Church-Turing thesis, deciable and undecidable languages

204213 Theory of Computation

Jittat Fakcharoenphol

Kasetsart University

January 14, 2009

Outline

What's so good about TM?

What's so good about TM?

- Easy describe and understand.

What's so good about TM?

- Easy describe and understand.
- So hard to extend!

Failed attempts to make TM more powerful

- Add “stay put”,

Failed attempts to make TM more powerful

- Add "stay put",
- Add more tapes,

Failed attempts to make TM more powerful

- Add “stay put”,
- Add more tapes,
- Add nondeterminism,

Failed attempts to make TM more powerful

- Add "stay put",
- Add more tapes,
- Add nondeterminism,
- Add doubly infinite tape.

Enumerators

- An **enumerator** is a TM with a printer.
- It can print strings to the printer.
- A language **enumerated** by an enumerator E is a set of strings printed by E.

Equivalence to TM recognizable languages

Theorem 1

A language is Turing-recognizable iff some enumerator enumerates it.

Proof

Again, there're 2 directions:

Proof

Again, there're 2 directions:

- If some enumerator E enumerates language A, there exists a TM M that recognizes A,

Proof

Again, there're 2 directions:

- If some enumerator E enumerates language A, there exists a TM M that recognizes A,

$$A \text{ enumeratable} \Rightarrow A \text{ Turing-recognizable}$$

and

Proof

Again, there're 2 directions:

- If some enumerator E enumerates language A, there exists a TM M that recognizes A,

  $$A \text{ enumeratable} \Rightarrow A \text{ Turing-recognizable}$$

  and

- If some TM M recognizes a language A, there exists an enumerator E that enumerates A.

Proof

Again, there're 2 directions:

- If some enumerator E enumerates language A, there exists a TM M that recognizes A,

$$A \text{ enumeratable} \Rightarrow A \text{ Turing-recognizable}$$

and

- If some TM M recognizes a language A, there exists an enumerator E that enumerates A.

$$A \text{ enumeratable} \Leftarrow A \text{ Turing-recognizable}$$

Proof ($\Rightarrow$)

M = "On input w,

- M simulates E,

Proof ($\Rightarrow$)

M = "On input w,

- M simulates E,
- When E prints any string, compare with w,

Proof ($\Rightarrow$)

M = "On input w,

- M simulates E,
- When E prints any string, compare with w,
- If they're the same, M accepts w."

Wrong proof ($\Leftarrow$)

E = "Ignore the input,

1. E tries every possible inputs,

Wrong proof ($\Leftarrow$)

E = "Ignore the input,

1. E tries every possible inputs,
2. For each input w, E runs M, and prints w if M accepts it."

Wrong proof ($\Leftarrow$)

E = "Ignore the input,

1. E tries every possible inputs,
2. For each input w, E runs M, and prints w if M accepts it."

What's wrong with this proof?

Wrong proof ($\Leftarrow$)

E = "Ignore the input,

1. E tries every possible inputs,
2. For each input w, E runs M, and prints w if M accepts it."

What's wrong with this proof? if M loops on some input, E'll never print any string after that.

Proof ($\Leftarrow$)

E = "Ignore the input,

1. Repeat the steps below for $i = 1, 2, \ldots$:

Proof ($\Leftarrow$)

E = "Ignore the input,

1. Repeat the steps below for $i = 1, 2, \ldots$:
   - E tries every possible inputs w of length at most i,

Proof ($\Leftarrow$)

E = "Ignore the input,

1. Repeat the steps below for $i = 1, 2, \ldots$:
    - E tries every possible inputs w of length at most i,
    - For each input w, E runs M for i steps, and prints w if M accepts it."

Equivalence in Power

- **Should I write programs in C or Pascal?**

Equivalence in Power

- **Should I write programs in C or Pascal?**
- **Should I write programs in Python or Prolog?**

Equivalence in Power

- **Should I write programs in C or Pascal?**
- **Should I write programs in Python or Prolog?**
- **Should I write programs in Ruby or LISP?**

They are all the same, in terms of computability

Since you can write a C interpreter in Pascal and Pascal interpreter in C,

They are all the same, in terms of computability

Since you can write a C interpreter in Pascal and Pascal interpreter in C, what you can do in C, you can do in Pascal.

Turing machine

If you believe that Turing machines are ultimate model of computing, all those programming languages are equivalent because they all can simulate Turing machines (and they runs on Turing machines).

Two sides of a coin

- Computers are powerful

Two sides of a coin

- Computers are powerful (???)

Two sides of a coin

- Computers are powerful (???)
- How powerful?

Two sides of a coin

- Computers are powerful (???)
- How powerful?
- To understand that, we want to see samples of tasks that computers can do, and samples of tasks that they can't do.

Two sides of a coin

- Computers are powerful (???)
- How powerful?
- To understand that, we want to see samples of tasks that computers can do, and samples of tasks that they can't do.
- It's one story to show that computers can do something.

Two sides of a coin

- Computers are powerful (???)
- How powerful?
- To understand that, we want to see samples of tasks that computers can do, and samples of tasks that they can't do.
- It's one story to show that computers can do something. It's another to show that computers can't do something.
  - Maybe there's a limitation with "this" computer, but "other" computers might be able to do that thing.
  - We want to be able to say that **for all** computers.

Two sides of a coin

- Computers are powerful (???)
- How powerful?
- To understand that, we want to see samples of tasks that computers can do, and samples of tasks that they can't do.
- It's one story to show that computers can do something. It's another to show that computers can't do something.
  - Maybe there's a limitation with "this" computer, but "other" computers might be able to do that thing.
  - We want to be able to say that **for all** computers. In fact, **for any "thinkable"** computers.

What is a computer?

What is a computer?
Something that computes?

What is a computer?
Something that computes?
What is computation?

What is a computer?
Something that computes?
What is computation?
An act of following some instructions?

What is a computer?
Something that computes?
What is computation?
An act of following some instructions?
An act of following some algorithm?

What is a computer?
Something that computes?
What is computation?
An act of following some instructions?
An act of following some algorithm?
What is an algorithm?

Hilbert's problems

Mathematician David Hilbert asked:
"Find a process according to which it can be determined by a finite number of operations if a given polynomial has intergral root"

To say NO

We need an argument (a mathematical proof) that covers all possible "processes" or all "computations".

Possible definitions

- Church's λ-calculus
- Turing's machines

Possible definitions

- Church's λ-calculus
- Turing's machines

They both turned out to be **equivalent**.

Church-Turing thesis

Turing machine algorithms = intuitive notion of algorithms

Final answer to Hilbert

No, there doesn't exist any algorithm for determining if a polynomial has integral root.

Notes

- There exists an enumerator of all polynomials with integral roots.

Notes

- There exists an enumerator of all polynomials with integral roots.
- For univariate polynomials, there exists a TM that can determine if a give univariate polynomial has integral root.

Describing Turing machines

There are many levels of description.

Describing Turing machines

There are many levels of description.

- Formal description — describes all 7 tuples,
- Implementation description — describes in English how TM works including how to move TM's head and how to maintain content on the tape,
- High-level description — describes algorithms in English, without even mentioning about the tape.

Describing Turing machines

There are many levels of description.

- Formal description — describes all 7 tuples,
- Implementation description — describes in English how TM works including how to move TM's head and how to maintain content on the tape,
- High-level description — describes algorithms in English, without even mentioning about the tape. **This is the format that we'll user later on.**

High-level descriptions

This is a format that we'll use.

High-level descriptions

This is a format that we'll use.

- **Input encoding:** If the input to a TM is an object O, we have to encode it as a string. We denote the encoded object as $\langle O \rangle$.

High-level descriptions

This is a format that we'll use.

- **Input encoding:** If the input to a TM is an object O, we have to encode it as a string. We denote the encoded object as $\langle O \rangle$.
- **Algorithm:** We describe algorithms in quotes, use indent to specify block structures, and specify the input on the first line.

Example: graph connectivity (1)

A graph is **connected** if every node can be reached from every other node along the edges in the graph.
Let A be the language consisting of all strings representing undirected graphs that are connected.

Example: graph connectivity (1)

A graph is **connected** if every node can be reached from every other node along the edges in the graph.
Let A be the language consisting of all strings representing undirected graphs that are connected. Write

$$A = \{\langle G\rangle \mid G \text{ is a connected undirected graph}\}.$$

Example: graph connectivity (2)

$M =$ "On input $\langle G \rangle$, the encoding of G:

1. Selected the first node of G and mark it.
2. Repeat the following stage until no new nodes are marked:
3. For each node in G, mark it if it is attached by an edge to a node that is already marked.
4. Scan all the node of G to determine if they all are marked. Output 'accept' if they are, output 'reject' otherwise."

Decidable languages

Definition

A language is called **Turing-decidable** or **decidable** if some Turing machine decides it.

Decidable problems concerning regular languages

- Acceptance problems
- Emptiness testing
- Equivalence

Acceptance problems: DFA

Let

$$A_{DFA} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}.$$

Theorem 2

A_{DFA} is a decidable language.

Acceptance problems: NFA

Let

$$A_{NFA} = \{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\}.$$

Theorem 3

A_{NFA} is a decidable language.

Acceptance problems: Regular expression

Let

$$A_{REX} = \{\langle B, w\rangle \mid$$
$$B \text{ is a regular expression that generates input string } w\}.$$

Theorem 4

A_{REX} is a decidable language.

Emptiness testing

Let

$$E_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$$

Theorem 5

E_{DFA} is a decidable language.

Equivalence testing

Let

$$EQ_{DFA} = \{\langle A, B \rangle \mid A \text{ and } B \text{ are DFA's and } L(A) = L(B)\}$$

Theorem 6

EQ_{DFA} is a decidable language.

Decidable problems concerning context-free languages

- Acceptance problems
- Emptiness testing
- Equivalence (?)

Acceptance

Let

$$A_{CFG} = \{\langle G, w\rangle \mid C \text{ is a CFG that generates input string } w\}.$$

Theorem 7

A_{CFG} is a decidable language.

Emptiness testing

Let

$$E_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$$

Theorem 8

E_{CFG} is a decidable language.

Equivalence testing

Let

$$EQ_{CFG} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are CFG's and } L(A) = L(B)\}$$

Can we use the previous techniques?

Equivalence testing

Let

$$EQ_{CFG} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are CFG's and } L(A) = L(B)\}$$

Can we use the previous techniques?
It turns out that EQ_{CFG} is undecidable

Decidability of CFL

Theorem 9

Every context-free language is decidable.

It's now time to study
something that computers can't do.
You may think that machines are cool.
But if you believe that
there's nothing that it can't do,
You may be a fool.

Software verification

Acceptance problem of TMs

Let

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts input string } w\}.$$

Theorem 10

A_{TM} is undecidable.

A_{TM} is recognizable

$U =$ "On input $\langle M, w \rangle$, where M is a TM and w is a string:

1. Simulate M on input w.
2. If M ever enters its accept state, accept, if M ever enters its reject state, reject."

A_{TM} is recognizable

$U =$ "On input $\langle M, w \rangle$, where M is a TM and w is a string:

1. Simulate M on input w.
2. If M ever enters its accept state, accept, if M ever enters its reject state, reject."

- U loops when M loops on w. If there's a way to decide if M loops on w, we can turn U into a decider.

A_{TM} is recognizable

$U =$ "On input $\langle M, w \rangle$, where M is a TM and w is a string:

1. Simulate M on input w.
2. If M ever enters its accept state, accept, if M ever enters its reject state, reject."

- U loops when M loops on w. If there's a way to decide if M loops on w, we can turn U into a decider.
- U a TM that simulates other TM. U is called a **universal Turing machine**.

Diagonalization method

- Diagonalization is a method that mathematician Georg Cantor used to show that two infinite sets, i.e., real numbers and integers, are of different size.

Comparing the sizes of two infinite sets

How can we say that two sets are of the same size?

Comparing the sizes of two infinite sets

How can we say that two sets are of the same size?

- For finite sets, we just compare the numbers of their members

Comparing the sizes of two infinite sets

How can we say that two sets are of the same size?

- For finite sets, we just compare the numbers of their members
- Doesn't work for infinite sets.