Advanced computability theory

Textbook: Chapter 6

Quines

- A **Quine** is a self-producing program
- A Quine takes no input (files, command-line, cin, etc) and outputs only its own source code

In C:

```
/* (formatted to fit on slide) */
char*s="char*s=%c%s%c;main(){printf(s,34,s,34);}";
main(){printf(s,34,s,34);}
```

In English:

Print out this sentence.

Can we build a TM to do this?

Kleene's Recursion Theorem

- By the same Kleene from section 1
- Allows any TM M to access its own description $\langle M \rangle$

The idea:

> *"Print the previous with quotes and then without"*
> *Print the previous with quotes and then without*

Kleene Recursion pt. 2

Lemma: Computable printing. There exists a computable function $q : \Sigma^* \to \Sigma^*$ where, for any string w, $q(w)$ is the description of a Turing machine P_w that prints out w and then halts.

Pf: The following TM Q computes $q(w)$.

$Q =$ "On input string w:

1. Construct the following TM P_w: $P_w =$ "On any input:
   1.1 Erase input
   1.2 Write w on the tape
   1.3 Halt."
2. Output $\langle P_w \rangle$."

The TM *SELF*

- *SELF* will be a Quine TM
- We will make it 2 parts: A and B
  - $\langle SELF \rangle = \langle AB \rangle$
- A will print out $\langle B \rangle$ and B will print out $\langle A \rangle$
- A is easy! Just use $q(\langle B \rangle)$
  - **A leaves $\langle B \rangle$ on the tape**
- **How do we create B?**
  - We can't use q, or we would get a circulate definition!
- B can look at the contents of the tape as its input
- We already know $\langle A \rangle = q(\langle B \rangle)$, so we can compute $\langle A \rangle$ given only $\langle B \rangle$

SELF pt. 2

$B =$ "On input $\langle M \rangle$:

1. Compute $q(\langle M \rangle)$ (the description of a TM that prints out the description of M)
2. Erase the tape
3. Write $q(\langle M \rangle)\,\langle M \rangle$"

$\langle SELF \rangle = q(\langle B \rangle)\,\langle B \rangle$

Running *SELF*

- What happens if we run *SELF*?

1. Start with some input tape
2. Start running the TM $q(\langle B \rangle)$
   2.1 Erases tape
   2.2 Writes $\langle B \rangle$
   2.3 Halts
3. Start running the TM B on the contents of the tape
   3.1 The input $\langle M \rangle$ is $\langle B \rangle$!
   3.2 Computes $q(\langle B \rangle)$
   3.3 Erases the tape
   3.4 Writes $q(\langle B \rangle)\,\langle B \rangle$
   3.5 Halts
4. Halts

- The contents of the tape are now $q(\langle B \rangle)\,\langle B \rangle = \langle SELF \rangle$
- **Self takes any input and prints its own description**

Recursion theorem

Def: Kleene's recursion theorem. Let T be a TM that computes a function $t : \Sigma^* \times \Sigma^* \to \Sigma^*$. There is a TM R that computes some $r : \Sigma^* \to \Sigma^*$ where, for every w,

$$r(w) = t\left(\langle R\rangle, w\right)$$

- **A TM can always obtain its own description!**

Recursion theorem pf

- Let $\langle R\rangle = \langle ABT\rangle$

$A =$ "On input w:

1. Compute $q(\langle BT\rangle)$
2. Erase tape
3. Write $q(\langle BT\rangle)w$"

$B =$ "On input $q(\langle MH\rangle)w$:

1. Create the TM N:

   $N =$ "On input w:

   1.1 Compute $q(\langle MH\rangle)$
   1.2 Erase tape
   1.3 Write $q(\langle MH\rangle)w$"

2. Let $\langle R\rangle = \langle NMH\rangle$

3. Simulate H on input $\langle R, w\rangle$"

The Minimality problem

Def: For a TM M, M is **minimal** if there exist no TMs equivalent to M with a smaller description. Let

$$MIN_{TM} = \{M : M \texttt{ is a minimal TM}\}$$

Thm: MIN_{TM} is not Turing-recognizable.

The Minimality problem pt. 2

Pf: We will assume some enumerator E for MIN_{TM} exists and obtain a contradiction.

$C =$ "On input w:

1. Obtain, via the recursion theorem, own description $\langle C \rangle$
2. Run E until it yields a TM D such that $|\langle C \rangle| < |\langle D \rangle|$. Since MIN_{TM} is infinite, this is guaranteed to happen
3. Simulate D on input w."

All items yielded by E are definitionally in MIN_{TM}. However, C simulates D and is thus equivalent to it. We know C is shorter than D, but D is minimal: **Contradiction**.

Decidability of logical theories

Is logic decidable? Given some statement, can we every know whether or not it is true?

Three logic problems of increasing difficulty:

1. $\forall q \exists p \forall x, y\, [p > q \wedge (x, y > 1 \rightarrow xy \neq p)]$
   - (Infinitely many primes exist: Proven by Euclid)
2. $\forall a, b, c, n\, [(a, b, c > 0 \wedge n > 2) \rightarrow a^n + b^n \neq c^n]$
   - (Fermat's last theorem: Only recently proven)
3. $\forall q \exists p \forall x, y\, [p > q \wedge (x, y > 1 \rightarrow (xy \neq p \wedge xy \neq p + 2))]$
   - (Twin prime conjecture: Unproven)

Formal definition of logic

Let the alphabet of logic be:

$$\{\wedge, \vee, \neg, (,), \forall, x, \exists, R_1, \cdots, R_k\}$$

- $\wedge, \vee$, and $\neg$ are **Boolean operations**
- "(" and ")" are the **parenthesis**
- $\forall$ and $\exists$ are the **quantifiers**
- x represents the infinite sets of **variables**
- $R_1, \cdots, R_k$ denote **relations**

Def: A string ϕ is a **formula** if:

1. ϕ has the form $R_i(x_1, \ldots, x_j)$ **or**
2. ϕ has the form $\phi_1 \wedge \phi_2$ or $\phi_1 \vee \phi_2$ or $\neg\phi_1$, where ϕ_1 and ϕ_2 are formulas **or**
3. ϕ has the form $\exists x_i(\phi_1)$ or $\forall x_i(\phi_1)$, where ϕ_1 is a formula

Mathematical models and theories

- A formula is in **prenex normal form** iff all quantifiers occur at the beginning
- An unquantified variable is called a **free variable**
- A formula with no free variables is called a **sentence** or **statement**
- The **universe** is the set of values that variables may take
- A **model** $\mathcal{M}$ is a tuple $(U, P_1, \ldots, P_k)$, where U is the universe and $P_1, \ldots, P_k$ are the relations assigned to symbols R_1 through R_k
- The **theory of** $\mathcal{M}$, written $Th(\mathcal{M})$, is the set of all true sentences on model $\mathcal{M}$

Ex: $(\mathcal{N}, +, \times)$ is a model where variables can take any value $\in \mathcal{N}$ and the relations $+$ and $\times$ can be used. Note: $+$ corresponds to a relation $R_+(x_1, x_2, x_3) \iff (x_1 + x_2 = x_3)$, with a similar relation for $\times$.

Proving Gödel's incompleteness theorem

Def: Formal proof. The **formal proof** π of a statement ϕ uis a sequence of statements $S_1, S_2, \ldots, S_l$ where $S_l = \phi$. Each statement must follow simply and directly from the previous statements.

Assume the following are true:

1. The correctness of a proof of a statement can be checked by a machine. Formally, $\{\langle \phi, \pi \rangle : \pi \texttt{ is a proof of } \phi\}$ is decidable
   - Given a proof, we can check if it implies a given statement
2. If a statement is provable, it is true

Gödel pt. 2

Thm 1: $Th(\mathcal{N}, +, \times)$ is Turing-recognizable.

Pf: Use the implied proof-checker from property 1 to enumerate the set of all proofs. If a given proof proves the statement in question, accept. This is a recognizer, but not a decider. End of proof.

- Corollary: There is an algorithm to decide if a given statement is provable. Let this algorithm be called P

Lemma 1.1: A_{TM} is reducible to $Th(\mathcal{N}, +, \times)$.

Pf: You can use $+$ and $\times$ to extract and encode symbols in very large integers such that they simulate the evolution of computation histories. Therefore, $Th(\mathcal{N}, +, \times)$ is Turing-complete and therefore undecidable.

- Consider encoding a TM in binary, then representing binary as a series of additions and multiplications

Gödel pt. 3

Thm 2: Some (true) statement in $Th(\mathcal{N}, +, \times)$ is not provable.

Pf: We will assume all true statements are provable and derive an algorithm to decide $Th(\mathcal{N}, +, \times)$, a contradiction of lemma 1.1.

Take in some statement ϕ. Simulate P on both ϕ and $\neg\phi$. Since a proof exists for ϕ, one of these two instances will halt. Therefore, our system decides the truth value of ϕ (A_{TM}). Contradiction! End of proof.

Gödel pt. 4

Thm 3: The sentence $\psi_{\texttt{unprovable}} =$ "this statement has no proof **within the system**" is true.

Pf: Let $\phi_{M,w}$ be the statement "TM M accepts input w". This is implied to exist for any TM M and input w by lemma 1.1.

We will construct the statement "This statement is not provable **within the proof system**" using the recursion theorem.

$S =$ "On any input **(including** 0**)**:

1. Obtain own description $\langle S \rangle$ via the recursion theorem
2. Construct the sentence $\psi_{\texttt{unprovable}} = \neg\exists c[\phi_{S,0}]$ using lemma 1.1. ('this TM never accepts 0')
   - If $\psi_{\texttt{unprovable}}$ is true, this TM never accepts 0
3. Run P (the theorem prover) on $\psi_{\texttt{unprovable}}$
4. If P accepts (there is a proof for $\psi_{\texttt{unprovable}}$), **accept**. If P halts and rejects (there is no proof for $\psi_{\texttt{unprovable}}$), **reject.**"

Gödel pt. 5

$$S = \begin{cases} \texttt{always accept if there is a proof for } \psi_{\texttt{unprovable}} \\ \texttt{reject or loop if there is no proof for } \psi_{\texttt{unprovable}} \end{cases}$$

$$= \begin{cases} \texttt{accept if proof exists that } S \texttt{ never accepts } 0 \\ \texttt{reject or loop otherwise} \end{cases}$$

Run S on input 0

Gödel pt. 6

S must accept, reject, or loop forever on 0

- If S accepts 0, proof exists that S never accepts 0
  - Since all proven statements are true, this can't happen!
  - Therefore, **this can never be the case**
  - S must reject or loop on 0
- "S rejects or loops on 0" is the same as $\psi_{\texttt{unprovable}}$
  - Only happens if no proof exists
  - "This statement has no proof within the proof system"
- $\psi_{\texttt{unprovable}}$ being false causes a contradiction
  - **Therefore, $\psi_{\texttt{unprovable}}$ must be true**
- We have used a **more powerful proof system** to prove that a **less powerful proof system** (Peano arithmetic) contains unprovable truths

Turing reducibility pt. 1

- Is mapping reducibility the best explanation? **No!**
- Consider: Could we prove that $\overline{A_{TM}}$ is undecidable by mapping reducibility?
  - Remember, $\overline{A_{TM}}$ is not Turing-recognizable
  - Therefore, we can't even identify instances of it, let along map them!
- However, $\overline{A_{TM}}$ is intuitively reducible to A_{TM} and vice versa since a solution to one solves the other
- **Solution:** Oracle Turing Machines. An Oracle TM has a "magic oracle" decider that it can call without cost at any time
  - Assume the "oracle" is magic: It can decide even undecidable and unrecognizable languages

Turing reducibility pt. 2

Def: A **TM** M with access to a decider for **language** B is written M^B.

- Ex: A TM T with access to a decider for A_{TM} would be written $T^{A_{TM}}$

Def: If M^B decides A, we say A is **decidable relative** to B. - Given a decider for B, we can decide A

- **Much more intuitive than mapping reducibility**

Def: If A is decidable relative to B, then A is **Turing reducible** to B, written $A \leq_T B$.

Ex: $E_{TM} \leq_T A_{TM}$

Thm: $E_{TM} \leq_T A_{TM}$ ("given a decider for A_{TM}, we can decide E_{TM}")

Pf: By construction. Let `accepts(M, w)` be the oracle procedure deciding A_{TM}.

```
def is_empty(M):          # Decides emptiness problem
  def T(_):               # Define a helper TM T
    for every string s:   # Enumerate every possible string
      if accepts(M, s):   # If M accepts this random string
        accept
  if accepts(T, ''):      # T is empty iff not M(s)
    reject
  else:
    accept
```

This decides E_{TM}. End of proof.

Ex: $A_{TM} \leq_T E_{TM}$

Thm: $A_{TM} \leq_T E_{TM}$ ("given a decider for E_{TM}, we can decide A_{TM}")

Pf: By construction. Let `is_empty(M)` be an **oracle** (not the fn from the previous: That would be circular).

```
def accepts(M, w): # Decides acceptance problem
  def T(x):        # Helper TM
    if x == w:
      if M(w):     # Simulates M on w
        accept
    reject         # Rejects all non-w
  if is_empty(T):  # Is empty iff M rejects w
    reject
  else:
    accept         # Nonempty iff M accepts w
```

This decides A_{TM}. End of proof.

The *other* meaning of Turing-Equivalence

- $E_{TM} \leq_T A_{TM}$ **and** $A_{TM} \leq_T E_{TM}$
- In this case we say that E_{TM} and A_{TM} are **Turing-equivalent**
- Written $E_{TM} \equiv_T A_{TM}$

This is an **equivalence relation**

Not to be confused with the Turing-equivalence of a system S, where a TM can simulate S and vice versa.

A definition of information

- There is no singular definition of information
- *One* definition is the size of the minimal representation

$$A = 10101010101010101010$$
$$B = 10111001011111101000$$

- A is just 01 10 times
- B has "more information" than A, since it doesn't appear to follow a pattern

Def: Minimal descriptions. Let x be a binary string. The **minimal description** of x, written $d(x)$, is the shortest string $\langle M, w \rangle$ where TM M on input w halts with x on its tape. The **descriptive complexity** of x, written $K(x)$, is

$$K(x) = |d(x)|$$

Note: x's descriptive complexity **is not** its length! It can be **longer!**

Compression

- $K(x)$ might be longer than x
- If it's shorter, we can treat it as the compressed version, running M on w to uncompress it

Def: Let x be a string. x is said to be c-**compressible** for some natural number c if

$$K(x) \leq |x| - c$$

That is, if $|\langle M, w \rangle| \leq |x| - c$ for minimal $\langle M, w \rangle$.

If x is not compressible by 1, we say it is **incompressible**.

Incompressible strings

Thm: There are incompressible strings of every length.

Pf: If a string x is compressible, there exists a description $d(x)$ such that $|d(x)| < |x|$. Let n be any arbitrary integer.

There are 2^n binary string of length n: However, there are only $\sum_{i=1}^{n-1} 2^i = 2^n - 1$ descriptions of length $< n$. Therefore, there must exist at least one incompressible string of length n. End of proof.

What do incompressible strings look like?

- It can be shown that incompressible strings look like series of random coin tosses
- K is **incomputable**, so no examples exist
- If we had an example, we couldn't prove it was incompressable

Next up: Intro to complexity and asymptotic analysis

End of part 2 out of 3!

Notation	Common name	Limit test (note limit may not exist)
$f(n) \in O(g(n))$	Asymptotic upper bound	$\lim_{x \to \infty} \left\lvert \frac{f(x)}{g(x)} \right\rvert < \infty$
$f(n) \in o(g(n))$	Asymptotically negligible	$\lim_{x \to \infty} \left\lvert \frac{f(x)}{g(x)} \right\rvert = 0$
$f(n) \in \Omega(g(n))$	Asymptotic lower bound	$\lim_{x \to \infty} \left\lvert \frac{f(x)}{g(x)} \right\rvert > 0$
$f(n) \in \omega(g(n))$	Asymptotically dominant	$\lim_{x \to \infty} \left\lvert \frac{f(x)}{g(x)} \right\rvert = \infty$
$f(n) \in \Theta(g(n))$	Asymptotically tight bound	$0 < \lim_{x \to \infty} \left\lvert \frac{f(x)}{g(x)} \right\rvert < \infty$