Church - Turing Thesis

CENG 280, 2019

Theory of computation

What can be computed ?

Theory of computation

What can be computed ?

What can not be computed?

Theory of Computation

- decide a language
- semi-decide a language
- compute a function
- generate a language

Theory of Computation

- decide a language
- semi-decide a language
- compute a function
- generate a language

(informally) algorithm: a rule for solving a problem in a finite number of steps

Theory of Computation

- decide a language
- semi-decide a language
- compute a function
- generate a language

(informally) algorithm: a rule for solving a problem in a finite number of steps

TMs that halt on every input word are called algorithms.

- decide a language
- semi-decide a language
- compute a function
- generate a language

(informally) algorithm: a rule for solving a problem in a finite number of steps

TMs that halt on every input word are called algorithms.

Church - Turing thesis: An algorithm corresponds to a TM that halts on all inputs.

Church - Turing Thesis

Church - Turing thesis: An algorithm corresponds to a TM that halts on all inputs.
CHURCH: A function of positive integers is effectively computable only if recursive.

TURING: TM can do anything that could be described purely mathematical.

Church - Turing Thesis

Church - Turing thesis: An algorithm corresponds to a TM that halts on all inputs.
CHURCH: A function of positive integers is effectively computable only if recursive.

TURING: TM can do anything that could be described purely mathematical.

What can not be computed? Only if there is no TM for it.

Church - Turing Thesis

Church - Turing thesis: An algorithm corresponds to a TM that halts on all inputs.
CHURCH: A function of positive integers is effectively computable only if recursive.

TURING: TM can do anything that could be described purely mathematical.

What can not be computed? Only if there is no TM for it.

Countably many recursive (or recursively enumerable) languages, uncountably many languages.....

What can not be computed?

- Define a TM that takes TM as inputs !!! (universal TM)

What can not be computed?

- Define a TM that takes TM as inputs !!! (universal TM)
- Universal TM manipulates input TM's encodings.

What can not be computed?

- Define a TM that takes TM as inputs !!! (universal TM)
- Universal TM manipulates input TM's encodings.
- What happens when universal TM receives its own encoding as input ? (diagonalization principle)

Universal Turing Machine

Represent TMs as strings:

- States $q00$, $q01$, $q10$, $q11$ (increase the number of bits w.r.to the number of states)

Universal Turing Machine

Represent TMs as strings:

- States $q00$, $q01$, $q10$, $q11$ (increase the number of bits w.r.to the number of states)
- min i such that $2^i \geq |K|$, $q + i$ bits

Universal Turing Machine

Represent TMs as strings:

- States $q00$, $q01$, $q10$, $q11$ (increase the number of bits w.r.to the number of states)
- min i such that $2^i \geq |K|$, $q + i$ bits
- (fix representation) $q0^i$ is the start state

Universal Turing Machine

Represent TMs as strings:

- States $q00$, $q01$, $q10$, $q11$ (increase the number of bits w.r.to the number of states)
- min i such that $2^i \geq |K|$, $q + i$ bits
- (fix representation) $q0^i$ is the start state
- Symbols $a0$, $a1$

Universal Turing Machine

Represent TMs as strings:

- States $q00$, $q01$, $q10$, $q11$ (increase the number of bits w.r.to the number of states)
- min i such that $2^i \geq |K|$, $q + i$ bits
- (fix representation) $q0^i$ is the start state
- Symbols $a0$, $a1$
- min j such that $2^j \geq |\Sigma| + 2$ (left/right), $a + j$ bits

Universal Turing Machine

Represent TMs as strings:

- States $q00$, $q01$, $q10$, $q11$ (increase the number of bits w.r.to the number of states)
- min i such that $2^i \geq |K|$, $q + i$ bits
- (fix representation) $q0^i$ is the start state
- Symbols $a0$, $a1$
- min j such that $2^j \geq |\Sigma| + 2$ (left/right), $a + j$ bits
-
    - $\sqcup$: $a0^j$
    - $\triangleright$: $a0^{j-1}1$
    - $\leftarrow$: $a0^{j-2}10$
    - $\rightarrow$: $a0^{j-2}11$

Universal Turing Machine

$$U(\text{``}M\text{''}\ \text{``}w\text{''}) = \text{``}M(w)\text{''}$$

Universal Turing Machine

$$U(\text{“}M\text{”}\,\text{“}w\text{”}) = \text{“}M(w)\text{”}$$

U halts on “M” “w” iff M halts on w.

Universal Turing Machine

$$U(\text{“}M\text{”}\,\text{“}w\text{”}) = \text{“}M(w)\text{”}$$

U halts on “M” “w” iff M halts on w.

Define U', a 3 tape TM that simulates U

Universal Turing Machine

$$U(\text{“}M\text{”}\,\text{“}w\text{”}) = \text{“}M(w)\text{”}$$

U halts on "M" "w" iff M halts on w.

Define U', a 3 tape TM that simulates U

- Initially $\rhd\sqcup$"M" "w" is on the first tape of U'

Universal Turing Machine

$$U(\text{“}M\text{”}\,\text{“}w\text{”}) = \text{“}M(w)\text{”}$$

U halts on “M” “w” iff M halts on w.

Define U', a 3 tape TM that simulates U

- Initially $\rhd\sqcup$“M” “w” is on the first tape of U'
- U' copies “M” to the second tape, shifts “w” to the left ($\rhd\sqcup$“w”)

Universal Turing Machine

$$U(\text{“}M\text{”}\,\text{“}w\text{”}) = \text{“}M(w)\text{”}$$

U halts on “M” “w” iff M halts on w.

Define U', a 3 tape TM that simulates U

- Initially $\triangleright\sqcup$“M” “w” is on the first tape of U'
- U' copies “M” to the second tape, shifts “w” to the left ($\triangleright\sqcup$“w”)
- U' writes $q0^i$ to its third tape (initial state)

Universal Turing Machine

$$U(\text{“}M\text{”}\,\text{“}w\text{”}) = \text{“}M(w)\text{”}$$

U halts on “M” “w” iff M halts on w.

Define U', a 3 tape TM that simulates U

- Initially $\triangleright\sqcup$“M” “w” is on the first tape of U'
- U' copies “M” to the second tape, shifts “w” to the left ($\triangleright\underline{\sqcup}$“$w$”)
- U' writes $q0^i$ to its third tape (initial state)
- Then U' simulates M as follows:
  - U' scans its second tape until it finds a quadruple (a transition) whose first element matches the string in the third tape (e.g. the state) and second element matches the string on the first tape (e.g. the symbol under the reading head of M)
  - If it finds such a quadruple, updates the string on the third tape (the state of M), and performs the action on the first tape (move the head, or write a symbol)
  - If it can not find a matching quadruple or the state is in H, then halt.

The Halting Problem

Halts(P, X)

$$\text{Halts}(P, X) = \begin{cases} YES & \text{if } P \text{ halts on } X \\ NO & \text{if } P \text{ does not halt on } X \end{cases}$$

The Halting Problem

Halts(P, X)

YES	if P halts on X
NO	if P does not halt on X

diagonal(X)
a : if $Halts(X, X)$ then go to a else halt

The Halting Problem

Halts(P, X)

YES	if P halts on X
NO	if P does not halt on X

diagonal(X)
a : if $Halts(X, X)$ then go to a else halt

Does $diagonal(diagonal)$ **halt?**

The Halting Problem

Halts(P, X)

YES	if P halts on X
NO	if P does not halt on X

diagonal(X)
a : if $Halts(X, X)$ then go to a else halt

Does $diagonal(diagonal)$ **halt?**

The Halting Problem

Halts(P, X)

YES	if P halts on X
NO	if P does not halt on X

diagonal(X)
a : if $Halts(X, X)$ then go to a else halt

Does $diagonal(diagonal)$ **halt?**

$diagonal(diagonal)$ halts if $halts(diagonal, diagonal)$ returns NO.
$diagonal(diagonal)$ does not halt if $halts(diagonal, diagonal)$ returns YES.

The Halting Problem

Halts(P, X)

$$\begin{array}{ll} YES & \text{if } P \text{ halts on } X \\ NO & \text{if } P \text{ does not halt on } X \end{array}$$

diagonal(X)
a : if $Halts(X, X)$ then go to a else halt

Does $diagonal(diagonal)$ **halt?**

$diagonal(diagonal)$ halts if $halts(diagonal, diagonal)$ returns NO.
$diagonal(diagonal)$ does not halt if $halts(diagonal, diagonal)$ returns YES.

The contradiction implies that the initial hypothesis, that $Halts(P, X)$ exists, is wrong. There can be no program, no algorithm to tell whether arbitrary programs halt or loop.

The Halting Problem

Halts(P, X)

YES	if P halts on X
NO	if P does not halt on X

diagonal(X)
a : if $Halts(X, X)$ then go to a else halt

Does $diagonal(diagonal)$ **halt?**

$diagonal(diagonal)$ halts if $halts(diagonal, diagonal)$ returns NO.
$diagonal(diagonal)$ does not halt if $halts(diagonal, diagonal)$ returns YES.

The contradiction implies that the initial hypothesis, that $Halts(P, X)$ exists, is wrong. There can be no program, no algorithm to tell whether arbitrary programs halt or loop. **The halting problem is undecidable**

A language that is not recursive

A language that is not recursive

$$H = \{\text{“}M\text{” “}w\text{”} \mid \text{ Turing Machine } M \text{ halts on } w\}$$

A language that is not recursive

$$H = \{\text{“}M\text{” “}w\text{”} \mid \text{ Turing Machine } M \text{ halts on } w\}$$

H is recursively enumerable, proof?

A language that is not recursive

$$H = \{ \text{“}M\text{”}\,\text{“}w\text{”} \mid \text{ Turing Machine } M \text{ halts on } w \}$$

H is recursively enumerable, proof?
H is the language semi-decided by the universal TM U

A language that is not recursive

$$H = \{\text{“}M\text{” “}w\text{”} \mid \text{ Turing Machine } M \text{ halts on } w\}$$

H is recursively enumerable, proof?
H is the language semi-decided by the universal TM U

“If H is recursive, then all recursively enumerable languages are recursive.”

A language that is not recursive

$$H = \{\text{“}M\text{” “}w\text{”} \mid \text{ Turing Machine } M \text{ halts on } w\}$$

H is recursively enumerable, proof?
H is the language semi-decided by the universal TM U

“If H is recursive, then all recursively enumerable languages are recursive.”

- Suppose it is true, and H is decided by M_0.

A language that is not recursive

$$H = \{\text{“}M\text{” “}w\text{”} \mid \text{ Turing Machine } M \text{ halts on } w\}$$

H is recursively enumerable, proof?
H is the language semi-decided by the universal TM U

“If H is recursive, then all recursively enumerable languages are recursive.”

- Suppose it is true, and H is decided by M_0.
- Given M semi-deciding $L(M)$

A language that is not recursive

$$H = \{\text{“}M\text{” “}w\text{”} \mid \text{ Turing Machine } M \text{ halts on } w\}$$

H is recursively enumerable, proof?
H is the language semi-decided by the universal TM U

"If H is recursive, then all recursively enumerable languages are recursive."

- Suppose it is true, and H is decided by M_0.
- Given M semi-deciding $L(M)$
- Design M' that decides $L(M)$ as follows:
  - Transform $\triangleright \sqcup w$ to $\triangleright \sqcup$ "M" "w" and simulate M_0 on this
  - If M_0 halts on y, then M' halts on y
  - If M_0 halts on n, then M' halts on n

Prove that H is not recursive

If H were recursive, then H_1 is recursive (e.g. $halts(X, X)$)

$$H_1 = \{\text{"}M\text{"} \mid \text{ TM } M \text{ halts on "}M\text{"}\}$$

Prove that H is not recursive

If H were recursive, then H_1 is recursive (e.g. $halts(X, X)$)

$$H_1 = \{\text{"}M\text{"} \mid \text{ TM } M \text{ halts on } \text{"}M\text{"}\}$$

M_1 decides H_1 by using M_0.
M_1 : Transform $\triangleright \sqcup$ "M" to $\triangleright \sqcup$ "M" "M" and run M_0 on it

Prove that H is not recursive

If H were recursive, then H_1 is recursive (e.g. $halts(X, X)$)

$$H_1 = \{\text{"}M\text{"} \mid \text{ TM } M \text{ halts on } \text{"}M\text{"}\}$$

M_1 decides H_1 by using M_0.
M_1 : Transform $\rhd \sqcup$ "M" to $\rhd \sqcup$ "M" "M" and run M_0 on it

If H_1 is recursive, then it's complement is also recursive. (analogous to the diagonal program)

$\overline{H}_1 = \{w \mid w$ is not an encoding of a TM or it encodes a TM "M" that does not halt on "M" "$\}$

Prove that H is not recursive

If H were recursive, then H_1 is recursive (e.g. $halts(X, X)$)

$$H_1 = \{\text{"}M\text{"} \mid \text{ TM } M \text{ halts on } \text{"}M\text{"}\}$$

M_1 decides H_1 by using M_0.
M_1 : Transform $\triangleright \sqcup$ "M" to $\triangleright \sqcup$ "M" "M" and run M_0 on it

If H_1 is recursive, then it's complement is also recursive. (analogous to the diagonal program)

$\overline{H}_1 = \{w \mid w$ is not an encoding of a TM or it encodes a TM "M" that does not halt on "M" "$\}$

$\overline{H}_1$ is not even recursively enumerable yet alone recursive.

Prove that $\overline{H_1}$ is not recursively enumerable

- Assume $M^\star$ is a TM that semidecides $\overline{H_1}$.

Prove that $\overline{H}_1$ is not recursively enumerable

- Assume $M^\star$ is a TM that semidecides $\overline{H}_1$.
- Is "$M^\star$" in $\overline{H}_1$?

Prove that $\overline{H}_1$ is not recursively enumerable

- Assume $M^\star$ is a TM that semidecides $\overline{H}_1$.
- Is "$M^\star$" in $\overline{H}_1$?
  - By definition of $\overline{H}_1$:

$$\text{“}M^\star\text{”} \in \overline{H}_1 \text{ iff } M^\star \text{ does not halt on “}M^\star\text{”}$$

Prove that $\overline{H}_1$ is not recursively enumerable

- Assume $M^\star$ is a TM that semidecides $\overline{H}_1$.
- Is "$M^\star$" in $\overline{H}_1$?
  - By definition of $\overline{H}_1$:

    $$\text{“}M^\star\text{”} \in \overline{H}_1 \text{ iff } M^\star \text{ does not halt on “}M^\star\text{”}$$

  - By definition of $M^\star$, it semidecides $\overline{H}_1$, thus

    $$\text{“}M^\star\text{”} \in \overline{H}_1 \text{ iff } M^\star \text{ halts on “}M^\star\text{”}$$

Prove that $\overline{H}_1$ is not recursively enumerable

- Assume $M^{\star}$ is a TM that semidecides $\overline{H}_1$.
- Is "$M^{\star}$" in $\overline{H}_1$?
  - By definition of $\overline{H}_1$:

$$\text{“}M^{\star}\text{”} \in \overline{H}_1 \text{ iff } M^{\star} \text{ does not halt on “}M^{\star}\text{”}$$

  - By definition of $M^{\star}$, it semidecides $\overline{H}_1$, thus

$$\text{“}M^{\star}\text{”} \in \overline{H}_1 \text{ iff } M^{\star} \text{ halts on “}M^{\star}\text{”}$$

Contradiction: Our initial assumption is wrong, thus M_0 does not exists.

Prove that $\overline{H_1}$ is not recursively enumerable

- Assume $M^\star$ is a TM that semidecides $\overline{H_1}$.
- Is "$M^\star$" in $\overline{H_1}$?
  - By definition of $\overline{H_1}$:

$$\text{“}M^\star\text{”} \in \overline{H_1} \text{ iff } M^\star \text{ does not halt on “}M^\star\text{”}$$

  - By definition of $M^\star$, it semidecides $\overline{H_1}$, thus

$$\text{“}M^\star\text{”} \in \overline{H_1} \text{ iff } M^\star \text{ halts on “}M^\star\text{”}$$

Contradiction: Our initial assumption is wrong, thus M_0 does not exists.
H is not recursive, the class of recursive languages is a strict subset of the class of recursively enumerable languages

Prove that $\overline{H}_1$ is not recursively enumerable

- Assume $M^\star$ is a TM that semidecides $\overline{H}_1$.
- Is "$M^\star$" in $\overline{H}_1$?
  - By definition of $\overline{H}_1$:

$$\text{"}M^\star\text{"} \in \overline{H}_1 \text{ iff } M^\star \text{ does not halt on "}M^\star\text{"}$$

  - By definition of $M^\star$, it semidecides $\overline{H}_1$, thus

$$\text{"}M^\star\text{"} \in \overline{H}_1 \text{ iff } M^\star \text{ halts on "}M^\star\text{"}$$

Contradiction: Our initial assumption is wrong, thus M_0 does not exists.
H is not recursive, the class of recursive languages is a strict subset of the class of recursively enumerable languages
The class of recursively enumerable languages are not closed under complement