Decidability

Chanathip Namprempre

Computer Science
Reed College

Recap: Recognizable languages are languages that can be recognized by a Turing machine, i.e.

Definition

L is a recognizable (recursively enumerable) language iff there is a Turing machine M such that

$$\forall w \in L, \quad M(w) \text{ halts and accept.}$$
$$\forall w \notin L, \quad M(w) \text{ does not accept.}$$

Recap: Decidable languages are languages that can be decided by a Turing machine, i.e.

Definition

L is a decidable (recursive) language iff there is a Turing machine M such that

$$\forall w \in L, \quad M(w) \text{ halts and accept.}$$
$$\forall w \notin L, \quad M(w) \text{ halts and reject.}$$

Relationship between R and RE languages

Theorem

The class of recursive languages is a subset of the class of recursively enumerable languages.

Encoding the description of a machine

Often, a TM takes the description of another machine as an input. How does this work?

For example, a DFA $D = (Q, \Sigma, \delta, q_0, F)$ can be written as a string

$$q_1, \ldots, q_{|Q|} \# \sigma_1, \ldots, \sigma_{|\Sigma|} \# \cdots$$

then encode everything in, say, binary.

We write $\langle D \rangle$ to denote the encoding of D. For convenience, it is over $\{0, 1\}^*$.

Universal Turing Machine

Consider a TM U that can do this:

On $\langle M, w \rangle$,

1. run M on w
2. If M halts on w with some output, then so do we.
3. Otherwise, loop.

This is called a Universal Turing Machine.

How could it work?

Universal Turing Machine

Assume without loss of generality that M is a single-tape TM.
The TM U uses

- tape 1: contains description of M
- tape 2: contains the configurations of M
- tape 3: contains any scratch work U needs to do to simulate M.
- tape 4: etc. (more tapes as necessary)

The TM U updates tape 2 according to what tape 1 says.

[Each time U looks at the δ function on tape 1 and do what it says on tape 2.]

Decidability

Decidable problems are problems you can solve.

Example

Input: $x \in Z$
Output: Yes if $x = x^2$. No, otherwise.

You can write a program/algorithm to compute the answer for this problem.

Having done so, you have just shown that the language $L = \{x \mid x = x^2\}$ is decidable.

Think of the solutions as algorithms but formalize them as TMs.

Today, we learn how to figure out which languages can be decided and which cannot be decided. We consider these languages.

Example

$$
\begin{aligned}
L_{DFA} &= \text{a regular language} \\
A_{DFA} &= \{\langle D, w\rangle \mid D \text{ is a DFA and } D \text{ accepts } w\} \\
A_{NFA} &= \{\langle N, w\rangle \mid N \text{ is an NFA and } N \text{ accepts } w\} \\
A_{REX} &= \{\langle R, w\rangle \mid R \text{ is a regular expression generating } w\} \\
E_{DFA} &= \{\langle D\rangle \mid D \text{ is a DFA and } L(D) = \emptyset\} \\
EQ_{DFA} &= \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}
\end{aligned}
$$

$$
\begin{aligned}
L_{CFG} &= \text{a context-free language} \\
A_{CFG} &= \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\} \\
E_{CFG} &= \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\} \\
EQ_{CFG} &= \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}
\end{aligned}
$$

L_{DFA}

Theorem

Any regular language is decidable.

Membership problem for DFAs: A_{DFA}

Consider the following language:

$$A_{DFA} = \{\langle D, w\rangle \mid D \text{ is a DFA and } D \text{ accepts } w\} .$$

A_{DFA} is decidable.

Q: How to prove this?
A: Show that there is a TM deciding this language. So prove by construction.

Theorem

$A_{DFA} = \{\langle D, w\rangle \mid D \textit{ is a DFA and } D \textit{ accepts } w\}$ *is decidable.*

Proof.

M_{dfa} = "On input $\langle D, w\rangle$ where D is a DFA and w is a string,

1. Simulate D on input w.
2. If the simulation ends in an accept state, accept. Otherwise, reject."

□

Theorem

$A_{NFA} = \{\langle N, w\rangle \mid N \text{ is a NFA and } N \text{ accepts } w\}$ *is decidable.*

Proof.

M_{nfa} = "On input $\langle N, w\rangle$ where N is a NFA and w is a string,

1. Simulate N on input w.
2. If the simulation ends in an accept state, accept. Otherwise, reject."

□

Notice that this machine is a non-deterministic Turing machine!

We know that non-deterministic TMs have equal power to deterministic TMs. So this is ok.

Theorem

$A_{NFA} = \{\langle N, w\rangle \mid N \text{ is a NFA and } N \text{ accepts } w\}$ *is decidable.*

Proof.

M_{nfa} = "On input $\langle N, w\rangle$ where N is a NFA and w is a string,

1. Simulate N on input w.
2. If the simulation ends in an accept state, accept. Otherwise, reject."

□

Notice that this machine is a non-deterministic Turing machine!

We know that non-deterministic TMs have equal power to deterministic TMs. So this is ok.

Theorem

$A_{NFA} = \{\langle N, w\rangle \mid N \textit{ is an NFA and } N \textit{ accepts } w\}$ *is decidable.*

This is also ok.

Proof.

M_{nfa} = "On input $\langle N, w\rangle$ where N is a NFA and w is a string,

1. Convert N into an equivalent DFA D.
2. Simulate D on input w.
3. If the simulation ends in an accept state, accept. Otherwise, reject."

□

Theorem

$$A_{REX} = \{\langle R, w\rangle \mid R \textit{ is a regular expression that generates the string } w\} \textit{ is decidable.}$$

Proof.

M_{rex} = "On input $\langle R, w\rangle$ where R is a regular expression and w is a string,

1. Convert R to an equivalent DFA D.
2. Run M_{dfa} on $\langle D, w\rangle$.
3. If M_{dfa} accepts, accept. Otherwise, reject."

□

Emptiness for DFAs: E_{DFA}

Theorem

$E_{DFA} = \{\langle D \rangle \mid D \text{ is a DFA and } L(D) = \emptyset\}$ *is decidable.*

First try:
$M^{\emptyset}_{dfa}$ = "On input $\langle D \rangle$ where D is a DFA,

1. For each string w from the list $w_1, w_2, \ldots$ of all possible strings,
   1. Run D on w
   2. If D accepts w, halt and reject.
2. If D never accepts, accept."

This doesn't work! WHY?

Emptiness for DFAs: E_{DFA}

Theorem

$E_{DFA} = \{\langle D \rangle \mid D \text{ is a DFA and } L(D) = \emptyset\}$ *is decidable.*

First try:
$M^{\emptyset}_{dfa}$ = "On input $\langle D \rangle$ where D is a DFA,

1. For each string w from the list $w_1, w_2, \ldots$ of all possible strings,
   1. Run D on w
   2. If D accepts w, halt and reject.
2. If D never accepts, accept."

This doesn't work! WHY?

Emptiness for DFAs: E_{DFA}

Theorem

$E_{DFA} = \{\langle D \rangle \mid D \text{ is a DFA and } L(D) = \emptyset\}$ *is decidable.*

Idea: Figure out if the accept states of D can ever be reached. If so, D accepts something.

Proof.

$M^{\emptyset}_{dfa}$ = "On input $\langle D \rangle$ where D is a DFA,

1. Mark the start state of D.
2. Repeat until no new states get marked:
   Mark any state that has a transition coming into it from any state that is already marked.
3. If no accept state is marked, accept. Otherwise, reject."

□

Equivalence for DFAs: EQ_{DFA}

Theorem

$EQ_{DFA} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$ *is decidable.*

Idea: Notice that
$L(A) = L(B)$ iff both $L(A) \cap \overline{L(B)}$ and $\overline{L(A)} \cap L(B)$ are empty sets.

Proof.

$M^{=}_{dfa}$ = "On input $\langle A, B\rangle$ where A and B are DFAs,

1. Construct a DFA C for $L(A) \cap \overline{L(B)}$ and D for $\overline{L(A)} \cap L(B)$ as discussed.
2. Run $M^{\emptyset}_{dfa}$ on $\langle C\rangle$ and $\langle D\rangle$.
3. If $M^{\emptyset}_{dfa}$ accepts both languages, accept. Otherwise, reject."

□

Equivalence for DFAs: EQ_{DFA}

Theorem

$EQ_{DFA} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$ *is decidable.*

Idea: Notice that
$L(A) = L(B)$ iff both $L(A) \cap \overline{L(B)}$ and $\overline{L(A)} \cap L(B)$ are empty sets.

Proof.

$M^{=}_{dfa}$ = "On input $\langle A, B\rangle$ where A and B are DFAs,

1. Construct a DFA C for $L(A) \cap \overline{L(B)}$ and D for $\overline{L(A)} \cap L(B)$ as discussed.
2. Run $M^{\emptyset}_{dfa}$ on $\langle C\rangle$ and $\langle D\rangle$.
3. If $M^{\emptyset}_{dfa}$ accepts both languages, accept. Otherwise, reject."

□

Equivalence for DFAs: EQ_{DFA}

Theorem

$EQ_{DFA} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$ *is decidable.*

Idea: Notice that
$L(A) = L(B)$ iff both $L(A) \cap \overline{L(B)}$ and $\overline{L(A)} \cap L(B)$ are empty sets.

Proof.

$M^{=}_{dfa}$ = "On input $\langle A, B\rangle$ where A and B are DFAs,

1. Construct a DFA C for $L(A) \cap \overline{L(B)}$ and D for $\overline{L(A)} \cap L(B)$ as discussed.
2. Run $M^{\emptyset}_{dfa}$ on $\langle C\rangle$ and $\langle D\rangle$.
3. If $M^{\emptyset}_{dfa}$ accepts both languages, accept. Otherwise, reject."

□

Membership problem for CFGs: A_{CFG}

Theorem

$A_{CFG} = \{\langle G, w\rangle \mid G \textit{ is a CFG that generates string } w\}$ *is decidable.*

Idea: We'll use the following fact:

For any grammar G in Chomsky normal form, any derivation of any string w of length n has $2n - 1$ steps.

Proof.

M_{cfg} = "On input $\langle G, w\rangle$ where G is a grammar and w is a string,

1. Convert G to an equivalent grammar in Chomsky normal form.
2. Let n be the length of w. List all derivations with $2n - 1$ steps.
3. If any of these derivations generate w, accept. Otherwise, reject."

Membership problem for CFGs: A_{CFG}

Theorem

$A_{CFG} = \{\langle G, w\rangle \mid G \textit{ is a CFG that generates string } w\}$ *is decidable.*

Idea: We'll use the following fact:

For any grammar G in Chomsky normal form, any derivation of any string w of length n has $2n - 1$ steps.

Proof.

M_{cfg} = "On input $\langle G, w\rangle$ where G is a grammar and w is a string,

1. Convert G to an equivalent grammar in Chomsky normal form.
2. Let n be the length of w. List all derivations with $2n - 1$ steps.
3. If any of these derivations generate w, accept. Otherwise, reject."

L_{CFG}

Theorem

Every context-free language is decidable.

Idea: If L is a CFL, there is a grammar G for it. So we simulate M_{cfg} on it and an input string.

Proof.

Let L be a CFL, and let G be the CFG for L. We construct a decider M_G for L as follows:

$M_G =$ "On input w,

1. Run M_{cfg} on $\langle G, w \rangle$
2. If M_{cfg} accepts, accept. Otherwise, reject."

□

Emptiness for CFGs: E_{CFG}

Theorem

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

Idea: Mark symbols (variables and terminals) starting with terminals and propagating up to the variables that can derive the marked terminals. Keep doing this and reject if the start symbol is eventually marked (which means the grammar can generate something).

Emptiness for CFGs: E_{CFG}

Theorem

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

Proof.

$M^{\emptyset}_{cfg}$ = "On input $\langle G \rangle$ where G is a grammar,

1. Mark all terminal symbols and ε in G.
2. Repeat until no new variables get marked:
   Mark any variable A where G has a rule $A \to U_1 U_2 \dots U_k$ and each symbol $U_1, \dots, U_k$ has been marked.
3. If the start symbol is not marked, accept. Otherwise, reject."

□

Equivalence for CFGs: EQ_{CFG}

Wrong Claim

$EQ_{CFG} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$ is decidable.

(Wrong) Idea: Try to do something similar to the case of EQ_{DFA}.

This does not work because the class of CFLs is not closed under intersection or complementation. (Can you prove this?)

In fact, it turns out that EQ_{CFG} is not decidable.

Co-Recursively Enumerable Languages

Definition

A language L is co-recursively enumerable if and only if $\overline{L}$ is recursively enumerable.

We can define 3 classes of languages as follows:

- $\mathcal{R} = \{L \mid L \text{ is recursive } \}$
- $\mathcal{RE} = \{L \mid L \text{ is recursively enumerable } \}$
- $co\mathcal{RE} = \{L \mid L \text{ is co-recursively enumerable } \}$

Closure properties for recursive languages

Theorem

Let L be a language.

1. *If L is recursive, then so is $\overline{L}$.*
2. *If L_1, L_2 are recursive, then so is $L_1 \cup L_2$.*
3. *If L_1, L_2 are recursive, then so is $L_1 \cap L_2$.*
4. *If L_1, L_2 are r.e., then so is $L_1 \cup L_2$.*
5. *If L_1, L_2 are r.e., then so is $L_1 \cap L_2$.*

Notice: The class of r.e. languages is not closed under complementation.

More properties of recursive languages

Theorem

A language L is recursive if and only if L is both r.e. and co-r.e.

So $\mathcal{R} = \mathcal{RE} \cap co\mathcal{RE}$.

Summary: language-deciding TMs we know

TM	Language that it decides
M_{dfa}	$A_{DFA} = \{\langle D, w\rangle \mid D \text{ is a DFA and } D \text{ accepts } w\}$
M_{nfa}	$A_{DFA} = \{\langle N, w\rangle \mid N \text{ is an NFA and } N \text{ accepts } w\}$
$M^{\emptyset}_{dfa}$	$E_{DFA} = \{\langle D\rangle \mid D \text{ is an DFA and } L(D) = \emptyset\}$
$M^{=}_{dfa}$	$EQ_{DFA} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$
M_{cfg}	$A_{CFG} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$
$M^{\emptyset}_{cfg}$	$E_{CFG} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$

Summary: transform-computing TMs we know

TM	Input	Output	Property
T^{comp}_{dfa}	$\langle D_1 \rangle$ [D_1 is a DFA]	$\langle D_2 \rangle$ [D_2 is a DFA]	$L(D_2) = \overline{L(D_1)}$
$T^{\cap}_{dfa}$	$\langle D_1, D_2 \rangle$ [D_1, D_2 are DFAs]	$\langle D \rangle$ [D is a DFA]	$L(D) = L(D_1) \cap L(D_2)$
$T^{\cup}_{dfa}$	$\langle D_1, D_2 \rangle$ [D_1, D_2 are DFAs]	$\langle D \rangle$ [D is a DFA]	$L(D) = L(D_1) \cup L(D_2)$
$T^{\bullet}_{dfa}$	$\langle D_1, D_2 \rangle$ [D_1, D_2 are DFAs]	$\langle D \rangle$ [D is a DFA]	$L(D) = L(D_1) \cdot L(D_2)$
T^{*}_{dfa}	$\langle D_1 \rangle$ [D_1 is a DFA]	$\langle D_2 \rangle$ [D_2 is a DFA]	$L(D_2) = L(D_1)^*$
$T_{nfa \to dfa}$	$\langle N \rangle$ [N is an NFA]	$\langle D \rangle$ [D is a DFA]	$L(D) = L(N)$

Summary: transform-computing TMs we know

TM	Input	Output	Property
$T^{\cup}_{cfg}$	$\langle G_1, G_2 \rangle$ [G_1, G_2 are CFGs]	$\langle G \rangle$ [G is a CFG]	$L(G) = L(G_1) \cup L(G_2)$
$T^{\bullet}_{cfg}$	$\langle G_1, G_2 \rangle$ [G_1, G_2 are CFGs]	$\langle G \rangle$ [G is a CFG]	$L(G) = L(G_1) \cdot L(G_2)$
T^{*}_{cfg}	$\langle G_1 \rangle$ [G_1 is a CFG]	$\langle G_2 \rangle$ [G_2 is a CFG]	$L(G_2) = L(G_1)^*$
$T_{dfa \to cfg}$	$\langle D \rangle$ [D is a DFA]	$\langle G \rangle$ [G is a CFG]	$L(G) = L(D)$
$T_{cfg \to cnf}$	$\langle G \rangle$ [G is a CFG]	$\langle G' \rangle$ [G' is a CFG in CNF]	$L(G) = L(G')$