Decidable Problems and Recursive Languages

We examine the computation of TMs that do and don’t halt.
A language is recursively enumerable (r.e.) if it is the set of strings accepted by some TM.

A language is recursive if it is the set of strings accepted by some TM that halts on every input.

For example, any regular language is recursive.

We will show later that there are many r.e. languages that are not recursive, and many languages that are not even r.e.
**Fact.** (a) The set of r.e. languages is closed under union and intersection.
(b) The set of recursive languages is closed under union and intersection.

*We prove (a) for union.*

Say languages $L_1$ and $L_2$ are r.e., accepted by TMs $M_1$ and $M_2$. A TM for $L_1 \cup L_2$ simply runs $M_1$ and $M_2$ *in parallel*. The input is in the union exactly when at least one machine halts and accepts.
Theorem. A language is recursive if and only if both it and its complement are r.e.

• If \( L \) is recursive, then so is its complement (interchange \( h_\alpha \) and \( h_r \)).

• Assume both \( L \) and \( \bar{L} \) are r.e.; that is, they have TMs. Then run the two TMs in parallel. At least one will halt, and that gives us the answer.
A printer-TM is TM with an added printer-tape. The printer-TM writes strings on the printer-tape (separated by $\Delta$); once written a string is not altered.

**Theorem.** A language is r.e. if and only if some printer-TM outputs precisely those strings.

The proof is in two constructions...
Armed with printer-TM $M$ for language $L$, we build standard TM $N$: On input $x$, TM $N$ runs $M$ and monitors $M$; if $N$ ever finds $x$ on the printer-tape, then it accepts. So $N$ accepts strings in $L$, and does not halt otherwise.
Armed with standard TM $N$ for $L$, we build printer-TM $M$. The idea is to run $N$ on every possible string in parallel—an infinite number of tasks!

The printer-TM works in rounds. In round $i$, $M$ starting from scratch, generates the first $i$ strings lexicographically (in dictionary order), runs $N$ on each for $i$ steps, and outputs any string that is accepted. Eventually, every string in $L(N)$ will be generated and $L$ run for long enough, and will appear in the output.
Given a question, we build a language by taking all the instances of the question where the answer is yes and converting the instance to a string.

A yes/no question is **decidable** if the associated language is recursive. This is equivalent to finding a program that always halts and answers the question correctly.
Encodings

We assume a standard *encoding* of a machine. E.g., the format you would use to describe it if you wrote a Java program that simulated that type. The actual encoding does not matter, as long as it is fixed and one can easily get between $A$ and its encoding $\langle A \rangle$.

We use $\langle A, B \rangle$ to denote the encoding of pair \{A, B\}.
Questions about Regular Languages

It is easy to answer questions about regular languages:

Recursive are:

a) The acceptance problem: \( A_{fa} = \{ \langle M, w \rangle : M \text{ is FA that accepts } w \} \).

b) The emptiness problem: \( Empty_{fa} = \{ \langle M \rangle : M \text{ is FA with empty language} \} \).

c) The equivalence problem: \( EQ_{fa} = \{ \langle A, B \rangle : A \text{ and } B \text{ are FAs with } L(A) = L(B) \} \).

We prove part (c).
Proof that Equivalence is Decidable

We build a TM that first checks the input has the correct syntax (encodes some pair of FAs). If not, it rejects.

Then, one approach is to construct the FA for:

$$\Delta(A, B) = (L(A) - L(B)) \cup (L(B) - L(A))$$

and test $$\Delta(A, B)$$ for emptiness.
Questions about Context-Free Languages

Many questions about context-free grammars or languages can be readily answered:

**Recursive are:**

a) The acceptance problem: \( A_{cfg} = \{ \langle G, w \rangle : G \text{ is a CFG and } w \text{ a string of } L(G) \} \)

b) Any context-free language with grammar \( G \)

c) The emptiness problem: \( Empty_{cfg} = \{ \langle G \rangle : L(G) \text{ is empty} \} \)

We prove part (a).
Proof that Acceptance is Decidable

For example, convert to Chomsky Normal Form. Examine all derivations of length $2|w| - 1$ and conclude. (See also CYK algorithm from earlier.)
Surprisingly, perhaps, testing whether a CFG generates every string is hard:

\[ \text{Total}_{cfg} = \{ \langle G \rangle : L(G) = \Sigma^* \} \text{ is not recursive.} \]

This will be established later.
Recall that a configuration of a machine is a complete record of the machine that tells one the current state and contents of memory.

Specifically, for a 1-tape TM, we write the state where the head is. That is, the configuration is written as

\[ t_L S t_R \]

where \( t_L \) is the used tape to the left of the head, \( S \) the current state, and \( t_R \) the used tape to the right of the head.
For a deterministic machine, if the same configuration recurs, then the machine is stuck in an infinite loop. Hence:

**Fact.** If there are at most $Q$ possible configurations of a deterministic machine and it runs for longer than $Q$, then it is in an infinite loop.
A **computation string** for machine $M$ accepting string $w$ is the string of configurations from start to finish (separated by some special symbol).

**Fact.** *It is decidable whether a given string is a computation string or not.*

Check that first configuration is correct and matches the input; check that each configuration follows from previous by the rules of $M$; and check that final configuration is accepting.
Other Models

Recall that we introduced the Chomsky Hierarchy earlier. Though we do not prove it, the top level corresponds to TMs:

**Theorem.** There is an unrestricted grammar for a language if and only if it is r.e.
A **linearly bounded automaton** (LBA) is a 1-tape TM whose head is not allowed to move off the input portion of the tape (and there is a device that tells it where the tape starts and finishes).

Though we do not prove it, it turns out that:

**Theorem.** There is a nondeterministic LBA for a language if and only if there is a context-sensitive grammar for it.
1. Show that it is decidable whether an NFA \( M \) generates all strings from the alphabet.

2. Show that the set of recursive languages is closed under reversal.
1. One approach is to convert $M$ to a DFA. It is easy to see whether a DFA accepts every string: only if every state is accepting.

2. The question asks one to show that, if $L$ is recursive, then so is $L^R = \{ x^R : x \in L \}$. If $L$ is decided by TM $M$, then $L^R$ can be decided by a TM that simply reverses the input and then calls $M$. 
Summary

Recursive languages are accepted by TMs that always halt; r.e. languages are accepted by TMs. These two families are closed under intersection and union. If a language is recursive, then so is its complement; if both a language and its complement are r.e., then the language is recursive. There is a connection with printer-TMs.

A problem is decidable if the associated language is recursive. All problems about FAs and REs are decidable; most problems about CFGs and PDAs are decidable. A computation string is a record of the computation of a machine.