Computability Theory Summary

Mathematical Notes

October 27, 2025

Contents

1 Introduction

Computability theory studies what can and cannot be computed algorithmically. It provides the mathematical foundation for understanding the limits of computation and establishes fundamental concepts that underlie computer science.

2 Computational Models

2.1 Turing Machines

Definition 2.1. A **Turing machine** consists of:

- A finite set of states Q
- A finite alphabet Σ (input symbols)
- A tape alphabet $\Gamma \supseteq \Sigma$ (includes blank symbol B)
- A transition function $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
- A start state $q_0 \in Q$
- Accept and reject states $q_{\text{accept}}, q_{\text{reject}} \in Q$

Definition 2.2. A language L is **Turing-recognizable** (recursively enumerable) if there exists a Turing machine that accepts all strings in L and either rejects or loops on strings not in L.

Definition 2.3. A language L is **Turing-decidable** (recursive) if there exists a Turing machine that accepts all strings in L and rejects all strings not in L.

2.2 Lambda Calculus

Definition 2.4. The **lambda calculus** consists of:

- Variables: $x, y, z, \ldots$
- Lambda abstractions: $\lambda x.M$
- Applications: MN

where M, N are lambda terms.

Definition 2.5. The **beta reduction** rule is:

$$(\lambda x.M)N \to_\beta M[x := N]$$

where $M[x := N]$ denotes substitution of N for x in M.

Definition 2.6. A lambda term is in **normal form** if no beta reduction can be applied to it.

2.3 Church Encoding

Definition 2.7. Natural numbers in lambda calculus:

$$0 = \lambda f.\lambda x.x \tag{1}$$
$$1 = \lambda f.\lambda x.fx \tag{2}$$
$$2 = \lambda f.\lambda x.f(fx) \tag{3}$$
$$n = \lambda f.\lambda x.f^n x \tag{4}$$

Definition 2.8. The **successor function**:

$$\text{succ} = \lambda n.\lambda f.\lambda x.f(nfx)$$

2.4 Recursive Functions

Definition 2.9. The **primitive recursive functions** are defined inductively:

- **Zero function**: $Z(x) = 0$
- **Successor function**: $S(x) = x + 1$
- **Projection functions**: $P_i^n(x_1, \ldots, x_n) = x_i$
- **Composition**: If $g, h_1, \ldots, h_m$ are primitive recursive, then so is $f(x_1, \ldots, x_n) = g(h_1(x_1, \ldots, x_n), \ldots, h_m(x_1,$
- **Primitive recursion**: If g and h are primitive recursive, then so is f defined by:

$$f(0, x_2, \ldots, x_n) = g(x_2, \ldots, x_n) \tag{5}$$
$$f(y+1, x_2, \ldots, x_n) = h(y, f(y, x_2, \ldots, x_n), x_2, \ldots, x_n) \tag{6}$$

Definition 2.10. The **general recursive functions** (partial recursive functions) extend primitive recursive functions with the **minimization operator**:

$$\mu y[g(x_1, \ldots, x_n, y) = 0] = \text{the least } y \text{ such that } g(x_1, \ldots, x_n, y) = 0$$

3 Church-Turing Thesis

Theorem 3.1 (Church-Turing Thesis)**.** The class of computable functions is exactly the class of functions computable by:

- Turing machines
- Lambda calculus
- General recursive functions
- Any other reasonable model of computation

4 Decidability

4.1 Decidable Languages

Theorem 4.1. The following languages are decidable:

- $A_{DFA} = \{\langle B, w\rangle : B \text{ is a DFA that accepts } w\}$
- $A_{NFA} = \{\langle B, w\rangle : B \text{ is an NFA that accepts } w\}$
- $A_{REX} = \{\langle R, w\rangle : R \text{ is a regular expression that generates } w\}$
- $E_{DFA} = \{\langle A\rangle : A \text{ is a DFA and } L(A) = \emptyset\}$
- $EQ_{DFA} = \{\langle A, B\rangle : A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$

4.2 Undecidable Languages

Theorem 4.2 (Halting Problem)**.** The language $A_{TM} = \{\langle M, w\rangle : M \text{ is a TM that accepts } w\}$ is undecidable.

Proof. Assume A_{TM} is decidable. Let H be a decider for A_{TM}. Construct TM D:

1. On input $\langle M\rangle$:
2. Run H on $\langle M, \langle M\rangle\rangle$
3. If H accepts, reject; if H rejects, accept

Then D on input $\langle D\rangle$ accepts if and only if D rejects $\langle D\rangle$, which is a contradiction. □

Theorem 4.3. The following languages are undecidable:

- $E_{TM} = \{\langle M\rangle : M \text{ is a TM and } L(M) = \emptyset\}$
- $EQ_{TM} = \{\langle M_1, M_2\rangle : M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$
- $REGULAR_{TM} = \{\langle M\rangle : M \text{ is a TM and } L(M) \text{ is regular}\}$

5 Reducibility

5.1 Mapping Reducibility

Definition 5.1. Language A is **mapping reducible** to language B (written $A \leq_m B$) if there exists a computable function $f : \Sigma^* \to \Sigma^*$ such that for every w:

$$w \in A \text{ if and only if } f(w) \in B$$

Theorem 5.1. If $A \leq_m B$ and B is decidable, then A is decidable.

Theorem 5.2. If $A \leq_m B$ and A is undecidable, then B is undecidable.

5.2 Rice's Theorem

Theorem 5.3 (Rice's Theorem)**.** Let P be a non-trivial property of Turing-recognizable languages. Then the language $\{\langle M\rangle : L(M) \text{ has property } P\}$ is undecidable.

6 Complexity Theory Basics

6.1 Time Complexity

Definition 6.1. The **time complexity** of a Turing machine M is the function $f : \mathbb{N} \to \mathbb{N}$ where $f(n)$ is the maximum number of steps that M uses on any input of length n.

Definition 6.2. **TIME**(t(n)) is the class of languages decided by $O(t(n))$ time Turing machines.

Definition 6.3.

$$\mathrm{P} = \bigcup_k \mathrm{TIME}(n^k)$$

is the class of languages decidable in polynomial time.

6.2 Space Complexity

Definition 6.4. The **space complexity** of a Turing machine M is the function $f : \mathbb{N} \to \mathbb{N}$ where $f(n)$ is the maximum number of tape cells that M scans on any input of length n.

Definition 6.5.

$$\mathrm{PSPACE} = \bigcup_k \mathrm{SPACE}(n^k)$$

is the class of languages decidable in polynomial space.

6.3 NP and NP-Completeness

Definition 6.6. A language L is in **NP** if there exists a polynomial-time verifier V such that:

$$L = \{w : \text{there exists } c \text{ such that } V \text{ accepts } \langle w, c \rangle\}$$

Definition 6.7. A language B is **NP-complete** if:

- $B \in \mathrm{NP}$
- For every $A \in \mathrm{NP}$, $A \leq_p B$ (polynomial-time reducible)

Theorem 6.1 (Cook-Levin Theorem)**.** SAT is NP-complete.

7 Lambda Calculus Theory

7.1 Simply Typed Lambda Calculus

Definition 7.1. Types in simply typed lambda calculus:

- Base types: $A, B, C, \ldots$
- Function types: $A \to B$

Definition 7.2. Typing rules:

- **Variable**: $\Gamma, x : A \vdash x : A$
- **Abstraction**: $\frac{\Gamma, x{:}A \vdash M{:}B}{\Gamma \vdash \lambda x.M{:}A \to B}$
- **Application**: $\frac{\Gamma \vdash M{:}A \to B \quad \Gamma \vdash N{:}A}{\Gamma \vdash MN{:}B}$

Theorem 7.1 (Strong Normalization)**.** Every well-typed term in simply typed lambda calculus has a normal form.

7.2 System F (Polymorphic Lambda Calculus)

Definition 7.3. Types in System F:

- Type variables: $\alpha, \beta, \gamma, \ldots$
- Universal quantification: $\forall \alpha.A$
- Function types: $A \to B$

Definition 7.4. Additional typing rules for System F:

- **Type abstraction**: $\frac{\Gamma \vdash M : A}{\Gamma \vdash \Lambda \alpha.M : \forall \alpha.A}$
- **Type application**: $\frac{\Gamma \vdash M : \forall \alpha.A}{\Gamma \vdash MB : A[\alpha := B]}$

7.3 Dependent Types

Definition 7.5. In **dependent type theory**, types can depend on values:

- Dependent function types: $\Pi_{x:A} B(x)$
- Dependent pair types: $\Sigma_{x:A} B(x)$

8 Curry-Howard Correspondence

Theorem 8.1 (Curry-Howard Correspondence)**.** There is a correspondence between:

- **Types** in lambda calculus and **propositions** in logic
- **Terms** in lambda calculus and **proofs** in logic
- **Reduction** in lambda calculus and **proof normalization**

Example 8.1.

- $A \to B$ corresponds to implication $A \Rightarrow B$
- $A \times B$ corresponds to conjunction $A \wedge B$
- $A + B$ corresponds to disjunction $A \vee B$
- $\forall \alpha.A$ corresponds to universal quantification $\forall x.A$

9 Computational Equivalence

9.1 Equivalence of Models

Theorem 9.1. The following computational models are equivalent:

- Turing machines
- Lambda calculus
- General recursive functions
- Register machines
- Cellular automata

9.2 Universal Computation

Definition 9.1. A computational model is **Turing-complete** if it can simulate any Turing machine.

Example 9.1. Turing-complete systems include:

- Lambda calculus
- Cellular automata (Rule 110)
- Conway's Game of Life
- Most programming languages

10 Algorithmic Information Theory

10.1 Kolmogorov Complexity

Definition 10.1. The **Kolmogorov complexity** of a string s is:

$$K(s) = \min\{|p| : U(p) = s\}$$

where U is a universal Turing machine and p is a program.

Theorem 10.1. For any computable function f, there exists a constant c such that:

$$K(f(s)) \leq K(s) + c$$

10.2 Incompressibility

Definition 10.2. A string s is **incompressible** if $K(s) \geq |s|$.

Theorem 10.2. Most strings are incompressible.

11 Applications

11.1 Programming Language Design

- Type systems based on lambda calculus
- Functional programming languages
- Proof assistants and theorem provers
- Compiler design and optimization

11.2 Logic and Proof Theory

- Constructive mathematics
- Intuitionistic logic
- Automated theorem proving
- Formal verification

11.3 Computer Science Theory

- Complexity theory
- Algorithm design
- Cryptography
- Distributed systems

12 Important Theorems

12.1 Recursion Theorem

Theorem 12.1 (Recursion Theorem)**.** For any computable function f, there exists a Turing machine M such that M and $f(\langle M \rangle)$ compute the same function.

12.2 Fixed Point Theorem

Theorem 12.2 (Fixed Point Theorem in Lambda Calculus)**.** For any lambda term F, there exists a term X such that $FX =_\beta X$.

12.3 Church-Rosser Theorem

Theorem 12.3 (Church-Rosser Theorem)**.** If $M \to^* N_1$ and $M \to^* N_2$, then there exists N_3 such that $N_1 \to^* N_3$ and $N_2 \to^* N_3$.

13 Open Problems

13.1 P vs NP

Conjecture 13.1 (P vs NP Problem)**.** Is P = NP?

13.2 Church-Turing-Deutsch Principle

Conjecture 13.2. Every finitely realizable physical system can be perfectly simulated by a universal model computing machine operating by finite means.

14 Conclusion

Computability theory provides the mathematical foundation for understanding computation. Key insights include:

- The equivalence of different computational models (Church-Turing thesis)
- The existence of undecidable problems (halting problem)
- The connection between computation and logic (Curry-Howard correspondence)
- The fundamental limits of algorithmic computation

These concepts are essential for computer science, logic, and the philosophy of computation, providing deep insights into what can and cannot be computed.