Math 114C: Computability Theory
Lecture Notes 1–10

Based on Course Materials

Contents

Lecture 1: Introduction

Decidability and Algorithms

Given two integers n, m, can we compute their greatest common divisor? Yes, the Euclidean algorithm solves this. Given a positive integer n, can we decide whether n is prime? Yes, by checking if any $k < n$ divides n.

Big Question: What precisely do we mean when we say we can "compute" a function or "decide" a problem? We mean we have an *algorithm* (an effective procedure).

Characteristics of an Algorithm

Some characteristics we look for in an algorithm:

1. Finite list of instructions.
2. Procedure is deterministic (what to do next is always clear).
3. No intuition required by the user (never vague).
4. May use "scratch work" (memory).
5. When it gives an answer, the procedure terminates after a finite number of steps.

In a modern context, we could summarize by saying: "Can be implemented by a computer". However, to prove that a function is *not* computable, we need a formal mathematical definition.

Hilbert's 10th Problem

Is there an algorithm which, given a Diophantine equation (multivariable polynomial equation with integer coefficients, e.g., $x^2 + y^2 = z^2$), determines whether or not the equation has an integer solution?

- **Answer:** No. The Diophantine integer solution problem is undecidable.

Mathematical Preliminaries

- The set of natural numbers is $\mathbb{N} := \{0, 1, 2, 3, \dots\}$.
- For a natural number $n > 0$, $\mathbb{N}^n := \{(x_0, \dots, x_{n-1}) : x_i \in \mathbb{N}\}$ is the set of n-tuples of $\mathbb{N}$. We index starting at 0.

Relations

Let $n > 0$ be an integer. An n-ary relation is a subset $R \subseteq \mathbb{N}^n$. Notation: For $\vec{x} = (x_0, \dots, x_{n-1}) \in \mathbb{N}^n$:

- $R(\vec{x}) \iff \vec{x} \in R$ ("R holds at $\vec{x}$").
- $\neg R(\vec{x}) \iff \vec{x} \notin R$.

Example 1. *Divisibility is a binary relation on* $\mathbb{N}$*:*

$$R(x, y) \iff x \text{ divides } y \iff (\exists k \in \mathbb{N})(y = k \cdot x)$$

So $R(2, 4)$ *holds, but* $\neg R(4, 2)$*.*

—

Lecture 2: Partial Functions

Definitions

Algorithms can hang on inputs (run forever). In computability theory, we deal with **partial functions**.

Definition 1. *Let X, Y be sets. A partial function f from X to Y is a function whose domain D is a subset $D \subseteq X$ (maybe $D = X$, maybe $D = \emptyset$) and whose codomain is Y.*

Notation:

- $f : X \rightharpoonup Y$ means f is a partial function.
- $f(x)\downarrow$: $f(x)$ converges (x is in the domain).
- $f(x)\uparrow$: $f(x)$ diverges (x is not in the domain).

We are concerned with n-ary partial functions on $\mathbb{N}$, $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$. If $f(\vec{x})\downarrow$ for every $\vec{x} \in \mathbb{N}^n$, f is a **total function**.

Definition Schemes

Three important schemes for defining partial functions: 1. Definition by Substitution. 2. Definition by Minimization. 3. Definition by Primitive Recursion.

1. Definition by Substitution

$f(\vec{x}) = h(g_0(\vec{x}), \ldots, g_{k-1}(\vec{x}))$. Here h is k-ary and g_i are n-ary. Specifically for partial functions:

$$f(\vec{x}) = y \iff (\exists y_0, \ldots, y_{k-1})[g_0(\vec{x}) = y_0 \wedge \cdots \wedge g_{k-1}(\vec{x}) = y_{k-1} \wedge h(y_0, \ldots, y_{k-1}) = y]$$

If any inner function diverges, the composition diverges.

2. Definition by Minimization

Let g be a partial function. The partial function f defined via minimization from g is written:

$$f(\vec{x}) = \mu y[g(\vec{x}, y) = 0]$$

This means $f(\vec{x}) = y$ iff: 1. $g(\vec{x}, y) = 0$. 2. For all $i < y$, $g(\vec{x}, i)\downarrow$ and $g(\vec{x}, i) \neq 0$.

Remark 1. *Computational intuition: We compute $g(\vec{x}, 0), g(\vec{x}, 1), \ldots$ one by one. If we find a 0, we output the index. If any computation $g(\vec{x}, i)$ hangs, we get stuck and never return an output.*

—

Lecture 3: Primitive Recursion

Examples and Definition

Example 2. *Factorial:* $0! = 1$, $(y+1)! = y! \cdot (y+1)$. *Addition:* $x + 0 = x$, $x + (y+1) = (x+y) + 1$.

Definition 2 (Primitive Recursion)**.** *Let $g : \mathbb{N}^n \rightharpoonup \mathbb{N}$ and $h : \mathbb{N}^{n+2} \rightharpoonup \mathbb{N}$ be partial. $f : \mathbb{N}^{n+1} \rightharpoonup \mathbb{N}$ is defined by primitive recursion from g and h if:*

$$f(\vec{x}, 0) = g(\vec{x})$$
$$f(\vec{x}, y+1) = h(\vec{x}, f(\vec{x}, y), y)$$

The domain implies that if $f(\vec{x}, y)\uparrow$, then $f(\vec{x}, y+1)\uparrow$.

Recursively Closed Classes

A class $\mathcal{C}$ of partial functions on $\mathbb{N}$ is **recursively closed** if it satisfies:

1. $\mathcal{C}$ contains the Successor ($S(x) = x + 1$), Zero, and Projection ($\Pi_i^n(\vec{x}) = x_i$) functions.
2. $\mathcal{C}$ is closed under substitution.
3. $\mathcal{C}$ is closed under primitive recursion.
4. $\mathcal{C}$ is closed under minimization.

Definition 3 (Recursive Function)**.** *A partial function f is **recursive** if it is a member of every recursively closed class of partial functions. [cite_start]The class of recursive functions R is the smallest recursively closed class [cite : 351 − 358].*

—

Lecture 4: More Recursive Partial Functions

Basic Recursive Functions

The following are recursive total functions:

1. Predecessor: $\text{Pred}(x) = x - 1$ if $x > 0$, else 0.
2. Addition: $+ : \mathbb{N}^2 \to \mathbb{N}$.
3. Multiplication: $\cdot : \mathbb{N}^2 \to \mathbb{N}$.
4. Exponentiation: x^y.

Proof idea: Defined via primitive recursion.

Advanced Arithmetic

- **Truncated Subtraction** ($x \dot{-} y$): equal to $x - y$ if $x \geq y$, else 0.

$$x \dot{-} 0 = x, \quad x \dot{-} (y + 1) = \text{Pred}(x \dot{-} y)$$

- **Division by 2**: $f(x) = \mu y[x \dot{-} 2y = 0]$. This is recursive.
- **Absolute Difference**: $|x - y| = (x \dot{-} y) + (y \dot{-} x)$.
- **Max/Min**: $\max(x, y) = \frac{x+y+|x-y|}{2}$ (requires division by 2 handling or logical equivalent).
- **Bounded Summation**: $g(\vec{x}, y) = \sum_{i=0}^{y} f(\vec{x}, i)$. If f is recursive, g is recursive via primitive recursion.

Recursive Relations

Definition 4. *An n-ary relation $R \subseteq \mathbb{N}^n$ is **recursive** if its characteristic function $Char_R$ is a recursive total function:*

$$Char_R(\vec{x}) = \begin{cases} 1 & \text{if } R(\vec{x}) \\ 0 & \text{if } \neg R(\vec{x}) \end{cases}$$

Examples: $x = 0$, $x < y$ are recursive relations.

—

Lecture 5: Properties of Recursive Relations

Closure Properties

Theorem 1. *Recursive relations are closed under recursive total substitution. If R is recursive and f_i are recursive total functions, then $Q(\vec{x}) \iff R(f_0(\vec{x}), \dots)$ is recursive.*

Note: Not necessarily closed under partial substitution yet.

Theorem 2. *Let R, Q be recursive. Then $\neg R$, $R \wedge Q$, $R \vee Q$ are recursive. Proof: Use arithmetic on characteristic functions, e.g., $Char_{\neg R} = 1 \dot{-} Char_R$.*

Theorem 3 (Bounded Quantification)**.** *If $R(\vec{x}, y)$ is recursive, then so are:*

- $P(\vec{x}, y) \iff (\forall i \leq y) R(\vec{x}, i)$
- $Q(\vec{x}, y) \iff (\exists i \leq y) R(\vec{x}, i)$.

Definition by Cases

If R is recursive and f, g are recursive total functions, then $h(\vec{x}) = f(\vec{x})$ if $R(\vec{x})$ else $g(\vec{x})$ is recursive.

Graphs of Partial Functions

The graph of f is $G_f(\vec{x}, y) \iff f(\vec{x}) = y$.

Theorem 4. *If f is a total function, f is recursive iff G_f is recursive.*

—

Lecture 6: Sequence Coding

We need to code finite sequences of numbers $(x_0, \dots, x_{n-1})$ into single natural numbers $\mathbb{N}$. Let $p_0, p_1, \dots$ be the prime numbers $(2, 3, 5, \dots)$. The function $n \mapsto p_n$ is recursive.

The Coding

Definition 5. *The code of the sequence $(x_0, \dots, x_{n-1})$ is:*

$$\langle x_0, \dots, x_{n-1} \rangle = p_0^{x_0+1} \cdot p_1^{x_1+1} \cdots p_{n-1}^{x_{n-1}+1}$$

The empty sequence is coded as $\langle \emptyset \rangle = 1$.

Example: $\langle 2, 1, 0 \rangle = 2^{2+1} \cdot 3^{1+1} \cdot 5^{0+1} = 2^3 \cdot 3^2 \cdot 5^1 = 360$.

Sequence Predicate

The relation $Seq(u) \iff u$ is a sequence code, is recursive. $Seq(u) \iff (\forall x \leq u)(\forall y \leq u)[(Prime(x) \wedge Prime(y) \wedge x < y \wedge y \mid u) \implies x \mid u]$.

—

Lecture 7: Sequence Operations

Recursive Functions on Sequences

The following are recursive total functions:

1. **Length** $lh(u)$: The length of the sequence coded by u.
2. **Decoding** $(u)_i$: The i-th coordinate of the sequence. If $i \geq lh(u)$, return 0.
3. **Truncation** $u \restriction i$: Returns code of first i elements.
4. **Concatenation** $u * v$: Returns code of sequence u followed by v.

Course-of-Values Recursion

Idea: To compute $f(\vec{x}, y+1)$, use all previous values $f(\vec{x}, 0), \ldots, f(\vec{x}, y)$.

Theorem 5. *If f is defined by $f(\vec{x}, y+1) = h(\vec{x}, \langle f(\vec{x}, 0), \ldots, f(\vec{x}, y)\rangle, y)$, then f is recursive.*

Example: Fibonacci sequence is recursive.

—

Lecture 8: The Unlimited Register Machine (URM)

Components

- **Registers:** $R_0, R_1, R_2, \ldots$ containing natural numbers $r_0, r_1, \ldots$.
- **Program:** A finite list of instructions $I_0, I_1, \ldots, I_k$.

Instructions

1. **Zero** $Z(n)$: Replace r_n with 0.
2. **Successor** $S(n)$: Replace r_n with $r_n + 1$.
3. **Transfer** $T(m, n)$: Replace r_n with r_m.
4. **Jump** $J(m, n, q)$: If $r_m = r_n$, go to instruction I_q; else go to next instruction.

Computability

A program **halts** if it tries to execute an instruction that does not exist. A function $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ is **URM-computable** if there is a program P such that for input $\vec{x}$ in registers $R_1, \ldots, R_n$ (and 0 elsewhere), the machine halts with $f(\vec{x})$ in R_0.

Theorem 6. $\mathcal{R} = \mathcal{U}$. *The class of recursive functions is exactly the class of URM-computable functions.*

Direction $\mathcal{R} \subseteq \mathcal{U}$: Show URM can simulate basic recursive functions and closure properties (tedious programming).

—

Lecture 9: Arithmetization of URM

To show $\mathcal{U} \subseteq \mathcal{R}$, we "arithmetize" the machine. We assign numbers to instructions, programs, and machine states.

Coding Instructions

- $\#Z(n) = \langle 0, n \rangle$
- $\#S(n) = \langle 1, n \rangle$
- $\#T(m, n) = \langle 2, m, n \rangle$
- $\#J(m, n, q) = \langle 3, m, n, q \rangle$.

A program P is coded as $\#P = \langle \#I_0, \ldots, \#I_{k-1} \rangle$.

Kleene T-Predicate

We define a predicate $T_n(e, \vec{x}, y)$ meaning: "e is a program code, $\vec{x}$ is input, and y codes a halting computation of program e on $\vec{x}$". The computation code y is a sequence of codes of machine states.

Theorem 7. *For any $n > 0$, $T_n(e, \vec{x}, y)$ is a recursive relation.*

Result

Any URM-computable function g can be written as:

$$g(\vec{x}) = U(\mu y[T_n(e, \vec{x}, y)])$$

Where $U(y)$ extracts the output from the final state in the computation sequence y. Since T_n is recursive and minimization is recursive, g is recursive. Thus $\mathcal{U} \subseteq \mathcal{R}$.

—

Lecture 10: Effective Enumerations

Universal Function

Based on the Kleene T-predicate, we can enumerate all computable functions. Let $\phi_e^{(n)}(\vec{x}) = U(\mu y[T_n(e, \vec{x}, y)])$. The sequence $\phi_0^{(n)}, \phi_1^{(n)}, \ldots$ contains all n-ary computable partial functions. The universal function $\Psi_U^{(n)}(e, \vec{x}) = \phi_e^{(n)}(\vec{x})$ is computable.

S-m-n Theorem (Parameter Theorem)

Let $f(x, y)$ be a computable function. There exists a computable total function $S(x)$ such that:

$$\phi_{S(x)}(y) = f(x, y)$$

Generally, for $f : \mathbb{N}^m \times \mathbb{N}^n \rightharpoonup \mathbb{N}$, there is a total computable $S : \mathbb{N}^m \to \mathbb{N}$ such that $\phi_{S(\vec{x})}^{(n)}(\vec{y}) = f(\vec{x}, \vec{y})$.

Interpretation: We can treat the first m arguments as "parameters" and effectively find a program code that hard-codes these parameters.