarily total or surjective) then $f^{-1}$ is computable. (cf. exercise 2-5.4(1)).

4. (cf. example 3-7.1(b)) Suppose that $f$ and $g$ are unary computable functions; assuming that $T_1$ has been formally proved to be decidable, prove formally that the function $h(x)$ defined by
   $h(x) = \begin{cases} 1 & \text{if } x \in Dom(f) \text{ or } x \in Dom(g), \\ \text{undefined} & \text{otherwise,} \end{cases}$ is computable.

5. Prove the equivalent of example 5 in Chapter 5-3.1 for the operations of substitution and minimalisation, namely:
   (a) Fix $m, n \geq 1$; there is a total computable function $s(e, e_1, \ldots, e_m)$ such that (in the notation of theorem 2.2) $\phi^{(n)}_{s(e,e_1,\ldots,e_m)} = Sub(\phi^{(m)}_e; \phi^{(n)}_{e_1}, \phi^{(n)}_{e_2}, \ldots, \phi^{(n)}_{e_m})$.
   (b) Fix $n \geq 1$; there is a total computable function $k(e)$ such that for all $e$, $\phi^{(n)}_{k(e)}(\mathbf{x}) \simeq \mu y(\phi^{(n+1)}_e(\mathbf{x}, y) = 0)$. (We could extend the notation of theorem 2.2 in the obvious way and write $\phi^{(n)}_{k(e)} = Min(\phi^{(n+1)}_e)$.)

6. Show that the following problems are undecidable.
   (a) '$x \in E_x$' (*Hint.* Either use a direct diagonal construction, or reduce '$x \in W_x$' to this problem using the s-m-n theorem).

(b) '$W_x = W_y$' (*Hint.* Reduce '$\phi_x$ is total' to this problem).

(c) '$\phi_x(y) = 0$'.

(d) '$E_x$ is infinite'.

(e) '$\phi_x = g$', where $g$ is any fixed computable function.

7. Show that there is no total computable function $f(x, y)$ with the following property: if $P_x(y)$ stops, then it does so in $f(x, y)$ or fewer steps. (*Hint.* Show that if such a function exists, then the Halting problem is decidable.)

8. Show that the following predicates are partially decidable:

   (a) '$E_x^{(n)} \neq \varnothing$' ($n$ fixed).

   (b) '$n$ is a Fermat number'. (We say that $n$ is a *Fermat number* if there is a number $x$, $y$, $z > 0$ such that $x^n + y^n = z^n$.)

   (c) '$M(\mathbf{x})$ and $N(\mathbf{x})$', if $M(\mathbf{x})$ and $N(\mathbf{x})$ are partially decidable.

9. This exercise shows how the technique of reducibility (Chapter 6 §1) may be used to show that a predicate is not partially decidable.

   (a) Suppose that $M(x)$ is a predicate and $k$ a total computable function such that $x \in W_x$ iff $M(k(x))$ does not hold. Prove that $M(x)$ is not partially decidable.

   (b) Prove that '$\phi_x$ is not total' is not partially decidable.
   (Hint. Consider the function $k$ in the proof of theorem 1.6.)

   (c) By considering the function
   $$f(x, y) = \begin{cases} 1 & \text{if } P_x(x) \text{ does not converge in } y \text{ or fewer steps,} \\ \text{undefined} & \text{otherwise.} \end{cases}$$
   Show that '$\phi_x$ is total' is not partially decidable. (*Hint.* Use the s-m-n theorem and (a).)

# Lab4-Various Sets

Name: Yang Fei December 23, 2014

1. Let $A, B$ be subsets of $\mathbb{N}$. Define sets $A \oplus B$ and $A \otimes B$ by

$$A \oplus B = \{2x : x \in A\} \cup \{2x+1 : x \in B\},$$
$$A \otimes B = \{\pi(x,y) : x \in A \text{ and } y \in B\},$$

where $\pi$ is the pairing function $\pi(x,y) = 2^x(2y+1) - 1$ of Theorem 4-1.2. Prove that

   (a) $A \oplus B$ is recursive iff $A$ and $B$ are both recursive.

   (b) If $A, B \neq \varnothing$, then $A \otimes B$ is recursive iff $A$ and $B$ are both recursive.

   (c) Suppose $B$ is r.e. If $A$ is creative, then so are $A \oplus B$ and $A \otimes B$ (provided $B \neq \varnothing$).

   (d) If $A$ is simple, then $A \otimes \mathbb{N}$ is r.e., but neither recursive, creative nor simple.

   (e) If $A$, $B$ are simple sets, then $A \oplus B$ is simple, $A \otimes B$ is not simple but $\overline{\overline{A} \otimes \overline{B}}$ is simple.

2. (a) Let $B \subseteq \mathbb{N}$ and $n > 1$; prove if $B$ is recursive (or r.e.) then the predicate $M(x_1, \ldots, x_n)$ given by "$M(x_1, \ldots, x_n) \equiv 2^{x_1}3^{x_2} \ldots p_n^{x_n} \in B$" is decidable (or partially decidable).

   (b) Prove that $A \subseteq \mathbb{N}^n$ is recursive (or r.e.) iff $\{2^{x_1}3^{x_2} \ldots p_n^{x_n} : (x_1, \ldots, x_n) \in A\}$ is recursive (or r.e., respectively).

   (c) Prove that $A \subseteq \mathbb{N}^n$ is r.e. iff $A = \varnothing$ or there is a total computable function $\boldsymbol{f} : \mathbb{N} \to \mathbb{N}^n$ such that $A = Ran(\boldsymbol{f})$. (A *computable function* $\boldsymbol{f}$ from $\mathbb{N}$ to $\mathbb{N}^n$ is an $n$-tuple $\boldsymbol{f} = (f_1, \ldots, f_n)$ where each $f_i$ is a unary computable function and $\boldsymbol{f}(x) = (f_1(x), \ldots, f_n(x))$.)

3. Which of the following sets are recursive? Which are r.e.? Which are productive? Which are creative? Prove your judgements.

   (a) $\{x : x \in E_x\}$,

   (b) $\{x : x$ is a perfect square$\}$,

   (c) $\{x : \phi_x$ is not injective$\}$,

   (d) $\{x : \phi_x$ is not surjective$\}$,

   (e) $\{x : \phi_x(x) = f(x)\}$, where $f$ is any total computable function.

4. Suppose $A$ is an r.e. set. Prove the following statements.

   (a) Show that the sets $\bigcup_{x \in A} W_x$ and $\bigcup_{x \in A} E_x$ are both r.e.

   (b) Show that $\bigcap_{x \in A} W_x$ is not necessarily r.e. (*Hint*: $\forall t \in \mathbb{N}$ let $K_t = \{x : P_x(x) \downarrow$ in t steps$\}$. Show that for any $t$, $K_t$ is recursive; moreover $K = \bigcup_{t \in \mathbb{N}} K_t$ and $\overline{K} = \bigcap_{t \in \mathbb{N}} \overline{K}_t$.)

5. Suppose that $f$ is a total computable function, $A$ is a recursive set and $B$ is an r.e.set. Show that $f^{-1}(A)$ is recursive and that $f(A)$, $f(B)$ and $f^{-1}(B)$ are r,e, but not necessarily recursive. What extra information about these sets can be obtained if $f$ is a bijection?

6. Prove Rice's theorem (Theorem 6-1.7) from Rice-Shapiro theorem (Theorem 7-2.16). (*Hint.* Suppose that '$\phi_x \in \mathscr{B}$' is decidable; then both $\mathscr{B}$ and $\mathscr{C}_1 \backslash \mathscr{B}$ satisfy the conditions of Rice-Shapiro: consider the cases $f_\varnothing \in \mathscr{B}$ and $f_\varnothing \notin \mathscr{B}$.)

7. Let $\mathscr{B}$ be a set of unary computable functions, and suppose that $g \in \mathscr{B}$ is such that for all finite $\theta \subseteq g$, $\theta \notin \mathscr{B}$. Prove that the set $\{x : \phi_x \in \mathscr{B}\}$ is productive. (*Hint.* Follow the first part of the proof of the Rice-Shapiro theorem.)

8. Prove the following statements.

   (a) If $B$ is r.e. and $A \cap B$ is productive, then $A$ is productive.

   (b) If $C$ is creative and $A$ is an r.e. set such that $A \cap C = \varnothing$, then $C \cup A$ is creative.

   (c) Every productive set contains an infinite recursive subset.

9. (a) (Cf. Theorem 7-2.14) Let $A$ be an infinite r.e. set. Show that $A$ can be enumerated without repetitions by a total computable function.

   (b) Suppose $f$ is a total injective computable function such that $Ran(f)$ is not recursive ((a) showed that such functions abound). Show that $A = \{x : \exists y(y > x \wedge f(y) < f(x))\}$ is simple. (*Hint.* To see that $\overline{A}$ is infinite, assume the contrary and show that there would then be a sequence of numbers $y_0 < y_1 < y_2 < \ldots$ such that $f(y_0) > f(y_1) > f(y_2) > \ldots$ To see that $\overline{A}$ does not contain an infinite r.e. set $B$, suppose to the contrary that $B \subseteq \overline{A}$. Then show that the problem $z \in Ran(f)$ is decidable as follows. Given $z$, find $n \in B$ such that $f(n) > z$; now use the fact that $n \notin A$ to devise a finite procedure for testing whether $z \in Ran(f)$.)

10. Recursively Inseparable and Effectively Recursively Inseparable

    Disjoint sets $A$, $B$ are said to be *recursively inseparable* if there is no recursive set $C$ such that $A \subseteq C$ and $B \subseteq \overline{C}$. Furthermore, $A$ and $B$ are said to be *effectively recursively inseparable* if there is a total computable function $f$ such that whenever $A \subseteq W_a$, $B \subseteq W_b$ and $W_a \cap W_b = \varnothing$ then $f(a,b) \notin W_a \cup W_b$ (see the right figure). Note: Recursive inseparability for a pair of disjoint sets corresponds to non-recursiveness for a single set; pair of recursively inseparable sets that are also r.e. correspond to r.e. sets that are not recursive.

    (a) Show that two disjoint sets $A$, $B$ are recursively inseparable iff whenever $A \subseteq W_a$, $B \subseteq W_b$ and $W_a \cap W_b = \varnothing$, there is a number $x \notin W_a \cup W_b$.

    (b) Suppose $A$, $B$ are effectively recursively inseparable. Prove that if $A$, $B$ are both r.e. then they are both creative. (*Note.* Extending the idea of effectiveness to a pair of recursively inseparable sets in this way parallels the step from a nonrecursive set to a set having productive complement; the counterpart to a single creative set is then a pair of effectively recursively separable sets that are both r.e.)

    (c) Let $K_0 = \{x : \phi_x(x) = 0\}$ and $K_1 = \{x : \phi_x(x) = 1\}$. Show that $K_0$ and $K_1$ are r.e. (in particular neither $K_0$ nor $K_1$ is recursive), and that they are both recursively inseparable and effectively recursively inseparable. (*Hint.* For recursively inseparable, suppose that there is such a set $C$ and let $m$ be an index for its characteristic function; consider whether or not $m \in C$. For effectively recursively inseparable, find a total computable function $f$ such that if $W_a \cap W_b = \varnothing$, then $\phi_{f(a,b)}(x) = \begin{cases} 1 & \text{if } x \in W_a, \\ 2 & \text{if } x \in W_b, \\ \text{undefined} & \text{otherwise.} \end{cases}$)

# Answer for Lab1-Computable Functions

Name: Yang Fei
March 5, 2012

1. Show that the following functions are computable by devising programs that computes them.

   (a) $f(x) = 5$;

   **Solution.** The URM program for this sub-question is shown as follows.

   $I_1$ Z(1)
   $I_2$ S(1)
   $I_3$ S(1)
   $I_4$ S(1)
   $I_5$ S(1)
   $I_6$ S(1)

   (b) $f(x,y) = \begin{cases} 0 & \text{if } x = y, \\ 1 & \text{if } x \neq y; \end{cases}$

   **Solution.** The URM program for this sub-question is shown as follows.

   $I_1$ Z(3)
   $I_2$ J(1,2,4)
   $I_3$ S(3)

   (c) $f(x,y) = \begin{cases} 0 & \text{if } x \leq y, \\ 1 & \text{if } x > y; \end{cases}$ If $f(x,y)$ is the $c_M$ of predicate $M$, what is $M$?

   **Solution.** The URM program for this sub-question is shown as follows.

   $I_1$ Z(3)
   $I_2$ Z(4)
   $I_3$ J(3,1,8)
   $I_4$ J(3,2,7)
   $I_5$ S(3)
   $I_6$ J(1,1,3)
   $I_7$ S(4)
   $M$ is $x > y$.

   (d) $f(x) = [3x/4]$ ($[z]$ denotes the greatest integer $\leq z$).

   **Solution.** The URM program for this sub-question is shown as follows.

   $I_1$ Z(2)
   $I_2$ Z(3)
   $I_3$ Z(4)
   $I_4$ J(1,2,10)
   $I_5$ S(2)
   $I_6$ S(3)
   $I_7$ S(3)
   $I_8$ S(3)
   $I_9$ J(1,1,4)
   $I_{10}$ Z(2)
   $I_{11}$ J(2,3,21)
   $I_{12}$ S(2)
   $I_{13}$ J(2,3,21)
   $I_{14}$ S(2)

$I_{15}$ J(2,3,21)
$I_{16}$ S(2)
$I_{17}$ J(2,3,21)
$I_{18}$ S(2)
$I_{19}$ S(4)
$I_{20}$ J(1,1,11)

2. Suppose $P$ is a program without any jump instructions. Show that

   (a) there is a number $m$ such that either $\forall x : f_P^{(1)}(x) = m$, or $\forall x : f_P^{(1)}(x) = x + m$.

   **Solution.** We prove a stronger statement to show this one is true.That is, for every such URM Program with only one input,there is a number m accordingly(not necessarily the same) for each register r such that when the program is finished the content of the corresponding register is either always m or always x+m.
   Let's prove it by induction on the length of the program.

   i. If the program is of length $l = 1$,it's easy to see the statement holds.

   ii. If the program is of length $l + 1$,the last instruction can be T,S or Z.
   For instruction $T(p, q)$:it updates $r_q$ to $r_p$,by induction hypothesis after the finish of the former $l$ instructions,$r_p$ is either always $m_p$ or always $x + m_p$.So after $l + 1$ instructions $r_q$ is either always $m_p$ or always $x + m_p$.Other registers remain the same as after $l$ istructions
   For instruction $Z(p)$.It updates $r_p$ to 0. So $r_p$ is always 0.It holds for $r_p$.Others remain unchanged as after $l$ instructions.
   For instruction $S(p)$,by induction hypothesis,after $l$ instructions $r_p$ is either always $m_p$ or always $m_p+x$. So after $l+1$ instructions $r_p$ is either always $m_p+1$ or $m_p+1+x$, that is $m'_p$ or $m'_p + x$.
   So,it holds for $l + 1$ if it holds for $l$. By induction the statement holds for all $l$. As $f_P^{(1)}(x)$ is only the result of $r_1$,the original statement holds.
   □

   (b) not every computable function is computable in this sense.

   **Solution.** From the previously discuss is easy to see $f(x) = 2x$ which is URM computable is not computable in this sense.□

3. Show that for each transfer instruction $T(m, n)$ there is a program without any transfer instructions that has exactly the same effect as $T(m, n)$ on any configuration of the URM (Thus transfer instructions are really redundant in the formulation of our URM; it is nevertheless natural and convenient to have transfer as a basic facility of the URM).

   **Solution.** Replace each $T(m, n)$ in the original program by the following piece of codes.

   $Z(n)$
   $I_{in} : J(m, n, out)$
   $S(n)$
   $J(1, 1, in)$
   $I_{out} : ...$
   and then after each replacement, add just the J instruction accordingly,because we increased the number of instructions.Make sure each J instruction has the same destination instruction as before.If the former destination instruction is a J instruction, which is replaced, update the destination to the first instruction of the replacement instruction block.This way the new program does the same work as the old one. □

4. Show that the following predicates are decidable.

(a) '$x$ is even' on $\mathbb{Z}$ (Firstly design an URM to '$x$ is even' on $\mathbb{N}$, then give coding and $f^*$).

**Solution.**

i. URM to '$x$ is even' on $\mathbb{N}$
$I_1$ Z(2)
$I_2$ Z(3)
$I_3$ Z(4)
$I_4$ S(3)
$I_5$ J(1,2,12)
$I_6$ J(1,3,13)
$I_7$ S(2)
$I_8$ S(2)
$I_9$ S(3)
$I_{10}$ S(3)
$I_{11}$ J(1,1,5)
$I_{12}$ S(4)
$I_{13}$ T(4,1)

ii. coding and $f^*$.
we define $\alpha(n) = \begin{cases} 2n & \text{if } n \geq 0, \\ -2n-1 & \text{if } n < 0. \end{cases}$
Then $\alpha^{-1}$ is given by
$\alpha^{-1}(m) = \begin{cases} \frac{1}{2}m & \text{if m is even,} \\ -\frac{1}{2}(m+1) & \text{if m is odd.} \end{cases}$
Consider now the function
$f(x) = \begin{cases} 1 & \text{if m is even,} \\ 0 & \text{if m is odd.} \end{cases}$ on $\mathbb{Z} \to \mathbb{Z}$ map to $f^*$ on $\mathbb{N} \to \mathbb{N}$ is given by
$f^* = \alpha \circ f \circ \alpha^{-1}$ and $f^*$ is computable is obviously. □

(b) $x$ is a power of a prime number,

**Solution.** *Let $P_z(x) = \mu z < x$ ($div(z,x) = 1$ and $P_r(z) = 1$), just the minimum prime factor of $x$. By textbook page 40 to page 41, $div(z,x)$, $equal(x,y)$ and $P_r(z)$ are computable, so $P_z(x)$ is computable. Define the $C_M$ to be:*
$C_M(x) = \sum_{i<x}(equal((P_z(x))^i, x)$
*So $f(x)$ is computable, it will be 1 when $x$ is power of prime number and be 0 otherwise. So the predicate is decidable.* □

(c) $x$ is a perfect cube.

**Solution.** *We define $c_M$ of this predicate as*
$c_M = \bar{sg}(\sum_{m \leq x} equal(m^3, x))$
*It is obviously that predicate M is decidable so M is decidable.* □

(d) $M(x,y) \equiv$ '$g(x) = y$', if $g(x)$ is a total computable function.

**Solution.** *The $c_M$ of the predicate is:*
$c_M(x,y) = \bar{sg}(|g(x) - y|)$
*Since $g(x)$ is totally computable, then $c_M(x,y)$ is totally computable too. So predicate $M(x,y)$ is decidable.*□

5. Without writing any programs, show that for all $m \in \mathbb{N}$ the following functions are computable:

   (a) $\mathbf{m}$ (recall that $\mathbf{m}(x) = m$, for all $x$).

   **Solution.** *When $m = 0$, $\mathbf{0}(\mathbf{x})$ is computable.*
   *Suppose $\mathbf{m}$ is computable when $m = k$, when $m = k + 1$:*
   *$k + 1 = successor(k)$ is computable according to the Lemma of basic functions.*
   *Therefore, $\mathbf{m}$ is computable.*□

   (b) $mx$.

   **Solution.** *Since $\mathbf{m}$ and $f(x, y) = x \cdot y$ are both computable, $mx$ is computable based on the substitution rule.*□

   (c) $f(x, m)$, if $f(x, y)$ is computable.

   **Solution.** *Since $m$ is computable, $f(x, m)$ is computable based on the substitution rule.*□

6. Show that the following functions are computable:

   (a) Any polynomial function $a_0 + a_1x + \ldots + a_nx^n$, where $a_0, a_1, \ldots, a_n \in \mathbb{N}$,

   **Solution.** *$xy$ and $x^y$ is computable. So for any $k \in \mathbb{N}$, $a_kx^k$ is computable. Define $f_n(x) = a_0 + a_1x + \ldots + a_nx^n$. By mathematical induction:*

   *i. When $n = 0$, $f_0(x) = a_0$ is computable.*

   *ii. When $n = k + 1$, assume $f_k(x)$ is computable. $f_{k+1}(x) = f_k(x) + a_{k+1}x^{k+1}$, since $a_{k+1}x^{k+1}$ is computable and $add(x, y) = x + y$ is computable, $f_{k+1}(x)$ is also computable.*

   *So for any $n$, $f_n(x) = a_0 + a_1x + \ldots + a_nx^n$ is computable.*□

   (b) $LCM(x, y) =$ the least common multiple of $x$ and $y$,

   **Solution.** *Define $LCM(x, y) = \mu z < xy + 1(div(x, z)div(y, z) = 1)$. Since $div(x, y)$ is computable, $LCM(x, y)$ is computable.*□

   (c) $HCF(x, y) =$ the highest common factor of $x$ and $y$,

   **Solution.** *$HCF(x, y) = \frac{xy}{LCM(x,y)}$. Because $qt(x, y)$ and $LCM(x, y)$ is computable, and $LCM(x, y)$ must divide $xy$, $HCF(x, y)$ is computable.* □

   (d) $f(x) =$ the number of prime divisors of $x$,

   **Solution.** *By the definition we can give that $f(x) = \sum_{m \le x} div(m, x) * Pr(m)$.*
   *And since the div function and Pr function is computable, so $f(x)$ is computable.*□

   (e) $\phi(x) =$ the number of positive integers less than $x$ which are relatively prime to $x$. (*Euler's function*)(We say that $x, y$ are *relatively prime* if $HCF(x, y) = 1$.)

   **Solution.** *By the definition we can give that $\phi(x) = \sum_{m \le x} \bar{sg}(HCF(m, x) - 1)$*
   *And since the $HCF$ and the sg functions are computable, so function $\phi$ is computable.*□

7. Let $\pi(x, y) = 2^x(2y + 1) - 1$. Show that $\pi$ is a computable bijection from $\mathbb{N}^2$ to $\mathbb{N}$, and that the functions $\pi_1$, $\pi_2$ such that $\pi(\pi_1(z), \pi_2(z)) = z$ for all $z$ are computable.

   **Solution.** *Divide the question to subproblems.*

(a) *Prove $\pi(x, y)$ is computable*
*Because the power function and the multiplication function are computable, so we can easily conclude that $\pi$ is computable.*

(b) *Prove $\pi(x, y)$ is a bijection from $\mathbb{N}^2$ to $\mathbb{N}$.*
*First we prove that for every (x,y) in $\mathbb{N}^2$ the function can only have a unique result.*
*Suppose $\pi(x_1, y_1) = \pi(x_2, y_2)$ and $(x_1, y_1) \neq (x_2, y_2)$.*
*If $x1 = x2$ then $(\pi(x_1, y_1) + 1)/2^{x_1} = (\pi(x_2, y_2) + 1)/2^{x_2}$, so $2y_1 + 1 = 2y_2 + 1$, so $y_1 = y_2$ it conflict with the consumption.*
*If $x1 \neq x2$, then suppose that $x_1 < x_2$.*
*A nd because $(\pi(x_1, y_1) + 1)/2^{x_1} = (\pi(x_2, y_2) + 1)/2^{x_1}$.*
*So $(2y_1 + 1) = 2^{x_2 - x_1}(2y_2 + 1)$. Because the left is odd, and the right is even, so conflict. So for every $(x, y)$, it only maps to unique result in $\mathbb{N}$. Second we prove that every $z$ in $\mathbb{N}$, it can be calculated by a pair of $(x, y)$.*
*If $z$ is an even number, then the value of $x$ must be 0. And then we got $z = \pi(x, y) = 2^x(2y + 1) - 1 = 2y + 1$, so their must be a value of $y$ which fits. If $z$ is an odd number, then the $z + 1$ is an even number, it can always be written as $2^x(2y + 1)$. Join these two directions of the proof, we can conclude that $\pi(x, y)$ is a bijection from $\mathbb{N}^2$ to $\mathbb{N}$*
*$\pi_1(z)$ can be defined like this:*

$$\pi_1(z) = \begin{cases} 0 & \text{if } z \text{ is even,} \\ (z+1)_1 & \text{otherwise.} \end{cases}$$

*$(z)_i$ *is computable so $\pi_1(z)$ is computable.*
*$\pi_2(z)$ can be defined like this:*

$$\pi_2(z) = \begin{cases} \frac{z}{2} & \text{if } z \text{ is even,} \\ \frac{\frac{z+1}{2^{\pi_1(z)}} - 1}{2} & \text{otherwise.} \end{cases}$$

*Since $\pi_1(z)$ is computable, $\pi_2(z)$ is computable.*□

8. Suppose $f(x)$ is defined by $\begin{cases} f(0) = 1, \\ f(1) = 1, \\ f(x+2) = f(x) + f(x+1). \end{cases}$ ($f(x)$ is the *Fibonacci* sequence)
Show that $f$ is computable (*Hint*: first show that the function $g(x) = 2^{f(x)}3^{f(x+1)}$ is computable, using recursion).

**Solution.** *We will use **prime power encoding** to solve this problem. Coding can be simply done by $z = \prod_{i<k} p_i^{a_i}$. And decoding can be simply done by $g(z, i) = (\mu x < z)(rm(z, p_i^{x+1}))$. These functions are all computable according to the theorems and corollaries in the textbook.*
*Let $(z)_i$ = the exponent of $p_i$ in the prime power decomposition of $z$. Then we use **prime power encoding** to rewrite the function $g(x)$ into*

$$G_0 = 6$$
$$G_{n+1} = 2^{(G_n)_2}3^{(G_n)_1 + (G_n)_2}$$

*From above equations which satisfy the recursion equation, we can conclude that $G_n$ is a computable function as a function of $n$. It follows that $f_n = (G_n)_1$ is computable as a function of $n$.*□

*$(z)_i$ is the exponent of $p_i$ in the prime power decomposition of $z$

9. Any number $x \in \mathbb{N}$ has a unique expression as

   (a) $x = \sum_{i=0}^{\infty} \alpha_i 2^i$, with $\alpha_i = 0$ or $1$, all $i$. Hence, if $x > 0$, there are unique expressions for $x$ in the forms;

   (b) $x = 2^{b_1} + 2^{b_2} + \ldots + 2^{b_1}$, with $0 \leq b_1 < b_2 < ... < b_l$ and $l \geq 1$. And

   (c) $x = 2^{a_1} + 2^{a_1+a_2+1} + \ldots + 2^{a_1+a_2+\ldots+a_k+k-1}$.

   Putting

   $\alpha(i, x) = \alpha_i$ as in the expression (a);

   $$l(x) = \begin{cases} l \text{ as in (b)}, & \text{if } x > 0, \\ 0 & \text{otherwise;} \end{cases}$$

   $$b(i, x) = \begin{cases} b_i \text{ as in (b)}, & \text{if } x > 0 \text{ and } 1 \leq i \leq l, \\ 0 & \text{otherwise;} \end{cases}$$

   $$a(i, x) = \begin{cases} a_i \text{ as in (c)}, & \text{if } x > 0 \text{ and } 1 \leq i \leq l, \\ 0 & \text{otherwise;} \end{cases}$$

   Show that each of the functions $\alpha$, $l$, $b$, $a$ is computable (The expression (c) is a way of regarding $x$ as coding the sequence $(a_1, a_2, \ldots, a_l)$ of numbers).

   **Solution.** *We can define $\alpha(i, x)$ in this way:*

   $$\alpha(i, x) = \begin{cases} 1 \textit{ as in (a)}, & \textit{if } rm(2, qt(x, 2^i)) = 1, \\ 0 & \textit{otherwise;} \end{cases}$$

   *Since $qt(x, y)$ and $rm(x, y)$ are computable functions, $\alpha(i, x) = \alpha_i$ as in expression(a) is computable.*

   *We can define $l(x)$ in this way:*

   $l(x) = \sum_{i<\mu y(2^y>x)} \alpha(i, x)$.

   *Since $\alpha(i, x)$ is computable and a bounded sum is computable, we have $l(x)$ is computable.*

   *We can define $b(x)$ as:*

   $b(i, x) = \mu y(\sum_{k=0}^{y} \alpha(k, x) \geq i)$

   *According to Corollary 5.3 and since $\alpha(i, x)$ is computable, $b(i, x)$ is computable.*

   *Let $a(0, x) = 0$, then $a(i, x)$ can be defined in this way:*

   $a(i, x) = b(i, x) - b(i - 1, x) - 1$ *for* $i \in \mathbb{N}$.

   *$b(i, x)$ is computable so $a(i, x)$ is computable.*$\square$

10. Gadgets

In order to construct URM to perform complex operations, it is useful to build it from smaller components that we'll call *gadgets*, which perform specific operations. A gadget will be defined by a series of instructions and will operate on registers that are specified in the gadget's name. For instance, the gadget "predecessor $r_n$" denoted by $P(n)$ will subtract 1 from the contents of register $R_n$ if it is non-zero. It can be represented by an instruction sequence shown in the right block. For simplicity, when we obey gadget function $P(l)$, we by default obey $P^{-1}[l_1, \cdots, l_n \to l]$, meaning we will use registers $R_{l_1}, \cdots, R_{l_n}$ $(l_i > \rho(P), \forall 1 \leq i \leq n)$ and place the result in $R_l$, without any interference to the next instructions. Now answer the following questions:

Gadget $P(1)$

| | |
|---|---|
| $I_1$ | J(1,4,9) |
| $I_2$ | S(3) |
| $I_3$ | J(1,3,7) |
| $I_4$ | S(2) |
| $I_5$ | S(3) |
| $I_6$ | J(1,1,3) |
| $I_7$ | T(2,1) |

URM $Q$

| | |
|---|---|
| $I'_1$ | J(1,2,6) |
| $I'_2$ | S(2) |
| $I'_3$ | T(2,3) |
| $I'_4$ | M(2,3,4) |
| $I'_5$ | G(1,4,2) |
| $I'_6$ | H(2) |

(a) Define a gadget "greater than $r_m > r_n$" denoted by $G(m, n, q)$, which determines whether the initial value of $R_m$ is greater than that of $R_n$. If yes, jump to the $q$th instruction, otherwise go on to the next instruction.

(b) Define a gadget "halt with $r_n$" denoted by $H(n)$, which leaves $R_n$ with its initial value, and overwrites the initial values of other registers into 0 (write the instruction sequences).

(c) Define a gadget "multiply $r_m$ by $r_n$ to $R_p$" denoted by $M(m, n, p)$, which multiplies $r_m$ by $r_n$ and stores the result in $R_p$.

(d) Describe the function of one argument $f(x)$ computed by the program $Q$. (What is $f_Q^{(1)}$?)

**Solution.** *(a) The URM program for $G(m, n, q)$*

$I'_1$ *J(1,n,5)*
$I'_2$ *J(1,m,q)*
$I'_3$ *S(1)*
$I'_4$ *J(1,1,1)*

*(b) The URM program for $H(n)$*
$I'_1$ *Z(1)*
$I'_2$ *Z(2)*
.....
$I'_{n-1}$ *Z(n-1)*
$I'_n$ *J(1,1,n+1)*
$I'_{n+1}$ *Z(n+1)*
.....
$I'_\rho$ *Z(ρ)*

*(c) The URP program for $M(m, n, p)$*
$I'_1$ *J(1,m,9)*
$I'_2$ *S(1)*
$I'_3$ *Z(2)*
$I'_4$ *J(2,n,8)*
$I'_5$ *S(2)*
$I'_6$ *S(3)*
$I'_7$ *J(1,1,4)*
$I'_8$ *J(1,1,1)*
$I'_9$ *T(3,p)*

*(d)* $f_Q^{(1)}$ *is* $\lceil \sqrt{x} \rceil$

# Lab2-Church's Thesis

Name: Yang Fei March 20, 2012

1. Show the computability of the following functions by minimalisation.

   (a) $f^{-1}(x)$, if $f(x)$ is a total injective computable function.

   **Sol.**

   $$f^{-1}(x) = \mu y(f(y) = x)$$

   Since $f(x)$ is computable, $f^{-1}(x)$ is computable by minimalisation. □

   (b) $f(a) = \begin{cases} \text{least non-negative integral root of } p(x) - a (a \in \mathbb{N}), \\ \text{undefined if there's no such root,} \end{cases}$

   where $p(x)$ is a polynomial with integer coefficients.

   **Sol.**

   $$f(a) = \mu y(p(y) = a)$$

   And $p(x)$ is computable based on the conclusions of Lab1, so $f(a)$ is computable by minimalisation. □

   (c) $f(x, y) = \begin{cases} x/y & \text{if } y \neq 0 \text{ and } y|x, \\ \text{undefined} & \text{otherwise,} \end{cases}$

   **Sol.**

   $$f(x, y) = \mu z(y \times z = x)$$

   Thus $f(x, y)$ is computable by minimalization. □

2. Devise Turing machines that will Turing-compute the functions: (a) $x \dot{-} 1$, and (b) $2x$.

   **Sol.** (a)

   $$q_S \triangleright R q_1$$
   $$q_1 1 0 q_1$$
   $$q_1 0 R q_2$$
   $$q_2 1 0 q_2$$
   $$q_2 0 L q_2$$
   $$q_1 \triangleleft L q_3$$
   $$q_2 \triangleleft L q_3$$
   $$q_3 0 L q_3$$
   $$q_3 1 L q_3$$
   $$q_3 \triangleright L q_H$$

(b)

$$q_S \triangleright B q_1$$
$$q_1 0 R q_2$$
$$q_2 1 R q_2$$
$$q_2 0 R q_3$$
$$q_3 1 R q_3$$
$$q_3 \triangleleft L q_4$$
$$q_3 0 L q_4$$
$$q_4 1 0 q_4$$
$$q_4 0 L q_5$$
$$q_5 1 L q_5$$
$$q_5 0 L q_6$$
$$q_6 1 L q_6$$
$$q_6 0 1 q_7$$
$$q_7 1 L q_7$$
$$q_7 0 1 q_8$$
$$q_8 1 R q_8$$
$$q_8 0 R q_9$$
$$q_9 1 R q_3$$
$$q_9 0 L q_{10}$$
$$q_{10} 0 L q_{11}$$
$$q_{11} 1 L q_{11}$$
$$q_{11} 0 \triangleright q_H$$

□

3. Suggest natural definition of computability on domain (a) $3 \times 3$ matrices; and (b) $\mathbb{Q}$.

**Sol.** (a) Note $\mathbb{M}$ to express the domain of $3 \times 3$. First construct a bijection from $\mathbb{M}$ to $\mathbb{N}$.
For the $3 \times 3$ matrix: $A = \begin{pmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{pmatrix}$
Let $\zeta'(m, n, q) = \pi(\pi(m, n), q)$, no minus one, it is different from $\zeta$.
And $M(A) = \zeta'(\zeta'(a_1, a_2, a_3), \zeta'(a_4, a_5, a_6), \zeta'(a_7, a_8, a_9))$.

i. Then define the basic functions:
   A. $Z(A) = \Lambda = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$
   B. $S(A) = \zeta'^{-1}(\zeta'(A) + 1)$
   C. $U_i^n(A_1, A_2 \ldots A_n) = A_i$

ii. $\mathbb{M}$ is closed under substitution.

iii. $\mathbb{M}$ is closed under primitive recursive definitions of the following form:
$h(A, \Lambda) \simeq f(A)$
$h(A, S(B)) \simeq g(A, B, h(A, B))$
where $A, B \in \mathbb{M}$.

iv. $\mathbb{M}$ is closed under minimalisation:
$h(A) = \mu B(f(B) = \Lambda)$
The order of the matrix $A_i$ corresponding to the value of $M(A_i)$.

(b) i. First define the basic functions:

A. $Z(q) = 0$

B. Note $q = \begin{cases} \frac{m}{n} & \text{if } q > 0 \text{ and } gcd(m, n) = 1 \\ -\frac{m}{n} & \text{if } q < 0 \text{ and } gcd(m, n) = 1 \end{cases}$

Define $d = \mu x(gcd(\pi_1(\pi(m, n) + x), \pi_2(\pi(m, n) + x)) = 1)$.

$S(q) = \begin{cases} -\frac{m}{n} & \text{if } q \geq 0 \\ \frac{\pi_1(\pi(m,n)+d)}{\pi_2(\pi(m,n)+d)} & \text{if } q < 0 \end{cases}$

C. $U_i^n(q_1, q_2 \ldots q_n) = q_i$

ii. $\mathbb{Q}$ is closed under substitution.

iii. $\mathbb{Q}$ is closed under primitive recursive definitions of the following form:
$h(q, 0) \simeq f(q)$
$h(q, S(r)) \simeq g(q, r, h(q, r))$
where $q, r \in \mathbb{Q}$.

iv. $\mathbb{Q}$ is closed under minimalisation:
$h(q) = \mu r(f(r) = 0)$
The order of $q$ is defined by the order of generation every rational number by $S(q)$ from 0.

□

4. Prove that the following functions are URM-computable by Church's Thesis.

(a) $h(x) = \begin{cases} x & \text{if } x \in Dom(f) \cap Dom(g), \\ \text{undefined} & \text{otherwise.} \end{cases}$

where $f(x)$ and $g(x)$ are effectively computable functions.

**Sol.** We start the algorithm by doing the calculation of $f(x)$ and $g(x)$ simultaneously. If they both terminate in finite time, then set $h(x) = x$. Otherwise, one of the function is undefined. In such case the simulation will never end, thus $h(x) = undefined$.
Therefore, $h(x)$ is intuitively computable and thus URM-computable by Church's Thesis.
□

(b) $h(x) = \begin{cases} 1 & \text{if } x \in Ran(f), \\ \text{undefined} & \text{otherwise.} \end{cases}$ if $f$ is a total unary computable function.

**Sol.** Since $f(x)$ is a total function, we could calculate $\mu y(f(y) = x)$, if it converges in finite time, $h(x) = 1$, otherwise $h(x) = undefined$.
Therefore, $h(x)$ is intuitively computable and thus URM-computable by Church's Thesis.
□

(c) Ackermann function $\psi(x, y)$.

**Sol.**

Using mathematic induction to prove for any $x$ and $y$, $\psi(x, y)$ is computable.

i. When $x = 0$, for any $\psi(0, y) = y + 1$ is computable.

ii. Assume for any $y$, $\psi(x, y)$ is computable. Then prove $\psi(x + 1, y)$ is computable. Firstly $\psi(x+1, 0) = \psi(x, 1)$. By assumption, $\psi(x, 1)$ is computable, thus $\psi(x+1, 0)$ is computable. When $y > 0$, $\psi(x + 1, y + 1) = \psi(x, \psi(x + 1, y))$. Using mathematic induction again to prove for the fixed $x$ and any $y$, $\psi(x + 1, y)$ is computable:

A. $\psi(x + 1, 0)$ is computable have been got.

B. Assume $\psi(x + 1, y)$ is computable. $\psi(x + 1, y + 1) = \psi(x, \psi(x + 1, y))$, since for any $\psi(x, m)$ is computable and $\psi(x + 1, y)$ is computable. Using substitution, $\psi(x + 1, y + 1)$ is computable.

Then for any $y$, $\psi(x + 1, y + 1)$ is computable.

Therefore, $\psi(x, y)$ is URM-computable by Church's Thesis. □

(d) $g(n) = n^{th}$ digit in the decimal expansion of $e$ ($e$ is the basis for natural logarithms).

**Sol.** Consider Taylor's series for $e$

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + ... = \sum_{n=0}^{\infty} \frac{1}{n!}$$

Let $c_k = 10^k$, $s_k = \sum_{n=0}^{c_k} h_n$. Hence $\forall k$, we have $s_k < e$. Also, $\forall k > 2$, we have

$$\begin{aligned}
e - s_k &= \sum_{n=c_k+1}^{\infty} \frac{1}{n!} \\
&= \frac{1}{(c_k+1)!}(1 + \frac{1}{c_k+2} + \frac{1}{(c_k+2)\times(c_k+3)} + \frac{1}{(c_k+2)\times(c_k+3)\times(c_k+4)} + ...) \\
&< \frac{1}{c_k!}(1 + \frac{1}{2} + \frac{1}{2\times 3} + \frac{1}{2\times 3\times 4} + ...) \\
&< \frac{e}{(10^k)!} \\
&< \frac{3}{(10^k)!} \\
&< \frac{1}{10^k} \ (k > 2)
\end{aligned}$$

Hence, $\forall k > 2$, $s_k < e < s_k + \frac{1}{10^k}$. Recall that $a_k$ is computable, $s_k$ is rational, so using long division we can effectively calculate the decimal expansion of $s_k$ to any disired digit. Following is an effetive method for $g(n)$:

'For $n = 0$, set $g(n) = 2$ and terminate. For $n = 1$, set $g(n) = 7$ and stop. For $n = 2$, set $g(n) = 1$ and halt. For $n > 2$, find the first $N \geq n + 1$ such that the decimal expansion

$$s_N = a_0.a_1a_2...a_na_{n+1}...a_N...$$

does not have all of $a_{n+1}, ..., a_N$ equal to 9. Then put $g(n) = a_n$.'

Similarly as the proof in the textbook, we have the $n$th decimal place of $e$ is indeed $a_n$. Therefore by Church's thesis, $g$ is computable. □

5. Find the Gödel encoding (a) $\beta(J(3, 4, 2))$, (b) $\beta^{-1}(503)$, (c) $P_{100}$, and (d) The code number of program $P = \{T(3, 4), S(3), Z(1)\}$.

**Sol.** (a) $\beta(J(3,4,2)) = 4\zeta(3,4,2) + 3 = 4\pi(\pi(2,3),1) + 3$
$= 4\pi(2^2(2\times 3+1)-1,1)+3 = 4\pi(27,1)+3$
$= 4\times(2^{27}(2\times 1+1)-1)+3$
$= 3\cdot 2^{29}-1$

(b) $\beta^{-1}(503) = \beta^{-1}(500+3) = \beta^{-1}(4\times 125+3)$
Hence, $\zeta(m,n,q) = 125$
And therefore,
$m = \pi_1(\pi_1(125)+1) = \pi_1(1)+1 = 2$
$n = \pi_2(\pi_1(125))+1 = \pi_2(1)+1 = 1$
$q = \pi_2(125)+1 = 32$
Thus,
$\beta^{-1}(503) = J(2,1,32)$.

(c) Since $100 = 2^0+2^2+2^5+2^6-1$,
hence,
$\beta(I_1) = 4\times 0$
$\beta(I_2) = 4\times 0+1$
$\beta(I_3) = 4\pi(m-1,n-1)+2$
$\beta(I_4) = 4\times 0$
Thus,
$P_{100} = Z(1); S(1); T(1,1); Z(1);$

(d) $\beta(T(3,4)) = 4\pi(2,3)+2 = 4(2^2(2\times 3+1)-1)+2 = 110$
$\beta(S(3)) = 4\times 2+1 = 9$
$\beta(Z(1)) = 0$
Thus,
$\gamma(P) = 2^{110}+2^{120}+2^{121}-1$

□

6. Let $f(x,y)$ be a total computable function. $\forall m$, let $g_m$ be the computable function given by $g_m(y) = f(m,y)$. Construct a total computable function $h$ such that for each $m$, $h \neq g_m$.

**Sol.** Let $h(x) = f(x,x)+1$, which is computable.

Then for each $m$, $h$ differs $g_m$ at the value of $m$, since $h(m) = f(m,m)+1 = g_m(m)+1 \neq g_m(m)$.

Therefore, $h \neq g_m$ for any $m$. □

7. Let $f_0, f_1, \ldots$ be an enumeration of partial functions from $\mathbb{N}$ to $\mathbb{N}$. Construct a function $g$ from $\mathbb{N}$ to $\mathbb{N}$ such that $Dom(g) \neq Dom(f_i)$ for each $i$.

**Sol.** For each $x$,

$$g(x) = \begin{cases} 1 & \text{if } x \notin Dom(f_x), \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Thus $Dom(g) \neq Dom(f_i)$ for each $i$, since $i$ either belongs to $Dom(g)$ or belongs to $Dom(f_i)$.
□

8. Let $f$ be a partial function from $\mathbb{N}$ to $\mathbb{N}$, and let $m \in \mathbb{N}$. Construct a non-computable function $g$ such that $g(x) \simeq f(x)$ for $x \leq m$.

**Sol.** Define $g$ as

$$g(x) = \begin{cases} f(x) & x \leq m; \\ \phi_{x-m-1}(x-m-1)+1 & x > m \text{ and } \phi_{x-m-1}(x-m-1) \text{ is defined;} \\ 0 & x > m \text{ and } \phi_{x-m-1}(x-m-1) \text{ is undefined.} \end{cases}$$

where $\phi_0, \phi_1, \phi_2, ...$ is the enumeration of $\mathscr{C}_1$.

First, for $x \leq m$, we have $g(x) \simeq f(x)$.

Second, for $x > m$, $g(x)$ differs from $\phi_{x-m-1}$ at $x-m-1$. Since $g$ differs from every unary computable function $\phi_n$ ($n \in \mathbb{N}$), $g$ does not appear in the enumeration of $\mathscr{C}_1$ and is thus not itself computable.

Therefore, we have constructed a non-computable function $g$ such that $g(x) \simeq f(x)$ for $x \leq m$. □

9. (a) (Cantor) Show that the set of all functions from $\mathbb{N}$ to $\mathbb{N}$ is not denumerable.

**Sol.**

Let $S$ be the set of all functions from $\mathbb{N}$ to $\mathbb{N}$. If it is denumerable, it can be enumerated as $S = \{f_1, f_2 \dots\}$.

Define $g(n) = \begin{cases} 1 & \text{if } f_n(n) \text{ is undefined} \\ f_n(n)+1 & \text{if } f_n(n) \text{ is defined} \end{cases}$

It is the function from $\mathbb{N}$ to $\mathbb{N}$. But for any $f_n$ has $g \neq f_n$, thus $g \notin S$, which leads to contradiction. Thus $S$ is not denumerable. □

(b) Show that the set of all non-computable total functions from $\mathbb{N}$ to $\mathbb{N}$ is not denumerable.

**Sol.** In previous part, I prove $S$, the set of all functions from $\mathbb{N}$ to $\mathbb{N}$, is not denumerable. Let $S_1$ be the set of all non-computable total functions from $\mathbb{N}$ to $\mathbb{N}$ and $S_2$ be the set of all computable total functions from $\mathbb{N}$ to $\mathbb{N}$. As we known, $\mathscr{C}$ is denumerable, thus $S_2$ is denumerable. If $S_1$ is denumerable, $S = S_1 \cup S_2$ must be denumerable, which leads to contradiction. Thus $S_1$ is not denumerable. □

10. Palindrome Recognization

Assume $M$ is a $k$-tape Turing Machine, in which the first tape is the read-only input tape, the other $k-1$ tapes are read/write work tapes, and the $k^{th}$ tape is also used as the output tape. Every tape comes with a read/write head. We define $L$, $R$, and $S$ as the action 'move left', 'move right', and 'stay' for a head. $M$ works on $k$ tapes simultaneously, and thus one instruction should be written as $\langle q_i, s_1, \cdots, s_k \rangle \to \langle q_l, s'_2, \cdots, s'_k, A_1, \cdots, A_k \rangle$, where each $s_i$ is the scanned symbol on the $i^{th}$ tape, $s'_i$ is the replaced symbol, and $A_i$ is the corresponding action. E.g., $\langle q_2, a, a \rangle \to \langle q_3, b, R, S \rangle$ works for $k = 2$. Give the specification of $M$ with $k = 3$ to recognize palindromes on symbol set $\{0, 1, \triangleright, \triangleleft, \square\}$ . (A palindrome is a word that reads the same both forwards and backwards. Example: anna, madam, nitalarbralatin. To recognize palindrome we need to check the input string, output 1 if the string is a palindrome, and 0 otherwise. Initially the head on each tape reads $\triangleright$ as the starting point, with state $q_S$).

**Sol.**

$\langle q_s, \triangleright, \triangleright, \triangleright\rangle \to \langle q_1, \square, \square, R, S, S\rangle$
$\langle q_1, 1, \square, \square\rangle \to \langle q_1, \square, \square, R, S, S\rangle$
$\langle q_1, 0, \square, \square\rangle \to \langle q_1, \square, \square, R, S, S\rangle$
$\langle q_2, 1, \square, \square\rangle \to \langle q_2, 1, \square, L, R, S\rangle$
$\langle q_2, 0, \square, \square\rangle \to \langle q_2, 0, \square, L, R, S\rangle$
$\langle q_2, \triangleright, \square, \square\rangle \to \langle q_3, \triangleleft, \square, R, L, S\rangle$
$\langle q_3, 1, 1, \square\rangle \to \langle q_3, 1, \square, R, L, S\rangle$
$\langle q_3, 0, 0, \square\rangle \to \langle q_3, 0, \square, R, L, S\rangle$
$\langle q_3, \triangleleft, \triangleright, \square\rangle \to \langle q_4, \triangleright, \triangleright, S, S, R\rangle$
$\langle q_4, \triangleleft, \triangleright, \square\rangle \to \langle q_5, \triangleright, 1, S, S, R\rangle$
$\langle q_5, \triangleleft, \triangleright, \square\rangle \to \langle q_H, \triangleright, \triangleleft, S, S, L\rangle$
$\langle q_3, 1, 0, \square\rangle \to \langle q_6, \square, \triangleright, S, S, R\rangle$
$\langle q_3, 0, 1, \square\rangle \to \langle q_6, \square, \triangleright, S, S, R\rangle$
$\langle q_6, 0, \square, \square\rangle \to \langle q_7, \square, 0, S, S, R\rangle$
$\langle q_6, 1, \square, \square\rangle \to \langle q_7, \square, 0, S, S, R\rangle$
$\langle q_7, 0, \square, \square\rangle \to \langle q_H, \square, \triangleleft, S, S, L\rangle$
$\langle q_7, 1, \square, \square\rangle \to \langle q_H, \square, \triangleleft, S, S, L\rangle$

*The output tape will be* $...\square \triangleright 1 \triangleleft \square...$ *if the input is a palindrome string, and* $...\square \triangleright 0 \triangleleft \square...$ *if the input is not a palindrome string.* $\square$

# Lab3-Denumerability

Name: Yang Fei December 8, 2014

1. Show that there is a total computable function $k$ such that for each $n$,

   (a) $k(n)$ is an index of the function $[\sqrt[y]{x}]$.

   **Sol.** Let
   $$f(x,y) = \mu z(z^y > x) - 1 = [\sqrt[y]{x}]$$
   $f(x,y)$ is computable, according to the s-m-n theorem, there exists a $k(y)$ for each $y$, which satisfies
   $$\phi_{k(y)}(x) \simeq f(x,y)$$
   □

   (b) $W^{(m)}_{k(n)} = \{(y_1, \ldots, y_m) : y_1 + y_2 + \ldots + y_m = n\}$ $(m \geq 1)$.

   **Sol.** Let
   $$f(x, y_1, y_2, ..., y_m) = \begin{cases} 1 & \text{if } y_1 + y_2 + \ldots + y_m = x \\ \text{undefined} & \text{otherwise.} \end{cases}$$
   $f(x, y_1, y_2, ..., y_m)$ is computable, according to the s-m-n theorem, there exists a $k(x)$ for each $x$, which satisfies
   $$\phi_{k(x)}(y_1, y_2, ..., y_m) \simeq f(x, y_1, y_2, ..., y_m)$$
   According to the definition of $f$, we know that for each $x$, $W^{(m)}_{k(x)} = \{(y_1, \ldots, y_m) : y_1 + y_2 + \ldots + y_m = x\}$ □

   (c) $E_{k(n)} = W_n$.

   **Sol.** Let
   $$f(x,y) = \begin{cases} y & \text{if } \phi_x(y) \text{ is defined} \\ \text{undefined} & \text{otherwise.} \end{cases}$$
   By the Church's Thesis, $f(x,y)$ is computable, then according to the s-m-n theorem, there exists a $k(x)$ for each $x$, which satisfies
   $$\phi_{k(x)}(y) \simeq f(x,y)$$
   And according to the definition of $f$, $E_{k(x)} = W_x$. □

   (d) $W_{k(n)} = f^{-1}(W_n)$, if $f(n)$ is computable.

   **Sol.** Let
   $$g(x,y) = \begin{cases} 1 & \text{if } \phi_x(f(y)) \text{ is defined} \\ \text{undefined} & \text{otherwise.} \end{cases}$$
   According to Church's Thesis, $f$ is computable. Therefore according to the s-m-n theorem, there exists a $k(x)$ for each $x$, which satisfies
   $$\phi_{k(x)}(y) \simeq g(x,y)$$
   According to the definition of $g$, $W_{k(x)} = f^{-1}(W_n)$. □

2. (a) Show that for each $m$ there is a total $(m+1)$-ary computable function $s^m$ such that for all $n$, $\phi_e^{(m+n)}(\mathbf{x},\mathbf{y}) \simeq \phi^{(n)}_{s^m(e,\mathbf{x})}(\mathbf{y})$, where $\mathbf{x},\mathbf{y}$ are $m$- and $n$-tuples respectively.
(*Hint.* Consider the definition of $s^m_n(e,\mathbf{x})$ given in the proof of Chapter 4-theorem 4.3. The only way in which $n$ was used was in determining how many of the $r_1, r_2, \ldots$ to transfer to $R_{m+1}, R_{m+2}, \ldots$. Now recall that the effect of $P_e$ depends only on the original contents of $R_1, \ldots, R_{\rho(P_e)}$, where $\rho$ is the function defined in Chapter 2 § 2; $\rho(P_e)$ is independent of $n$.)

**Sol.** We generalise the proof of theorem 4.1.
For any $i \geq 1$ let $Q(i,x)$ be the subroutine

$Q(i,x)$

| | |
|---|---|
| $I_1$ | $Z(i)$ |
| $I_2$ | $S(i)$ |
| $\vdots$ | |
| $I_{x+1}$ | $S(i)$ |

that replaces the current contents of $R_i$ by x. Then for fixed m, define the $s^m(e,\mathbf{x})$ to be the code number of the following program:

$S^m(e,\mathbf{x})$

| |
|---|
| $T(\rho(P_e), m+\rho(P_e))$ |
| $\vdots$ |
| $T(2,m+2)$ |
| $T(1,m+1)$ |
| $Q(1,x_1)$ |
| $Q(2,x_2)$ |
| $\vdots$ |
| $Q(m,x_m)$ |
| $P_e$ |

According to this explicit definition, and the effectiveness of $\gamma$ and $\gamma^{-1}$, we get that $s^m$ is effectively computable. □

(b) Show further that there is such a function $s^m$ that is primitive recursive.

**Sol.** Since $\pi(m,n)$, $\psi(m,n,q)$, $\beta$ and $\gamma$ is primitive recursive, the function to calculate the coding of the program shown above is primitive recursive. □

3. (a) Show that there is a decidable predicate $Q(x,y,z)$ such that

i. $y \in E_x$ if and only if $\exists z\ Q(x,y,z)$

**Sol.** Let

$$Q(x,y,z) = S_n(x,(z)_1,y,(z)_2)$$

which is apparently decidable according to substitution and the decidability of $S_n$. Then we have $y \in E_x$ if and only if $\exists z\ Q(x,y,z)$. □

ii. if $y \in E_x$ and $Q(x,y,z)$, then $\phi_x((z)_1) = y$.

**Sol.** Let

$$Q(x,y,z) = S_n(x,(z)_1,y,(z)_2)$$

which is decidable.
If $Q(x,y,z)$ and $y \in E_x$, we immediately get $\phi_x((z)_1) = y$ from the definition of $S_n$. □

(b) Deduce that there is a computable function $g(x, y)$ such that

i. $g(x, y)$ is defined if and only if $y \in E_x$.

**Sol.** Let

$$Q(x, y, z) = S_n(x, (z)_1, y, (z)_2)$$

which is decidable.
We define $g(x, y)$ as

$$g(x, y) = \begin{cases} (\mu z\ Q(x, y, z))_1 & \text{if } \exists z\ Q(x, y, z), \\ undefined & \text{otherwise;} \end{cases}$$

From the previous question, we conclude that $g(x, y)$ is defined if and only if $y \in E_x$. And from Church's Thesis, $g(x, y)$ is computable. □

ii. if $y \in E_x$, then $g(x, y) \in W_x$ and $\phi_x(g(x, y)) = y$; i.e.$g(x, y) \in \phi_x^{-1}(\{y\})$.

**Sol.** Let $g(x, y)$ be the same function as in (i), which is

$$g(x, y) = \begin{cases} (\mu z\ Q(x, y, z))_1 & \text{if } \exists z\ Q(x, y, z), \\ undefined & \text{otherwise;} \end{cases}$$

□

If $y \in E_x$, then we get from definition that $g(x, y)$ is defined and $g(x, y) = (\mu z\ Q(x, y, z))_1$. Furthermore, from the definition of $Q$, we get $(\mu z\ Q(x, y, z))_1 = g(x, y) \in W_x$ and $\phi_x((\mu z\ Q(x, y, z))_1) = \phi_x(g(x, y)) = y$.

(c) Deduce that if $f$ is a computable injective function (not necessarily total or surjective) then $f^{-1}$ is computable. (cf. exercise 2-5.4(1)).

**Sol.** Let $g(x, y)$ be the same function as in (i), which is

$$g(x, y) = \begin{cases} (\mu z\ Q(x, y, z))_1 & \text{if } \exists z\ Q(x, y, z), \\ undefined & \text{otherwise;} \end{cases}$$

Let $P_e$ be the program which computes $f$, i.e. $f \simeq \phi_e$.

From s-m-n theorem, there exists a total computable function $k(x)$, such that $g(x, y) \simeq \phi_{k(x)}(y)$.

Then from the results of (i) and (ii), $g(e, y) = \phi_{k(e)}(y) = \phi_e^{-1}(y) = f^{-1}(y)$ if $f$ is injective.

□

4. (cf. example 3-7.1(b)) Suppose that $f$ and $g$ are unary computable functions; assuming that $T_1$ has been formally proved to be decidable, prove formally that the function $h(x)$ defined by $h(x) = \begin{cases} 1 & \text{if } x \in Dom(f) \text{ or } x \in Dom(g), \\ \text{undefined} & \text{otherwise,} \end{cases}$ is computable.

**Sol.** Since $f$ and $g$ are computable, there exists $e_1$ and $e_2$ such that $f \simeq \phi_{e_1}$ and $g \simeq \phi_{e_2}$.

From Kleene's normal form theorem, there exists two decidable predicates $T_1(e_1, x, z)$ and $T_2(e_2, x, z)$, such that $f(x)$ is defined iff. $\exists z\ T_1(e_1, x, z)$ and $g(x)$ is defined iff. $\exists z\ T_2(e_2, x, z)$.

Therefore, $x \in Dom(f)$ or $x \in Dom(g)$ iff. $(\exists z)(T_1(e_1, x, z) \bigvee T_2(e_2, x, z))$.

Since $T$ is proved to be decidable, from the computability of minimalization, $h(x)$ is computable. □

5. Prove the equivalent of example 5 in Chapter 5-3.1 for the operations of substitution and minimalisation, namely:

   (a) Fix $m, n \geq 1$; there is a total computable function $s(e, e_1, \ldots, e_m)$ such that (in the notation of theorem 2.2) $\phi^{(n)}_{s(e,e_1,\ldots,e_m)} = Sub(\phi^{(m)}_e; \phi^{(n)}_{e_1}, \phi^{(n)}_{e_2}, \ldots, \phi^{(n)}_{e_m})$.

   **Sol.** Let $(mn + m + 1) - ary$ function $f$ be

$$\begin{aligned} f(e, e_1, e+2, \ldots, e_m, \mathbf{x_1}, \ldots, \mathbf{x_m}) &= \phi_e(\phi_{e_1}(\mathbf{x_1}), \phi_{e_2}(\mathbf{x_2}), \ldots, \phi_{e_m}(\mathbf{x_m})) \quad (0.1) \\ &= \psi_U(e, \phi_{e_1}(\mathbf{x_1}), \phi_{e_2}(\mathbf{x_2}), \ldots, \phi_{e_m}(\mathbf{x_m})) \quad (0.2) \end{aligned}$$

   From the computability of the universal function and substitution, we get $f$ is computable.

   Therefore, according to s-m-n theorem, there exists a total computable function $s(e, e_1, \ldots, e_m)$, such that $\phi_{s(e,e_1,\ldots,e_m)} \simeq f(e, e_1, e+2, \ldots, e_m, \mathbf{x_1}, \ldots, \mathbf{x_m})$.

   Hence $s(e, e_1, \ldots, e_m)$ is an index for the function obtained by substitution, i.e. $\phi^{(n)}_{s(e,e_1,\ldots,e_m)} = Sub(\phi^{(m)}_e; \phi^{(n)}_{e_1}, \phi^{(n)}_{e_2}, \ldots, \phi^{(n)}_{e_m})$. □

   (b) Fix $n \geq 1$; there is a total computable function $k(e)$ such that for all $e$, $\phi^{(n)}_{k(e)}(\mathbf{x}) \simeq \mu y(\phi^{(n+1)}_e(\mathbf{x}, y) = 0)$. (We could extend the notation of theorem 2.2 in the obvious way and write $\phi^{(n)}_{k(e)} = Min(\phi^{(n+1)}_e)$.)

   **Sol.** Let $f$ be

$$f(e, \mathbf{x}, y) \simeq \mu y(\phi^{(n+1)}_e(\mathbf{x}, y) = 0) \simeq \mu y(\psi_U(e, \mathbf{x}, y) = 0)$$

   According to the computability of minimalization (Theorem 2-5.2), $f$ is computable.

   Then from s-m-n theorem, there exists a total computable function $k(e)$ such that $\phi^{(n)}_{k(e)}(\mathbf{x}) \simeq f(e, \mathbf{x}, y)$, i.e. $\phi^{(n)}_{k(e)} = Min(\phi^{(n+1)}_e)$. □

6. Show that the following problems are undecidable.

   (a) '$x \in E_x$' (*Hint.* Either use a direct diagonal construction, or reduce '$x \in W_x$' to this problem using the s-m-n theorem).

   **Sol.** Let $g(x, y)$ be

$$g(x, y) = \begin{cases} y & \text{if } x \in W_x, \\ \text{undefined} & \text{otherwise,} \end{cases}$$

   $g$ is computable by Church's Thesis and hence by s-m-n theorem, there exists a total computable function $k(x)$, such that $g(x, y) \simeq \phi_{k(x)}(y)$.

   Then $x \in W_x \Rightarrow E_{k(x)} = \mathbb{N} \Rightarrow x \in E_{k(x)}$.

   and $x \notin W_x \Rightarrow E_{k(x)} = \varnothing \Rightarrow x \notin E_{k(x)}$

   Therefore, $x \in E_{k(x)}$ iff. $x \in W_x$, hence $x \in E_{k(x)}$ is not decidable and so is $x \in E_x$. □

   (b) '$W_x = W_y$' (*Hint.* Reduce '$\phi_x$ is total' to this problem).

   **Sol.** Let $f(x, y)$ be the characteristic function of '$W_x = W_y$' and let $c$ be the Godel number of zero function $\mathbf{0}$.

   Then $g(x) = f(x, c)$ is the characteristic function of '$W_x = \mathbb{N}$' which is not computable since '$\phi_x$ is total' is not decidable.

   Therefore, $f(x, y)$ is also not computable and hence '$W_x = W_y$' is not decidable. □

(c) '$\phi_x(y) = 0$'.

**Sol.** Let $f(x, y)$ be

$$f(x,y) = \begin{cases} 0 & \text{if } x \in W_x, \\ \text{undefined} & \text{otherwise,} \end{cases}$$

Since $f$ is computable by Church's Thesis, according to s-m-n theorem, there is a total computable function $k(x)$, such that $\phi_{k(x)}(y) \simeq f(x, y)$.

Then $x \in W_x \Leftrightarrow \phi_{k(x)}(y) = 0$.

Therefore, $\phi_{k(x)}(y) = 0$ is not decidable hence so is $\phi_x(y) = 0$. □

(d) '$E_x$ is infinite'.

**Sol.** Let

$$f(x,y) = \begin{cases} y & \text{if } x \in W_x, \\ \text{undefined} & \text{otherwise,} \end{cases}$$

Since $f$ is computable by Church's Thesis, according to s-m-n theorem, there is a total computable function $k(x)$, such that $\phi_{k(x)}(y) \simeq f(x, y)$.

Then $x \in W_x \Rightarrow E_{k(x)} = \mathbb{N} \Rightarrow E_{k(x)}$ is infinite.

and $x \notin W_x \Rightarrow E_{k(x)} = \varnothing \Rightarrow E_{k(x)}$ is not infinite

Therefore '$E_{k(x)}$ is infinite' is not decidable hence so is '$E_x$ is infinite'.

□

(e) '$\phi_x = g$', where $g$ is any fixed computable function.

**Sol.** If it is decidable, then let $g = \mathbf{0}$, which results that $\phi_x = \mathbf{0}$ is decidable. It is thus contradictory, hence '$\phi_x = g$' is not decidable. □

7. Show that there is no total computable function $f(x, y)$ with the following property: if $P_x(y)$ stops, then it does so in $f(x, y)$ or fewer steps. (*Hint.* Show that if such a function exists, then the Halting problem is decidable.)

**Sol.** We prove it by *reductio ad absurdum*. Assume such $f(x, y)$ exists, since $f(x, y)$ is total, for any $x$ and $y$, we have '$p_x(y)$ is defined' iff. '$H_n(x, y, f(x, y)) = 0$'.

The reason is that if $P_x(y)$ stops, then it does so in $f(x, y)$ or fewer steps. Therefore, if $H_n(x, y, f(x, y)) \neq 0$, then $P_x(y)$ never stops.

And since '$H_n(x, y, f(x, y)) = 0$' is decidable, hence so is '$p_x(y)$ is defined'. Thus we have proved the decidabliity of Halting problem, which leads to contradiction. Therefore, there isn't such $f(x, y)$. □

8. Show that the following predicates are partially decidable:

(a) '$E_x^{(n)} \neq \varnothing$' ($n$ fixed).

**Sol.** '$E_x^{(n)} \neq \varnothing$' iff. '$(\exists y)(\exists z)(\phi_x(z) = y)$'.

We first prove that predicate '$\phi_x(z) = y$' is partially decidable. This can be shown by noting that '$\phi_x(z) = y$' iff. '$(\exists t)S_n(x, z, y, t)$', where $S_n$ is a decidable predicate. Therefore, according to Theorem 6.4, '$\phi_x(z) = y$' is partially decidable.

Then, according to Corollary 6.6, since '$\phi_x(z) = y$' is partially decidable, we could deduce that '$(\exists y)(\exists z)(\phi_x(z) = y)$' is also partially decidable, which finishes our proof. □

(b) '$n$ is a Fermat number'. (We say that $n$ is a *Fermat number* if there is a number $x$, $y$, $z > 0$ such that $x^n + y^n = z^n$.)

**Sol.** '$n$ is a Fermat number' iff. '$(\exists x)(\exists y)(\exists z)(x^n + y^n = z^n)$'

It is apparent that predicate '$(x^n + y^n = z^n)$' is decidable given $x$,$y$ and $z$.

Thus according to Corollary 6.6, '$(\exists x)(\exists y)(\exists z)(x^n + y^n = z^n)$' is partially decidable, hence so is '$n$ is a Fermat number'. □

(c) '$M(\mathbf{x})$ and $N(\mathbf{x})$', if $M(\mathbf{x})$ and $N(\mathbf{x})$ are partially decidable.

**Sol.** Let $P(\mathbf{x})$ and $Q(\mathbf{x})$ be the partial decision procedures of $M$ and $N$ respectively.

Since $M$ and $N$ are both partially decidable, for each $x$ where $M$ or $N$ holds, $P(x)$ or $Q(x)$ will converge correspondingly.

Therefore, if we concatenate the standard form of $P$ and $Q$ to form a new program $R$, $R$ will converge iff. $P$ and $Q$ both converge i.e. '$M(\mathbf{x})$ and $N(\mathbf{x})$' holds.

Then from the definition of partial decidability, '$M(\mathbf{x})$ and $N(\mathbf{x})$' is partially decidable. □

9. This exercise shows how the technique of reducibility (Chapter 6 §1) may be used to show that a predicate is not partially decidable.

(a) Suppose that $M(x)$ is a predicate and $k$ a total computable function such that $x \in W_x$ iff $M(k(x))$ does not hold. Prove that $M(x)$ is not partially decidable.

**Sol.** $x \in W_x$ iff '$M(k(x))$ does not hold' implies that $x \notin W_x$ iff '$M(k(x))$ holds'.

Now assume $M(x)$ is partially decidable, whose partial characteristic function is $f(x)$. Then $M(k(x))$ is partially decidable since its partial characteristic function $f(k(x))$ is computable by substitution.

Since $x \notin W_x$ iff '$M(k(x))$ holds', $f(k(x))$ is also the partial characteristic function for '$x \notin W_x$', thus indicating that '$x \notin W_x$' is partially decidable, which leads to contradiction.

Therefore, $M(x)$ is not partially decidable. □

(b) Prove that '$\phi_x$ is not total' is not partially decidable.
(Hint. Consider the function $k$ in the proof of theorem 1.6.)

**Sol.** Let

$$f(x,y) = \begin{cases} y & \text{if } x \in W_x, \\ \text{undefined} & \text{otherwise,} \end{cases}$$

$f(x,y)$ is computable since '$x \in W_x$' is partially decidable. Then from s-m-n theorem, there is a total computable function $k(x)$, such that $\phi_{k(x)}(y) \simeq f(x,y)$.

From the definition of $f$, we have

$$x \in W_x \Rightarrow W_{k(x)} = \mathbb{N} \Rightarrow \phi_{k(x)} \text{ is total}$$

and

$$x \notin W_x \Rightarrow W_{k(x)} = \varnothing \Rightarrow \phi_{k(x)} \text{ is not total}$$

Thereby, we have $w \in W_x$ iff. $\phi_{k(x)}$ is total. Equivalently, we get $w \notin W_x$ iff. $\phi_{k(x)}$ is not total.

And '$w \notin W_x$' is not partially decidable, hence '$\phi_{k(x)}$ is not total' is also not partially decidable, thus nor is '$\phi_x$ is not total'. □

(c) By considering the function

$$f(x,y) = \begin{cases} 1 & \text{if } P_x(x) \text{ does not converge in } y \text{ or fewer steps,} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Show that '$\phi_x$ is total' is not partially decidable. (*Hint.* Use the s-m-n theorem and (a).)

**Sol.** Let

$$f(x,y) = \begin{cases} 1 & \text{if } P_x(x) \text{ does not converge in } y \text{ or fewer steps,} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Since $f(x,y)$ is computable by Church's Thesis, from s-m-n theorem, there is a total computable function $k(x)$, such that $\phi_{k(x)}(y) \simeq f(x,y)$.

From the definition of $f$, we have

$$x \in W_x \Rightarrow (\exists y)(P_x(x) \text{ converges in } y \text{ steps}) \Rightarrow \phi_{k(x)} \text{ is not total}$$

and

$$x \notin W_x \Rightarrow (\forall y)(P_x(x) \text{ does not converge in } y \text{ steps}) \Rightarrow \phi_{k(x)} \text{ is total}$$

Therefore, '$x \in W_x$' iff. '$\phi_{k(x)}$ is not total'. From (a), '$\phi_x$ is total' is not partially computable. □

# Lab4-Various Sets

Name: Yang Fei December 17, 2012

1. Let $A, B$ be subsets of $\mathbb{N}$. Define sets $A \oplus B$ and $A \otimes B$ by

$$A \oplus B = \{2x : x \in A\} \cup \{2x+1 : x \in B\},$$
$$A \otimes B = \{\pi(x,y) : x \in A \text{ and } y \in B\},$$

where $\pi$ is the pairing function $\pi(x,y) = 2^x(2y+1) - 1$ of Theorem 4-1.2. Prove that

(a) $A \oplus B$ is recursive iff $A$ and $B$ are both recursive.

**Sol.** $\Rightarrow$:

$$x \in A \Leftrightarrow 2x \in A \oplus B$$
$$x \in B \Leftrightarrow 2x+1 \in A \oplus B$$

For $A \oplus B$ is recursive, hence, $A$ and $B$ are both recursive.
$\Leftarrow$:
If $x$ is an even number,

$$x \in A \oplus B \Leftrightarrow \frac{x}{2} \in A$$

If $x$ is an odd number,

$$x \in A \oplus B \Leftrightarrow \frac{x-1}{2} \in B$$

For $A$ and $B$ are both recursive, hence, $A \oplus B$ is recursive. □

(b) If $A, B \neq \varnothing$, then $A \otimes B$ is recursive iff $A$ and $B$ are both recursive.

**Sol.** $\Rightarrow$:

$$x \in A \Leftrightarrow z = \pi(x,y) \in A \otimes B(\text{randomly pick y from B})$$
$$y \in B \Leftrightarrow z = \pi(x,y) \in A \otimes B(\text{randomly pick x from A})$$

For $\pi$ is computable, hence, $A$ and $B$ are both recursive if $A \otimes B$ is recursive.
$\Leftarrow$:

$$z \in A \otimes B \Leftrightarrow \pi_1(z) \in A \wedge \pi_2(z) \in B$$

let $x = \pi_1(z)$, $y = \pi_2(z)$,

$$z \in A \otimes B \Leftrightarrow x \in A \wedge y \in B$$

For both $\pi_1$ and $\pi_2$ are computable functions, we conclude that $A \otimes B$ is recursive if $A$ and $B$ are both recursive. □

(c) Suppose $B$ is r.e. If $A$ is creative, then so are $A \oplus B$ and $A \otimes B$ (provided $B \neq \varnothing$).

**Sol.** $A$ is r.e. and $\overline{A}$ is productive, since $A$ is creative.

If $x$ is an even number,

$$x \in A \oplus B \;\Leftrightarrow\; \frac{x}{2} \in A$$

If $x$ is an odd number,

$$x \in A \oplus B \;\Leftrightarrow\; \frac{x-1}{2} \in B$$

For $A$ and $B$ are both r.e., $A \oplus B$ is also r.e.

$$\begin{aligned} x \in \overline{A} \Leftrightarrow x \notin A &\;\Leftrightarrow\; 2x \notin A \oplus B \Leftrightarrow 2x \in \overline{A \oplus B} \\ x \in \overline{A} &\;\Leftrightarrow\; 2x \in \overline{A \oplus B} \end{aligned}$$

Since $\overline{A}$ is productive, hence, $\overline{A \oplus B}$ is productive according to Theorem-3.2.

Therefore, $A \oplus B$ is creative.

$$z \in A \otimes B \;\Leftrightarrow\; \pi_1(z) \in A \wedge \pi_2(z) \in B$$

Since both $A$ and $B$ are r.e., $A \otimes B$ is also r.e.

For $B \neq \emptyset$, randomly pick $y$ from $B$, such that

$$\begin{aligned} x \in \overline{A} \Leftrightarrow x \notin A &\;\Leftrightarrow\; \pi(x,y) \notin A \otimes B \Leftrightarrow \pi(x,y) \in \overline{A \otimes B} \\ x \in \overline{A} &\;\Leftrightarrow\; \pi(x,y) \in \overline{A \otimes B} \end{aligned}$$

Since $\overline{A}$ is productive, hence, $\overline{A \otimes B}$ is productive according to Theorem-3.2.

Therefore, $A \otimes B$ is creative.

□

(d) If $A$ is simple, then $A \otimes \mathbb{N}$ is r.e., but neither recursive, creative nor simple.

**Sol.** $x \in A \otimes \mathbb{N}$ iff. $\pi_1(x) \in A$, which is partially decidable by substitution. since $A$ is r.e.
Therefore $A \otimes \mathbb{N}$ is r.e.

Since $A$ is simple, which means that $A$ is not recursive, then so is $A \otimes \mathbb{N}$ from (b).

If $A \otimes \mathbb{N}$ is creative, then $\overline{A \otimes \mathbb{N}}$ is productive. And $x \in \overline{A \otimes \mathbb{N}}$ iff. $\pi_1(x) \in \overline{A}$, then we get $\overline{A}$ is productive by reduction, which in turn implies that $A$ is creative, thus leading to contradiction. Therefore $A \otimes \mathbb{N}$ is not creative.

We have $\overline{A \otimes \mathbb{N}} = \{\pi(x,y) | x \notin A\}$. Since $A$ is simple, there is an $a \notin A$ for $\overline{A}$ is infinite. Since both $\{a\}$ and $\mathbb{N}$ are r.e., we have $\{a\} \otimes \mathbb{N} \subset \overline{A \otimes \mathbb{N}}$ is r.e. This means that $\overline{A \otimes \mathbb{N}}$ has an infinite r.e. subset, indicating it is not simple. □

(e) If $A$, $B$ are simple sets, then $A \oplus B$ is simple, $A \otimes B$ is not simple but $\overline{\overline{A} \otimes \overline{B}}$ is simple.

**Sol.** From (c), $A \oplus B$ is r.e.
Since $\overline{A \oplus B} = \overline{A} \oplus \overline{B}$, and both $\overline{A}$ and $\overline{B}$ are infinite, then $\overline{A \oplus B} = \overline{A} \oplus \overline{B}$ is infinite.
Suppose there is a r.e. set $C \subseteq \overline{A \oplus B}$.

Since $C$ is r.e., then both $C_e = \{x | x \text{ is even and } 2x \in C\}$ and $C_o = \{x | x \text{ is odd and } 2x+1 \in C\}$ are r.e.
If $C \subseteq \overline{A} \oplus \overline{B}$, from definition, either $C_e \subseteq \overline{A}$ or $C_o \subseteq \overline{B}$, which leads to contradiction since $A$ and $B$ are simple.
Therefore, $\overline{A \oplus B}$ contains no r.e. subset, hence $A \oplus B$ is simple.

To show that $A \otimes B$ is not simple, since $\overline{A}$ is infinite, there is an $a \notin A$.
Since both $\{a\}$ and $\mathbb{B}$ are r.e., we have $\{a\} \otimes \mathbb{B} \subset \overline{A \otimes \mathbb{B}}$ is r.e. Therfore, $\overline{A \otimes \mathbb{B}}$ has an infinite r.e. subset, indicating it is not simple.

$\overline{\overline{A} \otimes \overline{B}} = \{\pi(x, y) | x \in A \vee y \in B\}$, then $x \in \overline{\overline{A} \otimes \overline{B}} \Leftrightarrow \pi_1(x) \in A \vee \pi_2(x) \in B$, which is partially decidable since both $A$ and $B$ are r.e. Therefore $\overline{\overline{A} \otimes \overline{B}}$ is r.e.
And since $\overline{A}$ and $\overline{B}$ are both infinite, $\overline{A} \otimes \overline{B}$ is infinite.
Futhermore, if there is a r.e. set $C \subseteq \overline{A} \otimes \overline{B}$, then $\pi_1(C) \subseteq \overline{A}$, which is contradictory to the fact that $A$ is simple. Hence $\overline{A} \otimes \overline{B}$ contains no r.e. subset.
Therefore, $\overline{\overline{A} \otimes \overline{B}}$ is simple.

□

2. (a) Let $B \subseteq \mathbb{N}$ and $n > 1$; prove if $B$ is recursive (or r.e.) then the predicate $M(x_1, \ldots, x_n)$ given by "$M(x_1, \ldots, x_n) \equiv 2^{x_1}3^{x_2} \ldots p_n^{x_n} \in B$" is decidable (or partially decidable).

**Sol.** case 1: $B$ is recursive

$$f(x) = \begin{cases} 1 & \text{if } x \in B, \\ 0 & \text{if } x \notin B. \end{cases}$$

is the characteristic function of $B$. For $B$ is recursive, hence $f(x)$ is computable.
The characteristic function of $M$ is

$$g(x) = \begin{cases} 1 & \text{if } f(2^{x_1}3^{x_2} \ldots p_n^{x_n}) = 1, \\ 0 & \text{if } f(2^{x_1}3^{x_2} \ldots p_n^{x_n}) = 0. \end{cases}$$

For power is computable, by substitution, $f(2^{x_1}3^{x_2} \ldots p_n^{x_n})$ is computable, hence $g(x)$ is computable.
Thus, the predicate $M(x_1, \ldots, x_n)$ given by "$M(x_1, \ldots, x_n) \equiv 2^{x_1}3^{x_2} \ldots p_n^{x_n} \in B$" is decidable.
case 2: $B$ is r.e.

$$f(x) = \begin{cases} 1 & \text{if } x \in B, \\ undefined & \text{if } x \notin B. \end{cases}$$

is the partial characteristic function of $B$. For $B$ is r.e., hence $f(x)$ is computable.
The partial characteristic function of $M$ is

$$g(x) = \begin{cases} 1 & \text{if } f(2^{x_1}3^{x_2} \ldots p_n^{x_n}) = 1, \\ undefined & \text{otherwise} \end{cases}$$

For power is computable, by substitution, $f(2^{x_1}3^{x_2} \ldots p_n^{x_n})$ is computable, hence $g(x)$ is computable.
Thus, the predicate $M(x_1, \ldots, x_n)$ given by "$M(x_1, \ldots, x_n) \equiv 2^{x_1}3^{x_2} \ldots p_n^{x_n} \in B$" is partially decidable. □

(b) Prove that $A \subseteq \mathbb{N}^n$ is recursive (or r.e.) iff $\{2^{x_1}3^{x_2}\ldots p_n^{x_n} : (x_1,\ldots,x_n) \in A\}$ is recursive (or r.e., respectively).

**Sol.** Define $B \subseteq \mathbb{N}$ to be $\{2^{x_1}3^{x_2}\ldots p_n^{x_n} : (x_1,\ldots,x_n) \in A\}$.

case 1: $A \subseteq \mathbb{N}^n$ is recursive

$\Rightarrow$:

$$f(x_1,\ldots,x_n) = \begin{cases} 1 & \text{if } (x_1,\ldots,x_n) \in A, \\ 0 & \text{if } (x_1,\ldots,x_n) \notin A. \end{cases}$$

is the characteristic function of $A$. For $A$ is recursive, hence $f(x_1,\ldots,x_n)$ is computable. The characteristic function of $B$ is

$$g(x) = \begin{cases} 1 & \text{if } f((x)_1,(x)_2\ldots(x)_n) = 1, \\ 0 & \text{if } f((x)_1,(x)_2\ldots(x)_n) = 0. \end{cases}$$

For $(x)_y$ is computable, by substitution, $f((x)_1,(x)_2\ldots(x)_n)$ is computable, hence $g(x)$ is computable.

Thus, $B$ is recursive.

$\Leftarrow$:

$$g(x) = \begin{cases} 1 & \text{if } x \in B, \\ 0 & \text{if } x \notin B. \end{cases}$$

is the characteristic function of $B$. For $B$ is recursive, hence $g(x)$ is computable. The characteristic function of $A$ is

$$f(x_1,\ldots,x_n) = \begin{cases} 1 & \text{if } g(2^{x_1}3^{x_2}\ldots p_n^{x_n}) = 1, \\ 0 & \text{if } g(2^{x_1}3^{x_2}\ldots p_n^{x_n}) = 0. \end{cases}$$

For power is computable, by substitution, $g(2^{x_1}3^{x_2}\ldots p_n^{x_n})$ is computable, hence $f(x_1,\ldots,x_n)$ is computable.

Thus, $A$ is recursive.

case 2: $A \subseteq \mathbb{N}^n$ is r.e.

$\Rightarrow$:

$$f(x_1,\ldots,x_n) = \begin{cases} 1 & \text{if } (x_1,\ldots,x_n) \in A, \\ undefined & \text{if } (x_1,\ldots,x_n) \notin A. \end{cases}$$

is the partial characteristic function of $A$. For $A$ is r.e., hence $f(x_1,\ldots,x_n)$ is computable. The partial characteristic function of $B$ is

$$g(x) = \begin{cases} 1 & \text{if } f((x)_1,(x)_2\ldots(x)_n) = 1, \\ undefined & \text{otherwise} \end{cases}$$

For $(x)_y$ is computable, by substitution, $f((x)_1,(x)_2\ldots(x)_n)$ is computable, hence $g(x)$ is computable.

Thus, $B$ is r.e.

$\Leftarrow$:

$$g(x) = \begin{cases} 1 & \text{if } x \in B, \\ undefined & \text{if } x \notin B. \end{cases}$$

is the partial characteristic function of $B$. For $B$ is r.e., hence $g(x)$ is computable.
The partial characteristic function of $A$ is

$$f(x_1,\ldots,x_n)=\begin{cases}1 & \text{if } g(2^{x_1}3^{x_2}\ldots p_n^{x_n})=1,\\ undefined & \text{otherwise}\end{cases}$$

For power is computable, by substitution, $g(2^{x_1}3^{x_2}\ldots p_n^{x_n})$ is computable, hence $f(x_1,\ldots,x_n)$ is computable.

Thus, $A$ is r.e. □

(c) Prove that $A\subseteq\mathbb{N}^n$ is r.e. iff $A=\varnothing$ or there is a total computable function $\boldsymbol{f}:\mathbb{N}\to\mathbb{N}^n$ such that $A=Ran(\boldsymbol{f})$. (A *computable function* $\boldsymbol{f}$ from $\mathbb{N}$ to $\mathbb{N}^n$ is an $n$-tuple $\boldsymbol{f}=(f_1,\ldots,f_n)$ where each $f_i$ is a unary computable function and $\boldsymbol{f}(x)=(f_1(x),\ldots,f_n(x))$.)

**Sol.** According to the conclusion from $2(b)$, we get

$$A\subseteq\mathbb{N}^n \text{ is r.e. } \Leftrightarrow \ B=\{2^{x_1}3^{x_2}\ldots p_n^{x_n}:(x_1,\ldots,x_n)\in A\} \text{ is r.e.}$$

By Listing Theorem,

$$\begin{aligned}B \text{ is r.e. } &\Leftrightarrow \text{ either } B=\emptyset \text{ or } B \text{ is the range of a unary total computable function.}\\ &\Leftrightarrow \ B=\emptyset \text{ or there exists a total computable function } g,\ B=Ran(g)\\ &\Leftrightarrow \ A=\emptyset \text{ or } A=Ran(\boldsymbol{f})\end{aligned}$$

where $\boldsymbol{f}=((g)_1,(g)_2,\ldots,(g)_n)$ and $\boldsymbol{f}$ is a total computable function.

Therefore, $A\subseteq\mathbb{N}^n$ is r.e. iff $A=\varnothing$ or there is a total computable function $\boldsymbol{f}:\mathbb{N}\to\mathbb{N}^n$ such that $A=Ran(\boldsymbol{f})$. □

3. Which of the following sets are recursive? Which are r.e.? Which are productive? Which are creative? Prove your judgements.

(a) $\{x:x\in E_x\}$,

**Sol.** It is creative.

According to Equivalence Theorem, it is r.e.

Let $f(x,y)=\begin{cases}y & \text{if } x\in W_x,\\ \text{undefined} & \text{otherwise,}\end{cases}$

According to s-m-n theorem, there is a total computable function $k(x)$, such that $\phi_{k(x)}(y)\simeq f(x,y)$.

Then

$$x\in W_x\Leftrightarrow E_{k(x)}=\mathbb{N}\Leftrightarrow x\in E_{k(x)}$$

$$x\notin W_x\Leftrightarrow E_{k(x)}=\varnothing\Leftrightarrow x\notin E_{k(x)}$$

Since $\bar{K}$ is productive, hence so is $\{x|x\notin E_x\}$.

Therefore $\{x:x\in E_x\}$ is creative. □

(b) $\{x:x$ is a perfect square$\}$,

**Sol.** It is recursive, since '$x$ is a perfect square' is a decidable predicate. □

(c) $\{x:\phi_x$ is not injective$\}$,

**Sol.** It is creative.

$\phi_x$ is not injective $\Leftrightarrow (\exists y)(\exists z)(\phi_x(y) = \phi_x(z))$
$\Leftrightarrow (\exists y)(\exists z)(\exists w)(\exists t)(S_n(x, y, w, t) \bigwedge S_n(x, z, w, t))$
which is partially decidable.

Therefore, $A = \{x : \phi_x \text{ is not injective}\}$ is r.e.

On the other hand, let $B = \{\phi_x : \phi_x \text{ is injective}\}$. Clearly $f_\phi \in B$, hence $\overline{A}$ is productive.

Therefore $A$ is creative. □

(d) $\{x : \phi_x \text{ is not surjective}\}$,

**Sol.** It is productive.

Denote $\{\phi_x : \phi_x \text{ is not surjective}\}$ as $B$, since $f_\phi \in B$, we have $\{x : \phi_x \text{ is not surjective}\}$ is productive. □

(e) $\{x : \phi_x(x) = f(x)\}$, where $f$ is any total computable function.

**Sol.** It is creative.

According to the Theorem 3.8, we only need to show that $A = \{x : \phi_x(x) = f(x)\}$ is r.e.

$\phi_x(x) = f(x) \Leftrightarrow (\exists t) S_n(x, x, f(x), t)$
which is partially decidable, hence $A$ is r.e. Furthermore, $A$ is creative by Theorem 3.8. □

4. Suppose $A$ is an r.e. set. Prove the following statements.

(a) Show that the sets $\bigcup\limits_{x\in A} W_x$ and $\bigcup\limits_{x\in A} E_x$ are both r.e.

**Sol.**

$$y \in \bigcup_{x\in A} W_x \Leftrightarrow (\exists x)(x \in A \wedge y \in W_x)$$

which is partially decidable since both '$x \in A$' and '$y \in W_x$' are partially decidable.

$$y \in \bigcup_{x\in A} E_x \Leftrightarrow (\exists x)(x \in A \wedge y \in E_x)$$

which is also partially decidable.

Therefore, $\bigcup\limits_{x\in A} W_x$ and $\bigcup\limits_{x\in A} E_x$ are both r.e. □

(b) Show that $\bigcap\limits_{x\in A} W_x$ is not necessarily r.e. (*Hint*: $\forall t \in \mathbb{N}$ let $K_t = \{x : P_x(x) \downarrow \text{ in t steps}\}$. Show that for any $t$, $K_t$ is recursive; moreover $K = \bigcup\limits_{t\in\mathbb{N}} K_t$ and $\overline{K} = \bigcap\limits_{t\in\mathbb{N}} \overline{K}_t$.)

**Sol.** Let $K_t = \{x : P_x(x) \downarrow \text{ in t steps}\}$.

Since $P_x(x) \downarrow$ in t steps is decidable, $K_t$ is recursive, hence so is $\overline{K_t}$.

From definition of $K_t$, we could get that $K = \bigcup\limits_{t\in\mathbb{N}} K_t$ and thus $\overline{K} = \bigcap\limits_{t\in\mathbb{N}} \overline{K}_t$ by De Morgen Rule.

Therefore, $\bigcap\limits_{t\in\mathbb{N}} \overline{K}_t$ is not r.e.

According to s-m-n theorem, there is a total computable function $k(t)$, such that

$$\phi_{k(t)}(x) = \begin{cases} 0 & \text{if } P_x(x) \not\downarrow \text{ in t steps,} \\ \text{undefined} & \text{otherwise,} \end{cases}$$

Then $W_{k(t)} = \overline{K_t}$.

Therefore, $\bigcap_{t\in\mathbb{N}} \overline{K}_t = \bigcap_{t\in\mathbb{N}} W_{k(t)} = \bigcap_{x\in Ran(k)} W_x$.

If $\bigcap_{x\in A} W_x$ is r.e., we could get that $\bigcap_{t\in\mathbb{N}} \overline{K}_t$ is also r.e., which leads to contradiction.

Hence $\bigcap_{x\in A} W_x$ is not r.e. □

5. Suppose that $f$ is a total computable function, $A$ is a recursive set and $B$ is an r.e.set. Show that $f^{-1}(A)$ is recursive and that $f(A)$, $f(B)$ and $f^{-1}(B)$ are r,e, but not necessarily recursive. What extra information about these sets can be obtained if $f$ is a bijection?

**Sol.** $x \in f^{-1}(A) \Leftrightarrow f(x) \in A$
Since $A$ is recursive, then $f(x) \in A$ is decidable by substitution. Therefore $f^{-1}(A)$ is recursive.

Since $A$ and $B$ are both r.e., there exists total computable function $g$ and $h$, where $A = Ran(g)$ and $B = Ran(h)$.

Then $f(A) = Ran(f(g(x)))$ and $f(B) = Ran(f(h(x)))$, which are both r.e. by Listing Theorem.

$x \in f^{-1}(B) \Leftrightarrow f(x) \in B$
Since $B$ is r.e., then $f(x) \in B$ is partially decidable by substitution. Therefore $f^{-1}(B)$ is r.e.

Let $B$ be a non-recursive r.e. set and $f$ be the identity function, then $f(B) = f^{-1}(B) = B$ are not recursive.

Let $A = \mathbb{N}$, and

$$f(x) = \begin{cases} x & \text{if } x \in W_x, \\ \text{undefined} & \text{otherwise,} \end{cases}$$

Then $f(A) = K$, which is not recursive.

If $f$ is a bijection, then $f^{-1}$ is a total computable function.

Therefore, we can further obtain that $f(A)$ is also recursive.

□

6. Prove Rice's theorem (Theorem 6-1.7) from Rice-Shapiro theorem (Theorem 7-2.16). (*Hint.* Suppose that '$\phi_x \in \mathscr{B}$' is decidable; then both $\mathscr{B}$ and $\mathscr{C}_1 \backslash \mathscr{B}$ satisfy the conditions of Rice-Shapiro: consider the cases $f_\varnothing \in \mathscr{B}$ and $f_\varnothing \notin \mathscr{B}$.)

**Sol.** Suppose $\mathscr{B} \neq \varnothing, \mathscr{C}_1$.

Assume that '$\phi_x \in \mathscr{B}$' is decidable, then according to the complementation theorem, both $\mathscr{B}$ and $\mathscr{C}_1 \backslash \mathscr{B}$ are r.e.

If $f_\phi \in \mathscr{B}$, Then for any $f \in \mathscr{C}_1$, since $f_\phi \subseteq f$ and $f_\phi \in \mathscr{B}$, we have $f \in \mathscr{B}$. Therefore $\mathscr{B} = \mathscr{C}_1$, which leads to contradiction.

On the contrary, if $f_\phi \in \mathscr{C}_1 \backslash \mathscr{B}$, similarly we could get $\mathscr{B} = \varnothing$, which also leads to contradiction.

Therefore, '$\phi_x \in \mathscr{B}$' is not decidable. □

7. Let $\mathscr{B}$ be a set of unary computable functions, and suppose that $g \in \mathscr{B}$ is such that for all finite $\theta \subseteq g$, $\theta \notin \mathscr{B}$. Prove that the set $\{x : \phi_x \in \mathscr{B}\}$ is productive. (*Hint.* Follow the first part of the proof of the Rice-Shapiro theorem.)

**Sol.** Let $B = \{x : \phi_x \in \mathscr{B}\}$.

Define $f(x, y)$ as

$$f(x,y) = \begin{cases} g(y) & \text{if } \neg H_n(x,x,y), \\ \text{undefined} & \text{otherwise,} \end{cases}$$

According to s-m-n theorem, there is a total computable function $k(x)$, such that $\phi_{k(x)}(y) \simeq f(x, y)$.

Then

$$x \in W_x \Leftrightarrow \phi_{k(x)} \text{ is finite } \Leftrightarrow k(x) \notin B$$
$$x \notin W_x \Leftrightarrow \phi_{k(x)} = g \Leftrightarrow k(x) \in B$$

Therefore, $x \in \overline{K} \Leftrightarrow k(x) \in B$.

Since $\overline{K}$ is productive, $B$ is productive.

□

8. Prove the following statements.

   (a) If $B$ is r.e. and $A \cap B$ is productive, then $A$ is productive.

   **Sol.** We can define

   $$f(x,y) = \begin{cases} 1 & \text{if } y \in W_x \wedge y \in B \\ undefined & \text{otherwise.} \end{cases}$$

   According to the partial decidability of '$y \in W_x$' and '$x \in B$', $f(x, y)$ is computable. The s-m-n theorem provides a total computable function $s(x)$ such that $f(x, y) \simeq \phi_{s(x)}(y)$.

   $$W_{k(x)} \;=\; W_x \cap B$$

   For whenever $W_x \subseteq A$,

   $$\begin{aligned} W_x \subseteq A \;&\Rightarrow\; W_{k(x)} \subseteq A \cap B \Rightarrow g(k(x)) \in A \cap B \backslash W_{k(x)} \\ &\Rightarrow\; g(k(x)) \in A \cap B \backslash (W_x \cap B) = (A \backslash W_x) \cap B \\ &\Rightarrow\; g(k(x)) \in A \backslash W_x \end{aligned}$$

   where $g$ is the productive function of $A \cap B$. Hence, $g(k(x))$ is the productive function of $A$.

   Therefore, $A$ is productive. □

   (b) If $C$ is creative and $A$ is an r.e. set such that $A \cap C = \varnothing$, then $C \cup A$ is creative.

   **Sol.** For both $A$ and $C$ are r.e., $A \cup C$ is r.e. according to Theorem 2.13.

   We can define

   $$f(x,y) = \begin{cases} 1 & \text{if } y \in W_x \vee y \in A \\ undefined & \text{otherwise.} \end{cases}$$

According to the partial decidability of '$y \in W_x$' and '$x \in A$', $f(x, y)$ is computable. The s-m-n theorem provides a total computable function $s(x)$ such that $f(x, y) \simeq \phi_{s(x)}(y)$.

$$W_{k(x)} = W_x \cup A$$

For whenever $W_x \subseteq \overline{C} \cap \overline{A}$,

$$\begin{aligned} W_x \subseteq \overline{C} \cap \overline{A} &\Rightarrow W_x \subseteq \overline{C} \Rightarrow g(x) \in \overline{C} \backslash W_x \Rightarrow g(k(x)) \in \overline{C} \backslash W_{k(x)} \\ &\Rightarrow g(k(x)) \in \overline{C} \backslash (A \cup W_x) \Rightarrow g(k(x)) \in \overline{C} \cap \overline{A} \backslash W_x \end{aligned}$$

where $g$ is the productive function of $\overline{C}$. Hence, $g(k(x))$ is the productive function of $\overline{C} \cap \overline{A}$.

Therefore, $C \cup A$ is creative. □

(c) Every productive set contains an infinite recursive subset.

**Sol.** According to Theorem 3.11, a productive set contains an infinite r.e. subset.

Also, according to Theorem 2.15, every infinite r.e. set has an infinite recursive subset.

Combining the two statements above, we can conclude that every productive set contains an infinite recursive subset. □

9. (a) (Cf. Theorem 7-2.14) Let $A$ be an infinite r.e. set. Show that $A$ can be enumerated without repetitions by a total computable function.

**Sol.** According to Listing Theorem, there is a total computable function $f$, such that $A = Ran(f)$. Then, we can define a function

$h(x) = \begin{cases} h(0) = f(0) \\ h(x+1) = f(\mu z(f(z) > h(x))) \end{cases}$

Since $A$ is infinite, $h(x)$ is a total computable function which enumerate $A$ without repetition. Therefore, $A$ can be enumerated without repetitions by $g$, which is a total computable function. □

(b) Suppose $f$ is a total injective computable function such that $Ran(f)$ is not recursive ((a) showed that such functions abound). Show that $A = \{x : \exists y(y > x \wedge f(y) < f(x))\}$ is simple. (*Hint.* To see that $\overline{A}$ is infinite, assume the contrary and show that there would then be a sequence of numbers $y_0 < y_1 < y_2 < \ldots$ such that $f(y_0) > f(y_1) > f(y_2) > \ldots$ To see that $\overline{A}$ does not contain an infinite r.e. set $B$, suppose to the contrary that $B \subseteq \overline{A}$. Then show that the problem $z \in Ran(f)$ is decidable as follows. Given $z$, find $n \in B$ such that $f(n) > z$; now use the fact that $n \notin A$ to devise a finite procedure for testing whether $z \in Ran(f)$.)

**Sol.** For $f$ is a total injective computable function, such that $y > x \wedge f(y) < f(x)$ is a decidable predicate. According to Quantifier Contraction Theorem, $A$ is r.e.

Assume that $A$ is infinite, $\exists\{y_0, \ldots | y_0 < y_1 < y_2 < \ldots\}$ such that $f(y_0) > f(y_1) > f(y_2) > \ldots$. For $f$ is total, then $f(y_0)$ is finite, so $A$ can not be infinite. Hence $A$ is finite and $\overline{A}$ is infinite.

Suppose $\overline{A}$ contains an infinite r.e. set $B$. Given $z$, we can find $n \in B$ that $f(n) > z$. Since $n \in \overline{A}$, for any $x$ such that $x \leq n$ or $f(x) \geq f(n)$. Then for any $x$, if $f(x) < f(n)$, then $x \leq n$. Thus '$z \in Ran(f)$' is equal to the predicate '$\exists x(x \leq n \wedge f(x) = z)$'. Since $f(x)$ is total computable, by Church's thesis, '$\exists x(x \leq n \wedge f(x) = z)$' is decidable. Then $Ran(f)$ is recursive which leads to contradiction. Thus $\overline{A}$ does not contain an infinite

r.e. set.

Therefore, $A$ is simple.

□

10. Recursively Inseparable and Effectively Recursively Inseparable

Disjoint sets $A$, $B$ are said to be *recursively inseparable* if there is no recursive set $C$ such that $A \subseteq C$ and $B \subseteq \overline{C}$. Furthermore, $A$ and $B$ are said to be *effectively recursively inseparable* if there is a total computable function $f$ such that whenever $A \subseteq W_a$, $B \subseteq W_b$ and $W_a \cap W_b = \varnothing$ then $f(a,b) \notin W_a \cup W_b$ (see the right figure). Note: Recursive inseparability for a pair of disjoint sets corresponds to non-recursiveness for a single set; pair of recursively inseparable sets that are also r.e. correspond to r.e. sets that are not recursive.

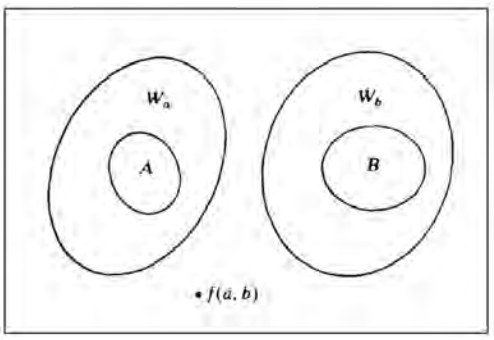



**Fig. Effectively Recursively Inseparable Sets**

(a) Show that two disjoint sets $A$, $B$ are recursively inseparable iff whenever $A \subseteq W_a$, $B \subseteq W_b$ and $W_a \cap W_b = \varnothing$, there is a number $x \notin W_a \cup W_b$.

**Sol.** $\Rightarrow$:
Assume that $\forall x(x \in W_a \cup W_b)$, which means $W_a \cup W_b = \mathbb{N}$. Then $W_b = \overline{W_a}$.

Let $C = W_a$, naturally $\overline{C} = W_b$, hence both $C$ and $\overline{C}$ are r.e. According to Theorem-2.6, $C$ is recursive, such that there is a recursive set $C$ s.t. $A \subseteq C$ and $B \subseteq \overline{C}$. Hence, $A$,$B$ are not recursively inseparable. Here, we get our contradiction.

Therefore, there is a number $x \notin W_a \cup W_b$.

$\Leftarrow$:

$$\exists x(x \notin W_a \cup W_b) \quad \Leftrightarrow \quad W_a \cup W_b \neq \mathbb{N} \text{ for any a,b}$$

Assume that there is a recursive set $C$ such that $A \subseteq C$ and $B \subseteq \overline{C}$. Since $C$ is recursive, both $C$ and $\overline{C}$ are r.e., which means that $C = W_i$ and $\overline{C} = W_j$ for some $i$,$j$.

$$W_i \cup W_j \quad = \quad C \cup \overline{C} = \mathbb{N}$$

Here we get our contradiction. Hence, there is no recursive set $C$ such that $A \subseteq C$ and $B \subseteq \overline{C}$.

Therefore, $A$, $B$ are recursively inseparable. □

(b) Suppose $A$, $B$ are effectively recursively inseparable. Prove that if $A$, $B$ are both r.e. then they are both creative. (*Note.* Extending the idea of effectiveness to a pair of recursively inseparable sets in this way parallels the step from a nonrecursive set to a set having productive complement; the counterpart to a single creative set is then a pair of effectively recursively separable sets that are both r.e.)

**Sol.** We need to show that both $\overline{A}$ and $\overline{B}$ are productive.
Suppose $W_x \subseteq \overline{A}$, then from s-m-n theorem, we could define

$$\phi_{k(x)}(y) = \begin{cases} 0 & \text{if } x \in W_x \vee x \in B, \\ \text{undefined} & \text{otherwise,} \end{cases}$$

Therefore $W_{k(x)} = W_x \cup B$.

Since $A$ is r.e., there is a number $a$, such that $A = W_a$. And since $B \subseteq W_{k(x)}$ and $A$ and $B$ are effectively recursively inseparable, $f(a, k(x)) \notin W_a \cup W_{k(x)}$.

Therefore, $f(a, k(x)) \in \overline{A}$ and $f(a, k(x)) \notin W_x$, which enables itself to be a candidate of a productive function of $\overline{A}$.

Consequently, $\overline{A}$ is productive and hence $A$ is creative. Similarly, we could formulate that $B$ is also creative.

□

(c) Let $K_0 = \{x : \phi_x(x) = 0\}$ and $K_1 = \{x : \phi_x(x) = 1\}$. Show that $K_0$ and $K_1$ are r.e. (in particular neither $K_0$ nor $K_1$ is recursive), and that they are both recursively inseparable and effectively recursively inseparable. (*Hint.* For recursively inseparable, suppose that there is such a set $C$ and let $m$ be an index for its characteristic function; consider whether or not $m \in C$. For effectively recursively inseparable, find a total computable function $f$ such that if $W_a \cap W_b = \varnothing$, then $\phi_{f(a,b)}(x) = \begin{cases} 1 & \text{if } x \in W_a, \\ 2 & \text{if } x \in W_b, \\ \text{undefined} & \text{otherwise.} \end{cases}$)

**Sol.** Since both '$\phi_x(x) = 0$' and '$\phi_x(x) = 1$' are partially decidable and not decidable, $K_0$ and $K_1$ are non-recursive r.e.

Suppose that $K_0$ and $K_1$ are not recursively separable, then there exists a recursive set $C$, such that $K_0 \subseteq C$ and $K_1 \subseteq \overline{C}$.

Let $m$ be an index for the characteristic function of $C$.

If $m \in C$, then by definition of characteristic function, $\phi_m(m) = 1$. But since $W_1 \subseteq \overline{C}$, so $\phi_m(m) \neq 1$.

If $m \notin C$, by definition of characteristic function, $\phi_m(m) = 0$. But since $W_0 \subseteq C$, so $\phi_m(m) \neq 0$.

Therefore we get a contradiction, hence such $C$ does not exist, indicating $K_0$ and $K_1$ are recursively inseparable

To prove $K_0$ and $K_1$ are effectively recursively inseparable, define

$$g(a, b, x) = \begin{cases} 1 & \text{if } x \in W_a, \\ 0 & \text{if } x \in W_b, \\ \text{undefined} & \text{otherwise,} \end{cases}$$

$g(a, b, x)$ is computable by Church's Thesis. Thus from s-m-n theorem, there is a total computable function $f(a, b)$, such that $\phi_{f(a,b)}(x) \simeq g(a, b, x)$.

Now we prove that for any $a, b$, if $K_0 \subseteq W_a$ and $K_1 \subseteq W_b$, and $W_a \cap W_b = \varnothing$, then $f(a, b) \notin W_a \cup W_b$.

If $f(a, b) \in W_a$, then $\phi_{f(a,b)}(f(a, b)) = 1$ by definition. But since $K_1 \subseteq W_b \subseteq \overline{W_a}$, $\phi_{f(a,b)}(f(a, b)) \neq 1$. Therefore $f(a, b) \notin W_a$.

Similarly, if $f(a, b) \in W_b$, then $\phi_{f(a,b)}(f(a, b)) = 0$ by definition. But since $K_0 \subseteq W_a \subseteq \overline{W_b}$, $\phi_{f(a,b)}(f(a, b)) \neq 0$. So $f(a, b) \notin W_b$.

Consequently, $f(a, b) \notin W_a \cup W_b$, hence $K_0$ and $K_1$ are effectively recursively inseparable.

□