Decidability

April 1, 2020
In the last lecture...

- Discussed the definition of algorithm
  - Church-Turing thesis
- Established terminology for describing TMs
  - Format and notation:
    - Encoding in terms of strings
- Began looking at an example
Example (from last lecture)

- Let $A$ be the language consisting of all strings representing undirected graphs that are connected. That is,
  
  $A = \{<G> \mid G \text{ is a connected undirected graph}\}$

- The following (next slide) is a high-level description of a TM $M$ that decides $A$
$M = \text{"On input }\langle G \rangle\text{, the encoding of a graph } G:\$ 

1. Select 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 nodes of $G$ to determine whether they all are marked. If they are, accept; otherwise, reject."
Just a bit of implementation detail on M...

Some details of M...

- Input check:
  - node list (distinct elements)
  - edge list (pairs drawn from node list)
- Stags 1 -- 3:
  - Markings
- Stage 4:
  - Scanning

$$G = \begin{array}{c}
\text{1} \\
\text{3} \\
\text{2} \\
\text{4}
\end{array}$$

$$\langle G \rangle = (1, 2, 3, 4) \times ((1, 2), (2, 3), (3, 1), (1, 4))$$

Encoding
Today’s lecture: Decidability

- Our objective is to explore the limits of algorithmic solvability
  - Certain problems can be solved algorithmically, and others cannot

- Why bother study unsolvability?
  1. Practice
     → (Re)formulation of problem
  2. Perspective
     → A glimpse of the unsolvable may stimulate imagination
Decidable languages

- Will look at decidable problems concerning
  - Finite automata
    - Acceptance
    - Emptiness
    - Equivalence
  - Context-free grammars
    - Generation
    - Emptiness

- Will cover results on FA today, and those on CFG next lecture
1) Finite Automata: Acceptance Problem (DFA)

Let:

$$A_{DFA} = \{<B,w> \mid B \text{ is a DFA that accepts input string } w\}$$

(Note: we choose to represent computation problems by languages. In the case above, the problem of testing whether a DFA $B$ accepts an input $w$ is the same as the problems of testing whether $<B,w>$ is a member of the language $A_{DFA}$)

Theorem: $A_{DFA}$ is a decidable language
Proof

We simply need to present a TM $M$ that decides $A_{\text{DFA}}$

$M = \text{“On input } \langle B, w \rangle, \text{ where } B \text{ is a DFA and } w \text{ is a string:}
\begin{enumerate}
\item Simulate } B \text{ on input } w.
\item If the simulation ends in an accept state, accept. If it ends in a nonaccepting state, reject.”
\end{enumerate}$

A few implementation details…

• $\langle B, w \rangle$
  • A reasonable representation of $B$ may be its 5 components $(Q, \Sigma, \delta, q_0 \text{ and } F)$

• Simulation
  • $M$ may do this directly
2) Finite Automata: Acceptance Problem (NFA)

We can prove a similar theorem for NFA

Let:

$$A_{\text{NFA}} = \{<B,w> \mid B \text{ is an NFA that represents input string } w\}$$

Theorem:

$$A_{\text{NFA}}$$ is a decidable language
We present a TM $N$ that decides $A_{NFA}$.

Instead of making $N$ simulate an NFA, we will make it use $M$ (the DFA) as a subroutine.

$N = \text{"On input } \langle B, w \rangle, \text{ where } B \text{ is an NFA and } w \text{ is a string:}\n\begin{enumerate}
\item Convert NFA $B$ to an equivalent DFA $C$, using the procedure for this conversion given in Theorem 1.39.
\item Run TM $M$ from Theorem 4.1 on input $\langle C, w \rangle$.
\item If $M$ accepts, accept; otherwise, reject."
\end{enumerate}$

**Thm 1.39:** every NFA has an equivalent DFA

**Thm 4.1:** $A_{DFA}$ is decidable
3) Regular expression: Generation

We can prove similar result for determining whether a regular expression generates a given string.

Let:

\[ A_{REX} = \{<R,w> | R \text{ is a regular expression that generates string } w\} \]

Theorem:

\[ A_{REX} \text{ is a decidable language} \]
Proof

The following TM $P$ decides $A_{\text{REX}}$

$P = \text{“On input } \langle R, w \rangle, \text{ where } R \text{ is a regular expression and } w \text{ is a string:} \$

1. Convert regular expression $R$ to an equivalent NFA $A$ by using the procedure for this conversion given in Theorem 1.54.
2. Run TM $N$ on input $\langle A, w \rangle$.
3. If $N$ accepts, accept; if $N$ rejects, reject.”

**Thm 1.54:** a language is regular iff some regular expression describes it
What did we observe so far?

The previous three results illustrate that, for decidability purposes, it is equivalent to present the TM with a DFA, an NFA or a regular expression because the machine can convert one form of encoding to another.

Next we see two different kinds of problems concerning FA:

- Emptiness testing
- Equivalence of two DFAs
4) Finite Automata: Emptiness

Let:

\[ E_{DFA} = \{<A> \mid A \text{ is a DFA and } L(A) = \emptyset\} \]

Theorem:

\[ E_{DFA} \text{ is a decidable language} \]
Proof

- A DFA accepts some string iff reaching an accept state from the start state by traveling along the arrows of the DFA is possible.
- To test this condition, we can design a TM $T$ that uses a marking algorithm similar to the example on *connected graphs* we saw at the beginning of this lecture.

\[
T = \text{“On input } \langle A \rangle, \text{ where } A \text{ is a DFA:}
\]

1. Mark the start state of $A$.
2. Repeat until no new states get marked:
3. Mark any state that has a transition coming into it from any state that is already marked.
4. If no accept state is marked, *accept*; otherwise, *reject*.”
Let:

\[ EQ_{DFA} = \{<A,B> \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\} \]

Theorem:

\[ EQ_{DFA} \text{ is a decidable language} \]
Proof

To prove this theorem, we use the previous theorem (emptiness).

We construct a new DFA $C$ from $A$ and $B$, where $C$ accepts only those strings that are accepted by either $A$ or $B$ but not by both.

Thus if $A$ and $B$ recognize the same language, $C$ will accept nothing.

The language $L(C)$ is the symmetric difference between $L(A)$ and $L(B)$:

$$L(C) = (L(A) \cap L(B))^c \cup (L(A)^c \cap L(B))$$

$F = \text{“On input } \langle A, B \rangle, \text{where } A \text{ and } B \text{ are DFAs:} \quad$

1. Construct DFA $C$ as described.
2. Run TM $T$ from Theorem 4.4 on input $\langle C \rangle$.
3. If $T$ accepts, accept. If $T$ rejects, reject.”

**Thm 4.4:** $E_{DFA}$ is a decidable language