Decidability

Question: Why study decidability?

- Your boss orders you to solve Hilbert's 10th, or else.
- Good for your imagination.
- Some of the most beautiful and important mathematics of the 20th century, and you can actually understand it! (YMMV).

Examples of Decidable Languages

Finite automata problems can be reformulated as languages.

Does DFA B accept input string w?

Consider the language:

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

These problems are equivalent:

- B accepts w
- $\langle B, w\rangle \in A_{\text{DFA}}$

Theorem

Theorem: A_{DFA} is a decidable language.

On input $\langle B, w \rangle$, where B is a DFA and w a string:

1. Simulate B on input w

2. if simulation ends in accepting state, *accept*, otherwise *reject*.

Note

- "where" clause means scan and check condition.
- B represented by a list of $(Q, \Sigma, \delta, q_0, F)$.
- simulation straightforward

Theorem

Does an NFA accept a string?

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

Theorem: A_{NFA} is a decidable language.

On input $\langle B, w\rangle$, where B is an NFA and w a string:

1. Convert. NFA B into equivalent C
2. Run previous TM on input $\langle C, w\rangle$.
3. if that TM accepts, accept, otherwise reject.

Note use of subroutine.

Theorem

Does a regular expression generate a string?

$$A_{\text{REX}} = \{\langle R, w\rangle | R \text{ is a regular expression that generates } w\}$$

Theorem: A_{REX} is a decidable language.

On input $\langle R, w\rangle$, where R is a regular expression and w a string:

1. Convert regular expression R into equivalent C.
2. Run earlier TM on input $\langle C, w\rangle$.
3. If that TM accepts, accept, otherwise reject.

Theorem

Does a DFA accept the empty language?

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

Theorem: E_{DFA} is a decidable language.

On input $\langle A\rangle$, where A is a DFA:

Mark. the start state of A.

2. Repeat until no new states are marked:

3. Mark any state that has a transition coming into it from any already marked state.

4. If no accept state is marked, accept, otherwise reject.

This TM actually just tests whether any accepting state is reachable from initial state.

Theorem

$$EQ_{\mathsf{DFA}} = \{\langle A, B\rangle | A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$$

Theorem: EQ_{DFA} is a decidable language.

We construct a new DFA C from A and B, such that C accepts only string accepted by A or by B, but not both.

$$L(C) = \left(L(A) \cap \overline{L(B)}\right) \cup \left(\overline{L(A)} \cap L(B)\right).$$

- *symmetric difference* of $L(A)$ and $L(B)$
- use construction used for regular language theorems
- this construction can be expressed as a TM

Theorem (continued)

Theorem: EQ_{DFA} is a decidable language.

On input $\langle A.B \rangle$, where A, B are DFAs:

1. Construct DFA C as described.

2. Run previous TM on input $\langle C \rangle$.

3. If that TM accepts, accept, otherwise reject.

Theorem

Does a CFG generate a particular string?

$$A_{\text{CFG}} = \{\langle G, w\rangle | g \text{ is a CFG that generates string } w\}$$

Theorem: A_{CFG} is a decidable language.

Theorem (continued)

Does a CFG generate a particular string?

Idea: Design a TM M to try all derivations.

Problem: M accepts but does not decide. Why?

Theorem (continued)

Lemma: If G is in Chomsky normal form and $|w| = n$, then w has a derivation of length $2n - 1$ or less.

We won't prove this. Go ahead — try it at home!

Idea:

- convert G to Chomsky normal form
- need only consider finite number of derivations

Theorem (continued)

Theorem: A_{CFG} is a decidable language.

On input $\langle G, w \rangle$, where G is a grammar and w a string,

1. Convert G to Chomsky normal form.

2. List all derivations with $2n - 1$ steps, were $n = |w|$.

3. If any generates w, accept, otherwise reject.

Remarks:

- related to problem of compiling
- would you want to use this algorithm?
- every theorem about CFLs is also about PDAs.

Theorem

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

Theorem: E_{CFG} is a decidable language.

Bad Idea: We can test whether $w \in L(G)$ for any string w, so just try it for each w. (Criticize this.)

Good Idea: Can the start variable generate a string of terminals?

Better Idea: Can a particular variable generate a string of terminals?

Theorem (continued)

Theorem: E_{CFG} is a decidable language.

On input $\langle G \rangle$ where G is a CFG:

1. Mark all terminal symbols in G.

2. Repeat until no new variables become marked:

3. Mark any A where

$$A \to U_1 U_2 \ldots U_k$$

and each U_i has already been marked.

4. If start symbol marked, accept, otherwise reject.

When Are Two CFGs equivalent?

$$EQ_{\mathsf{CFG}} = \{\langle G, H\rangle | G, H \text{ are CFGs and } L(G) = L(H)\}$$

We did this already for EQ_{DFA}!

We constructed C from A and B:

$$L(C) = \left(L(A) \cap \overline{L(B)}\right) \cup \left(\overline{L(A)} \cap L(B)\right).$$

and tested whether $L(C)$ is empty.

Danger! Danger! Danger!

When Are Two CFGs equivalent?

The class of context-free languages is *not* closed under complementation or intersection.

Fact: E_{CFG} is not a decidable language.

We are not yet prepared to prove this remarkable fact.

Last Word on Context-Free Languages

Theorem: Every context-free language is decidable.

Bad Idea:

- convert a PDA for A directly into a TM
- not hard why?
- non-deterministic PDA yields non-deterministic TM
- but non-deterministic TM same as deterministic TM

Problem:

- some branch of PDA might run forever
- some branch of non-deterministic TM might run forever
- deterministic TM accepts, but does not decide!

Last Word on Context-Free Languages

Let

- S be the TM that decides A_{CFG}, and
- G be a CFG for A.

On input w

1. run TM S on input $\langle G, w \rangle$

2. if S accepts, accept, otherwise reject.

Languages