Pre-lecture brain teaser

For each of the following languages is the language decidable?

- $\mathsf{A}_{\mathsf{DFA}} = \{\langle B, w\rangle | B \text{ is a DFA that accepts w}\}$
- $\mathsf{A}_{\mathsf{NFA}} = \{\langle B, w\rangle | B \text{ is a NFA that accepts w}\}$

CS/ECE-374: Lecture 24 - Decidability

Lecturer: Nickvash Kani
Chat moderator: Samir Khan

April 20, 2021

University of Illinois at Urbana-Champaign

Pre-lecture brain teaser

For each of the following languages is the language decidable?

- $A_{DFA} = \{\langle B, w\rangle | B \text{ is a DFA that accepts w}\}$
- $A_{NFA} = \{\langle B, w\rangle | B \text{ is a NFA that accepts w}\}$

Turing machines...

TM = Turing machine = program.

Reminder: Undecidability

Definition
Language $L \subseteq \Sigma^*$ is undecidable if no program P, given $w \in \Sigma^*$ as input, can **always stop** and output whether $w \in L$ or $w \notin L$.

(Usually defined using TM not programs. But equivalent.

Reminder: Undecidability

Definition
Language $L \subseteq \Sigma^*$ is undecidable if no program P, given $w \in \Sigma^*$ as input, can **always stop** and output whether $w \in L$ or $w \notin L$.

(Usually defined using TM not programs. But equivalent.

Definition
Language $L \subseteq \Sigma^*$ is undecidable if no program P, given $w \in \Sigma^*$ as input, can

always stop and output

whether $w \in L$ or $w \notin L$.

(Usually defined using TM not programs. But equivalent.

Reminder: The following language is undecidable

Decide if given a program M, and an input w, does M accepts w.
Formally, the corresponding language is

$$\mathrm{A}_{TM} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a } TM \text{ and } M \text{ accepts } w \right\}.$$

Reminder: The following language is undecidable

Decide if given a program M, and an input w, does M accepts w. Formally, the corresponding language is

$$\mathrm{A}_{TM} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a } TM \text{ and } M \text{ accepts } w \right\}.$$

Definition
A *decider* for a language L, is a program (or a TM) that always stops, and outputs for any input string $w \in \Sigma^*$ whether or not $w \in L$.

A language that has a decider is *decidable*.

Reminder: The following language is undecidable

Decide if given a program M, and an input w, does M accepts w. Formally, the corresponding language is

$$\mathrm{A}_{TM} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a } TM \text{ and } M \text{ accepts } w \right\}.$$

Definition
A *decider* for a language L, is a program (or a TM) that always stops, and outputs for any input string $w \in \Sigma^*$ whether or not $w \in L$.

A language that has a decider is *decidable*.

Turing proved the following:

Theorem
A_{TM} *is undecidable.*

The halting problem

A_{TM} is not TM decidable!

$$\mathrm{A}_{TM} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a } TM \text{ and } M \text{ accepts } w \right\}.$$

Theorem (The halting theorem.)
A_{TM} *is not Turing decidable.*

A_{TM} is not TM decidable!

$$\mathrm{A}_{TM} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a } TM \text{ and } M \text{ accepts } w \right\}.$$

Theorem (The halting theorem.)
A_{TM} is not Turing decidable.

Proof: Assume A_{TM} is TM decidable...

A_{TM} is not TM decidable!

$$\mathrm{A}_{TM} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a } TM \text{ and } M \text{ accepts } w \right\}.$$

Theorem (The halting theorem.)
A_{TM} is not Turing decidable.

Proof: Assume A_{TM} is TM decidable...

Halt: TM deciding A_{TM}. Halt always halts, and works as follows:

$$\text{Halt}\Big(\langle M, w \rangle \Big) = \begin{cases} \text{accept} & M \text{ accepts } w \\ \text{reject} & M \text{ does not accept } w. \end{cases}$$

Halting theorem proof continued 1

We build the following new function:

```
Flipper(⟨M⟩)
    res ← Halt(⟨M, M⟩)
    if res is accept then
                reject
    else
                accept
```

Halting theorem proof continued 1

We build the following new function:

```
Flipper(⟨M⟩)
    res ← Halt(⟨M, M⟩)
    if res is accept then
            reject
    else
            accept
```

Flipper *always stops*:

$$\text{Flipper}\Big(\langle M\rangle\Big) = \begin{cases} \text{reject} & M \text{ accepts } \langle M\rangle \\ \text{accept} & M \text{ does not accept } \langle M\rangle \,. \end{cases}$$

Halting theorem proof continued 2

$$\text{Flipper}\Big(\langle M\rangle\Big) = \begin{cases} \text{reject} & M \text{ accepts } \langle M\rangle \\ \text{accept} & M \text{ does not accept } \langle M\rangle\,. \end{cases}$$

Flipper is a TM (duh!), and as such it has an encoding ⟨Flipper⟩.
Run Flipper on itself:

$$\text{Flipper}\Big(\langle \text{Flipper}\rangle\Big) = \begin{cases} \text{reject} & \text{Flipper accepts } \langle \text{Flipper}\rangle \\ \text{accept} & \text{Flipper does not accept } \langle \text{Flipper}\rangle\,. \end{cases}$$

Halting theorem proof continued 2

$$\text{Flipper}\big(\langle M\rangle\big) = \begin{cases} \text{reject} & M \text{ accepts } \langle M\rangle \\ \text{accept} & M \text{ does not accept } \langle M\rangle\,. \end{cases}$$

Flipper is a TM (duh!), and as such it has an encoding ⟨Flipper⟩.
Run Flipper on itself:

$$\text{Flipper}\big(\langle \text{Flipper}\rangle\big) = \begin{cases} \text{reject} & \text{Flipper accepts } \langle \text{Flipper}\rangle \\ \text{accept} & \text{Flipper does not accept } \langle \text{Flipper}\rangle\,. \end{cases}$$

This is absurd. Ridiculous even!

Halting theorem proof continued 2

$$\text{Flipper}\left(\langle M \rangle\right) = \begin{cases} \text{reject} & M \text{ accepts } \langle M \rangle \\ \text{accept} & M \text{ does not accept } \langle M \rangle . \end{cases}$$

Flipper is a TM (duh!), and as such it has an encoding ⟨Flipper⟩.
Run Flipper on itself:

$$\text{Flipper}\left(\langle \text{Flipper} \rangle\right) = \begin{cases} \text{reject} & \text{Flipper accepts } \langle \text{Flipper} \rangle \\ \text{accept} & \text{Flipper does not accept } \langle \text{Flipper} \rangle . \end{cases}$$

This is absurd. Ridiculous even!

Assumption that Halt exists is false. $\implies$ A_{TM} is not TM decidable. □

Reductions

Reduction

Meta definition: Problem X *reduces* to problem B, if given a solution to B, then it implies a solution for X. Namely, we can solve Y then we can solve X. We will done this by $X \implies Y$.

Reduction

Meta definition: Problem X *reduces* to problem B, if given a solution to B, then it implies a solution for X. Namely, we can solve Y then we can solve X. We will done this by X $\implies$ Y.

Definition
oracle ORAC for language L is a function that receives as a word w, returns TRUE $\iff w \in L$.

Reduction

Meta definition: Problem X *reduces* to problem B, if given a solution to B, then it implies a solution for X. Namely, we can solve Y then we can solve X. We will done this by X $\implies$ Y.

Definition

oracle ORAC for language L is a function that receives as a word w, returns TRUE $\iff$ $w \in L$.

Lemma

A language X reduces *to a language Y, if one can construct a TM decider for X using a given oracle* ORAC_Y *for Y.*

We will denote this fact by $X \implies Y$.

Reduction proof technique

- Y: Problem/language for which we want to prove undecidable.

Reduction proof technique

- Y: Problem/language for which we want to prove undecidable.
- Proof via reduction. Result in a proof by contradiction.

Reduction proof technique

- Y: Problem/language for which we want to prove undecidable.
- Proof via reduction. Result in a proof by contradiction.
- L: language of Y.

Reduction proof technique

- Y: Problem/language for which we want to prove undecidable.
- Proof via reduction. Result in a proof by contradiction.
- L: language of Y.
- Assume L is decided by TM M.

Reduction proof technique

- Y: Problem/language for which we want to prove undecidable.
- Proof via reduction. Result in a proof by contradiction.
- L: language of Y.
- Assume L is decided by TM M.
- Create a decider for known undecidable problem X using M.

Reduction proof technique

- Y: Problem/language for which we want to prove undecidable.
- Proof via reduction. Result in a proof by contradiction.
- L: language of Y.
- Assume L is decided by TM M.
- Create a decider for known undecidable problem X using M.
- Result in decider for X (i.e., A_{TM}).

Reduction proof technique

- Y: Problem/language for which we want to prove undecidable.
- Proof via reduction. Result in a proof by contradiction.
- L: language of Y.
- Assume L is decided by TM M.
- Create a decider for known undecidable problem X using M.
- Result in decider for X (i.e., A_{TM}).
- Contradiction X is not decidable.

Reduction proof technique

- Y: Problem/language for which we want to prove undecidable.
- Proof via reduction. Result in a proof by contradiction.
- L: language of Y.
- Assume L is decided by TM M.
- Create a decider for known undecidable problem X using M.
- Result in decider for X (i.e., A_{TM}).
- Contradiction X is not decidable.
- Thus, L must be not decidable.

Reduction implies decidability

Lemma

Let X and Y be two languages, and assume that $X \implies Y$. If Y is decidable then X is decidable.

Proof.

Let T be a decider for Y (i.e., a program or a TM). Since X reduces to Y, it follows that there is a procedure $\mathsf{T}_{X|Y}$ (i.e., decider) for X that uses an oracle for Y as a subroutine. We replace the calls to this oracle in $\mathsf{T}_{X|Y}$ by calls to T. The resulting program T_X is a decider and its language is X. Thus X is decidable (or more formally TM decidable). □

The countrapositive...

Lemma
Let X and Y be two languages, and assume that $X \implies Y$. If X is undecidable then Y is undecidable.

Halting

The halting problem

Language of all pairs $\langle M, w\rangle$ such that M *halts* on w:

$$A_{\mathrm{Halt}} = \left\{ \langle M, w\rangle \;\middle|\; M \text{ is a } TM \text{ and } M \text{ stops on } w \right\}.$$

Similar to language already known to be undecidable:

$$\mathrm{A}_{TM} = \left\{ \langle M, w\rangle \;\middle|\; M \text{ is a } TM \text{ and } M \text{ accepts } w \right\}.$$

On way to proving that Halting is undecidable...

Lemma

The language A_{TM} *reduces to* A_{Halt}. *Namely, given an oracle for* A_{Halt} *one can build a decider (that uses this oracle) for* A_{TM}.

On way to proving that Halting is undecidable...

Proof.
Let ORAC_{Halt} be the given oracle for A_{Halt}. We build the following decider for A_{TM}.

```
AnotherDecider-A_TM(⟨M, w⟩)
        res ← ORAC_Halt(⟨M, w⟩)
        // if M does not halt on w then reject.
        if res = reject then
                halt and reject.
        // M halts on w since res =accept.
        // Simulating M on w terminates in finite time.
        res_2 ←Simulate M on w.
        return res_2.
```

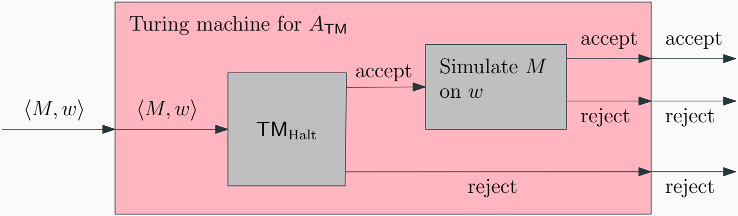
This procedure always return and as such its a decider for A_{TM}. □

The Halting problem is not decidable

Theorem
The language A_{Halt} is not decidable.

Proof.
Assume, for the sake of contradiction, that A_{Halt} is decidable. As such, there is a TM, denoted by TM_{Halt}, that is a decider for A_{Halt}. We can use TM_{Halt} as an implementation of an oracle for A_{Halt}, which would imply that one can build a decider for A_{TM}. However, A_{TM} is undecidable. A contradiction. It must be that A_{Halt} is undecidable. □

The same proof by figure...



... if A_{Halt} is decidable, then A_{TM} is decidable, which is impossible.

Emptiness

The language of empty languages

- $E_{TM} = \left\{ \langle M \rangle \,\middle|\, M \text{ is a TM and } L(M) = \emptyset \right\}$.
- TM_{ETM}: Assume we are given this decider for E_{TM}.
- Need to use TM_{ETM} to build a decider for A_{TM}.
- Decider for A_{TM} is given M and w and must decide whether M accepts w.
- Restructure question to be about Turing machine having an empty language.
- Somehow make the second input (w) disappear.

The language of empty languages

- $E_{TM} = \left\{ \langle M \rangle \;\middle|\; M \text{ is a TM and } L(M) = \emptyset \right\}$.
- TM_{ETM}: Assume we are given this decider for E_{TM}.
- Need to use TM_{ETM} to build a decider for A_{TM}.
- Decider for A_{TM} is given M and w and must decide whether M accepts w.
- Restructure question to be about Turing machine having an empty language.
- Somehow make the second input (w) disappear.
- Idea: hard-code w into M, creating a TM M_w which runs M on the fixed string w.
- TM M_w:
    1. Input = x (which will be ignored)
    2. Simulate M on w.
    3. If the simulation accepts, accept. If the simulation rejects, reject.

Embedding strings...

- Given program $\langle M \rangle$ and input w...
- ...can output a program $\langle M_w \rangle$.
- The program M_w simulates M on w. And accepts/rejects accordingly.
- EmbedString($\langle M, w \rangle$) input two strings $\langle M \rangle$ and w, and output a string encoding (TM) $\langle M_w \rangle$.

Embedding strings...

- Given program $\langle M \rangle$ and input w...
- ...can output a program $\langle M_w \rangle$.
- The program M_w simulates M on w. And accepts/rejects accordingly.
- EmbedString($\langle M, w \rangle$) input two strings $\langle M \rangle$ and w, and output a string encoding (TM) $\langle M_w \rangle$.
- What is $L(M_w)$?

Embedding strings...

- Given program $\langle M \rangle$ and input w...
- ...can output a program $\langle M_w \rangle$.
- The program M_w simulates M on w. And accepts/rejects accordingly.
- EmbedString($\langle M, w \rangle$) input two strings $\langle M \rangle$ and w, and output a string encoding (TM) $\langle M_w \rangle$.
- What is $L(M_w)$?
- Since M_w ignores input x.. language M_w is either Σ^* or $\emptyset$. It is Σ^* if M accepts w, and it is $\emptyset$ if M does not accept w.

Emptiness is undecidable

Theorem
The language E_{TM} is undecidable.

- Assume (for contradiction), that E_{TM} is decidable.
- TM_{ETM} be its decider.
- Build decider **AnotherDecider-A_{TM}** for A_{TM}:

```
AnotherDecider-A_TM(⟨M, w⟩)
    ⟨M_w⟩ ← EmbedString(⟨M, w⟩)
    r ← TM_ETM(⟨M_w⟩).
    if r = accept then
        return reject
    // TM_ETM(⟨M_w⟩) rejected its input
    return accept
```

Emptiness is undecidable...

Consider the possible behavior of **AnotherDecider-A_{TM}** on the input $\langle M, w \rangle$.

- If TM_{ETM} accepts $\langle M_w \rangle$, then $L(M_w)$ is empty. This implies that M does not accept w. As such, **AnotherDecider-A_{TM}** rejects its input $\langle M, w \rangle$.
- If TM_{ETM} accepts $\langle M_w \rangle$, then $L(M_w)$ is not empty. This implies that M accepts w. So **AnotherDecider-A_{TM}** accepts $\langle M, w \rangle$.

Emptiness is undecidable...

Consider the possible behavior of **AnotherDecider-A_{TM}** on the input $\langle M, w \rangle$.

- If TM_{ETM} accepts $\langle M_w \rangle$, then $L(M_w)$ is empty. This implies that M does not accept w. As such, **AnotherDecider-A_{TM}** rejects its input $\langle M, w \rangle$.
- If TM_{ETM} accepts $\langle M_w \rangle$, then $L(M_w)$ is not empty. This implies that M accepts w. So **AnotherDecider-A_{TM}** accepts $\langle M, w \rangle$.

$\implies$ **AnotherDecider-A_{TM}** is decider for A_{TM}.

But A_{TM} is undecidable...

Emptiness is undecidable...

Consider the possible behavior of **AnotherDecider-A_{TM}** on the input $\langle M, w \rangle$.

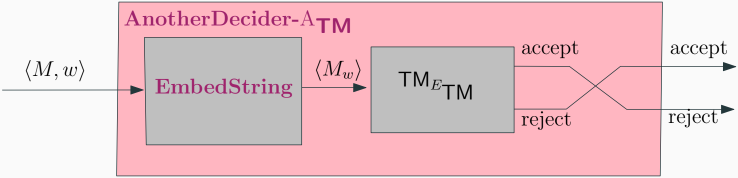
- If TM_{ETM} accepts $\langle M_w \rangle$, then $L(M_w)$ is empty. This implies that M does not accept w. As such, **AnotherDecider-A_{TM}** rejects its input $\langle M, w \rangle$.
- If TM_{ETM} accepts $\langle M_w \rangle$, then $L(M_w)$ is not empty. This implies that M accepts w. So **AnotherDecider-A_{TM}** accepts $\langle M, w \rangle$.

$\Longrightarrow$ **AnotherDecider-A_{TM}** is decider for A_{TM}.

But A_{TM} is undecidable...

...must be assumption that E_{TM} is decidable is false.

Emptiness is undecidable via diagram



AnotherDecider-A_{TM} never actually runs the code for M_w. It hands the code to a function TM_{ETM} which analyzes what the code would do if run it. So it does not matter that M_w might go into an infinite loop.

Equality

Equality is undecidable

$$EQ_{TM} = \left\{ \langle M, N \rangle \;\middle|\; M \text{ and } N \text{ are TM's and } L(M) = L(N) \right\}.$$

Lemma
The language EQ_{TM} is undecidable.

Scratch

Proof

Proof.
Suppose that we had a decider **DeciderEqual** for EQ_{TM}. Then we can build a decider for E_{TM} as follows:

TM R:

1. Input = $\langle M \rangle$
2. Include the (constant) code for a TM T that rejects all its input. We denote the string encoding T by $\langle T \rangle$.
3. Run **DeciderEqual** on $\langle M, T \rangle$.
4. If **DeciderEqual** accepts, then accept.
5. If **DeciderEqual** rejects, then reject.

□

Regularity

Many undecidable languages

- Almost any property defining a TM language induces a language which is undecidable.
- proofs all have the same basic pattern.
- Regularity language:
  $\mathrm{Regular}_{TM} = \left\{ \langle M \rangle \;\middle|\; M \text{ is a TM and } L(M) \text{ is regular} \right\}.$
- DeciderRegL: Assume TM decider for $\mathrm{Regular}_{TM}$.
- Reduction from halting requires to turn problem about deciding whether a TM M accepts w (i.e., is $w \in \mathrm{A}_{TM}$) into a problem about whether some TM accepts a regular set of strings.

Scratch

Proof continued...

- Given M and w, consider the following TM M'_w:

  TM M'_w:

  (i) Input = x
  (ii) If x has the form $a^n b^n$, halt and accept.
  (iii) Otherwise, simulate M on w.
  (iv) If the simulation accepts, then accept.
  (v) If the simulation rejects, then reject.
- ***not*** executing M'_w!
- feed string $\langle M'_w \rangle$ into **DeciderRegL**
- **EmbedRegularString**: program with input $\langle M \rangle$ and w, and outputs $\langle M'_w \rangle$, encoding the program M'_w.
- If M accepts w, then any x accepted by M'_w: $L(M'_w) = \Sigma^*$.
- If M does not accept w, then $L(M'_w) = \left\{ a^n b^n \mid n \geq 0 \right\}$.

Proof continued...

- $a^n b^n$ is not regular...
- Use **DeciderRegL** on M'_w to distinguish these two cases.
- Note - cooked M'_w to the decider at hand.
- A decider for A_{TM} as follows.

AnotherDecider-$\mathrm{A}_{TM}(\langle M, w\rangle)$
    $\langle M'_w\rangle \leftarrow$ **EmbedRegularString** $(\langle M, w\rangle)$
    $r \leftarrow$ **DeciderRegL**$(\langle M'_w\rangle)$.
    return r

- If **DeciderRegL** accepts $\implies L(M'_w)$ regular (its Σ^*)

Proof continued...

- $a^n b^n$ is not regular...
- Use **DeciderRegL** on M'_w to distinguish these two cases.
- Note - cooked M'_w to the decider at hand.
- A decider for A_{TM} as follows.

> **AnotherDecider-A_{TM}**$(\langle M, w \rangle)$
>
> $\langle M'_w \rangle \leftarrow$ **EmbedRegularString** $(\langle M, w \rangle)$
>
> $r \leftarrow$ **DeciderRegL**$(\langle M'_w \rangle)$.
>
> **return** r

- If **DeciderRegL** accepts $\implies L(M'_w)$ regular (its Σ^*) $\implies M$ accepts w. So **AnotherDecider-A_{TM}** should accept $\langle M, w \rangle$.

Proof continued...

- $a^n b^n$ is not regular...
- Use **DeciderRegL** on M'_w to distinguish these two cases.
- Note - cooked M'_w to the decider at hand.
- A decider for A_{TM} as follows.

AnotherDecider-A_{TM} $(\langle M, w\rangle)$
$\langle M'_w\rangle \leftarrow$ **EmbedRegularString** $(\langle M, w\rangle)$
$r \leftarrow$ **DeciderRegL**$(\langle M'_w\rangle)$.
return r

- If **DeciderRegL** accepts $\implies L(M'_w)$ regular (its Σ^*) $\implies M$ accepts w. So **AnotherDecider-A_{TM}** should accept $\langle M, w\rangle$.
- If **DeciderRegL** rejects $\implies L(M'_w)$ is not regular $\implies L(M'_w) = a^n b^n$

Proof continued...

- $a^n b^n$ is not regular...
- Use **DeciderRegL** on M'_w to distinguish these two cases.
- Note - cooked M'_w to the decider at hand.
- A decider for A_{TM} as follows.

AnotherDecider-A_{TM} $(\langle M, w\rangle)$
$\langle M'_w\rangle \leftarrow$ **EmbedRegularString** $(\langle M, w\rangle)$
$r \leftarrow$ **DeciderRegL**$(\langle M'_w\rangle)$.
return r

- If **DeciderRegL** accepts $\implies L(M'_w)$ regular (its Σ^*) $\implies M$ accepts w. So **AnotherDecider-A_{TM}** should accept $\langle M, w\rangle$.
- If **DeciderRegL** rejects $\implies L(M'_w)$ is not regular $\implies L(M'_w) = a^n b^n \implies M$ does not accept $w \implies$ **AnotherDecider-A_{TM}** should reject $\langle M, w\rangle$.

Rice theorem

The above proofs were somewhat repetitious...

...they imply a more general result.

Theorem (Rice's Theorem.)
Suppose that L *is a language of Turing machines; that is, each word in* L *encodes a* TM. *Furthermore, assume that the following two properties hold.*

(a) *Membership in* L *depends only on the Turing machine's language, i.e. if* $L(M) = L(N)$ *then* $\langle M \rangle \in \mathsf{L} \Leftrightarrow \langle N \rangle \in \mathsf{L}$.

(b) *The set* L *is "non-trivial," i.e.* $\mathsf{L} \neq \emptyset$ *and* L *does not contain all Turing machines.*

Then L *is a undecidable.*