Reading Assignment: Sipser Chapter 3.1, 4.2

4.1 covers algorithms for decidable problems about DFA, NFA, RegExp, CFG, and PDAs, e.g. slides 17 & 18 below. I’ve talked about most of this in class at one point or another, but skimming 4.1 would probably be a good review.
By definition, no transitions out of $q_{acc}$, $q_{rej}$; $M$ halts if (and only if) it reaches either $M$ loops if it never halts ("loop" might suggest "simple", but non-halting computations may of course be arbitrarily complex)

$M$ accepts if it reaches $q_{acc}$. $M$ rejects by halting in $q_{rej}$ or by looping

The language recognized by $M$:
$L(M) = \{ w \in \Sigma^* \mid M \text{ accepts } w \}$

$L$ is Turing recognizable if $\exists TM \ M$ s.t. $L = L(M)$
$L$ is Turing decidable if, furthermore, $M$ halts on all inputs

A key distinction!
Church-Turing Thesis

TM’s formally capture the intuitive notion of “algorithmically solvable”

Not provable, since “intuitive” is necessarily fuzzy.

But, give support for it by showing that
(a) other intuitively appealing (but formally defined) models are precisely equivalent, and
(b) models that are provably different are unappealing, either because they are too weak (e.g., DFA’s) or too powerful (e.g., a computer with a “solve-the-halting-problem” instruction).
Nondeterministic Turing Machines

\[ \delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L,R\}) \]

Accept if any path leads to \( q_{\text{accept}} \); reject otherwise, (i.e., all halting paths lead to \( q_{\text{reject}} \))

Simulating an NTM

Key issue: avoid getting lost on \( \infty \) path

Key Idea: breadth-first search

Tree arity \( \leq |Q| \times |\Gamma| \times |\{L,R\}| \) (3 in example)

Encoding things

\[ G = (V, \Sigma, R, S) \quad \text{or} \quad <G> = ((S,A,B,...),(a,b,...),(S \rightarrow aA, S \rightarrow b, A \rightarrow cAb,...),S) \]

\[ \Sigma = ? \]

Decidability

Recall: \( L \) decidable means there is a TM recognizing \( L \) that always halts.

Example:

“The acceptance problem for DFAs”

\( A_{\text{DFA}} = \{ <D,w> | D \text{ is a DFA } \& w \in L(D) \} \)
Some Decidable Languages

The following are decidable:

$A_{DFA} = \{ <D,w> | D \text{ is a DFA \& } w \in L(D) \}$

pf: simulate D on w

$A_{NFA} = \{ <N,w> | N \text{ is an NFA \& } w \in L(N) \}$

pf: convert N to a DFA, then use previous as a subroutine

$A_{REX} = \{ <R,w> | R \text{ is a regular expr \& } w \in L(R) \}$

pf: convert R to an NFA, then use previous as a subroutine

$\text{EMPTY}_{DFA} = \{ <D> | D \text{ is a DFA and } L(D) = \emptyset \}$

pf: is there no path from start state to any final state?

$\text{EQ}_{DFA} = \{ <A,B> | A \text{ & B are DFAs s.t. } L(A) = L(B) \}$

pf: equal iff $L(A) \oplus L(B) = \emptyset$, and $x \oplus y = (x^c \cap y^c) \cup (x \cap y^c)$, and regular sets are closed under $\cup$, $\cap$, complement

$A_{CFG} = \{ <G,w> | \ldots \}$

pf: see book

$\text{EMPTY}_{CFG} = \{ <G> | \ldots \}$

pf: see book

$\text{EQ}_{CFG} = \{ <A,B> | A \text{ & B are CFGs s.t. } L(A) = L(B) \}$

This is NOT decidable

\[ \text{FIGURE 4.10} \]

The relationship among classes of languages
The Acceptance Problem for TMs

\[ A_{TM} = \{ <M, w> \mid M \text{ is a TM } \& w \in L(M) \} \]

Theorem: \( A_{TM} \) is Turing recognizable

Proof: It is recognized by a TM \( U \) that, on input \( <M, w> \), simulates \( M \) on \( w \) step by step. \( U \) accepts iff \( M \) does. \( \square \)

\( U \) is called a Universal Turing Machine

(Ancestor of the stored-program computer)

Note that \( U \) is a recognizer, not a decider.

The Set of Languages in \( \Sigma^* \) is Uncountable

Suppose they were

\[ \begin{array}{cccccccc}
L & 0 & 0 & 0 & 0 & 0 & 0 \\
L & 1 & 1 & 1 & 1 & 1 & 1 \\
L & 0 & 1 & 0 & 1 & 0 & 1 \\
L & 0 & 1 & 0 & 0 & 0 & 0 \\
L & 1 & 1 & 1 & 0 & 0 & 0 \\
L & 1 & 1 & 1 & 1 & 0 & 1 \\
L & 1 & 0 & 1 & 1 & 1 & 0 \\
\end{array} \]

Then \( L \) is not in the list

Contradiction

“Most” languages are neither Turing recognizable nor Turing decidable

Proof idea:

“\( \langle \rangle \)” maps TMs into \( \Sigma^* \), a countable set, so the set of TMs, and hence of Turing recognizable languages is also countable; Turing decidable is a subset of Turing recognizable, so also countable. But by the previous result, the set of all languages is uncountable.
A specific non-Turing-recognizable language

Let $M_i$ be the TM encoded by $w_i$, i.e. $\langle M_i \rangle = w_i$.

$M_i$ = some default machine, if $w_i$ is an illegal code.

$i, j$ entry $= 1$ if $M_i$ accepts $w_j$.

$LD = \{ w_i | i, i \text{ entry} = 0 \}$

Then $LD$ is not recognized by any TM.

Theorem: The class of Turing recognizable languages is not closed under complementation.

Proof:

The complement of $D$ is Turing recognizable:

On input $w_i$, run $\langle M_i \rangle$ on $w_i$ ($= \langle M_i \rangle$); accept if it does. E.g., use a universal TM on input $\langle M_i, \langle M_i \rangle \rangle$.

E.g., in previous example, $D^c$ might be $L(M_6)$.

The Acceptance Problem for TMs

$ATM = \{ \langle M, w \rangle | M \text{ is a TM} \text{ & } w \in L(M) \}$

Theorem: $ATM$ is Turing recognizable

Pf: It is recognized by a TM $U$ that, on input $\langle M, w \rangle$, simulates $M$ on $w$ step by step. $U$ accepts iff $M$ does. $\square$

$U$ is called a Universal Turing Machine

(Ancestor of the stored-program computer)

Note that $U$ is a recognizer, not a decider.
A\_{TM} is Undecidable

\[ A_{TM} = \{ <M,w> \mid M \text{ is a TM} & w \in L(M) \} \]

Suppose it’s decidable, say by TM H. Build a new TM D:

- “on input <M> (a TM), run H on <M,<M>>; when it halts, halt & do the opposite, i.e. accept if H rejects and vice versa”

D accepts <M> iff H rejects <M,<M>> (by construction)
D accepts <M> iff M rejects <M> (H recognizes A\_{TM})

D accepts <D> iff D rejects <D> (special case)

Contradiction!

Decidable \nsubseteq \nexists Recognizable

Decidable = Rec \cap co-Rec

L decidable iff both L & L^c are recognizable

Pf: (\Rightarrow) on any given input, dovetail (run in parallel) a recognizer for L with one for L^c; one or the other must halt & accept, so you can halt & accept/reject appropriately.

(\Rightarrow): from above, decidable languages are closed under complement (flip acc/rej)
The Halting Problem

\[ \text{HALT}_{\text{TM}} = \{ <M, w> \mid \text{TM } M \text{ halts on input } w \} \]

Theorem: The halting problem is undecidable

Proof:
Suppose TM \( R \) decides \( \text{HALT}_{\text{TM}} \).
Consider \( S \):

- On input \( <M, w> \), run \( R \) on it. If it rejects, halt & reject; if it accepts, run \( M \) on \( w \); accept/reject as it does.

Then \( S \) decides \( A_{\text{TM}} \), which is impossible. \( R \) can’t exist.

Programs vs TMs

Fix \( \Sigma = \) printable ASCII
Programming language with ints, strings & function calls
“Computable function” = always returns something
“Decider” = computable function always returning 0 / 1
“Acceptor” = accept if return 1; reject if \( \neq 1 \) or loop

\( A_{\text{Proc}} = \{ <P, w> \mid \text{program } P \text{ returns 1 on input } w \} \)

\( \text{HALT}_{\text{Proc}} = \{ <P, w> \mid \text{prog } P \text{ returns something on } w \} \)

Many Undecidable Problems

About Turing Machines

\( \text{HALT}_{\text{TM}} \ \text{EQ}_{\text{TM}} \ \text{EMPTY}_{\text{TM}} \ \text{REGULAR}_{\text{TM}} \ldots \)

About programs
Ditto! And: array-out-of-bounds, unreachability, loop termination, assertion-checking, correctness, ...

About Other Things
\( \text{EMPTY}_{\text{LBA}} \ \text{ALL}_{\text{CFG}} \ \text{EQ}_{\text{CFG}} \ \text{PCP} \ \text{DiophantineEqns} \ldots \)
Summary

Turing Machines
   A simple model of “mechanical computation”
Church-Turing Thesis
   All “reasonable” models are alike in capturing the intuitive notion of “mechanically computable”
Decidable/Recognizable – Key distinction: Does it halt
Undecidability – counting, diagonalization, reduction
   \( A_{\text{TM}} = \{ <M,w> \mid \text{TM } M \text{ accepts } w \} \)
   \( \text{HALT}_{\text{TM}} = \{ <M,w> \mid \text{TM } M \text{ halts on } w \} \)

Want More?

Check out CSE 431
“Intro Computability & Complexity”