Recall that $A_{DFA} = \{\langle M, w \rangle \mid M \text{ is a DFA that accepts } w \}$ is decidable, while $A_{TM} = \{\langle M, w \rangle \mid M \text{ is a TM that accepts } w \}$ is undecidable. We similarly define $A_{CFG} = \{\langle G, w \rangle \mid G \text{ is a CFG and } w \in L(G)\}$. Is $A_{CFG}$ decidable?

Also note that a language is **co-Turing-recognizable** if its complement is Turing-recognizable.

### Topics Covered

1. CFLs and Decidability
2. More TM Languages
3. More CFG Languages

### 1 CFLs and Decidability

**Theorem** If $L$ is a CFL then $L$ is decidable.

**Proof** Recall the conditions for a context-free grammar to be in Chomsky normal form. All rules must be of the form $A \rightarrow BC$, $A \rightarrow a$, or $S \rightarrow \varepsilon$. If a grammar $G$ is in Chomsky normal form, then for a string $w \in L(G)$, a derivation of $w$ takes $2n - 1$ steps where $n = |w|$, or a single step if $w = \varepsilon$. In this case, we can decide $L = L(G)$ with a TM $D$:

[D, on input $w$:

1. Consider all derivations of length $2|w| - 1$. If one derives $w$, accept. Otherwise, reject.]

As we can convert any CFG to Chomsky normal form in finite time, we can use this decider to decide any context-free language. In other words, if $L$ is a CFL, then $L$ is decidable. ■
Proof that $A_{CFG}$ is Decidable  A decider for $A_{CFG}$ is quite similar to the above decider for a CFL. Construct $D$ as follows:

$D$, on input $(G, w)$:
1. Convert $G$ into Chomsky normal form.
2. Consider all derivations of length $2|w| - 1$. If one derives $w$, accept. Otherwise, reject.

Again, note that $D$ is a decider because steps 1 and 2 both take finite time.

We know that CFLs are decidable. Are the decidable languages $A_{DFA}$ and $A_{CFG}$ context-free? We don’t know, but we might suspect not. Intuition suggests that if we had a DFA accepting a finite language, and we then applied the pumping lemma for CFLs, we might reach a contradiction. However, more formalization would be needed to show that this is or isn’t true. Overall, we have the following relationships among types of languages:
2 More TM Languages

The language $E_{TM} = \{ \langle M \rangle \mid L(M) = \emptyset \}$. This is not Turing-recognizable, as can be proven by a reduction from $A_{TM}^C$. Given an input $\langle M, w \rangle$, we want a computable function $f(\langle M, w \rangle) = \langle M' \rangle$ such that $\langle M, w \rangle \in A_{TM}^C$ if and only if $\langle M' \rangle \in E_{TM}$.

Proof that $E_{TM}$ is Turing-Unrecognizable Define $f$ such that on input $\langle M, w \rangle$, $f$ outputs $\langle M' \rangle$ whose description is, “On input $x$, run $M$ on $w$, and do as $M$ does.” If $M$ does not accept $w$, then $M'$ accepts $\emptyset$, so $\langle M' \rangle \in E_{TM}$. If $M$ accepts $w$, then $M'$ accepts $\Sigma^*$, so $\langle M' \rangle \notin E_{TM}$. Moreover, $f$ is computable because we can construct a TM description using a straightforward algorithm. Therefore $A_{TM}^C \leq_m E_{TM}$, implying that $E_{TM}$ is Turing-unrecognizable.

Note that we could construct a recognizer for $E_{TM}^C$ by running $M$ on all inputs $w$ until it accepts. Thus, $E_{TM}^C$ is Turing-recognizable.

The language $\text{Decider}_{TM} = \{ \langle M \rangle \mid M$ is a decider$\}$. We cannot use Rice's Theorem to prove this is undecidable; at a high level, this is because the language is conditioned on the TM $M$ itself, not $M$'s language. However, we can reduce from $A_{TM}$. Without going into too much detail, note that the mapping reduction would involve taking in an input $\langle M, w \rangle$, and outputting $\langle M' \rangle$ with description, “On input $x$, run $M$ on $w$. Accept if $M$ accepts. Else, loop forever.” It turns out that we can even say more about $\text{Decider}_{TM}$—it is neither Turing-recognizable nor co-Turing-recognizable.

However, the language $\text{AcceptInTime}_{TM} = \{ \langle M, w, t \rangle \mid M$ accepts $w$ in at most $t$ steps$\}$ is decidable. A decider could run $M$ on $w$ for $t$ steps, and accept if $M$ accepts and reject otherwise. This takes finite time, so $\text{AcceptInTime}_{TM}$ is decidable.

Proof that $\text{Decider}_{TM}$ is Turing-Unrecognizable We prove this by a reduction from $\text{EQ}_{TM}$, which we know to be neither Turing-recognizable nor co-Turing-recognizable. To show that $\text{EQ}_{TM} \leq_m \text{Decider}_{TM}$, we want a computable function $f(\langle M_1, M_2 \rangle) = \langle M \rangle$ where $\langle M_1, M_2 \rangle \in \text{EQ}_{TM}$ if and only if $\langle M \rangle$ is a decider. Let $M$ be a machine with description, “On input $\langle w, t \rangle$, run $M_1$ and $M_2$ on $w$ for $t$ steps. If both or neither accept, then accept. If one accepts, then finish running the other on $w$. If it accepts, accept. If it rejects, loop forever. Otherwise, also loop forever.”

First, note that $f$ is computable, as we can construct a Turing machine to compute it. Next, consider the correctness of $f$.

1. For the first direction, suppose $\langle M_1, M_2 \rangle \in \text{EQ}_{TM}$. Let $\langle w, t \rangle$ be given. We will show
that $M$ halts on $(w, t)$ in this case.

(a) First, suppose $w \in L(M_1) = L(M_2)$. Let $t_1$ be the number of steps $M_1$ takes to accept $w$, and let $t_2$ be the number of steps $M_2$ takes to accept $w$. If $t < t_1, t_2$, then neither $M_1$ nor $M_2$ accepts, and $M$ accepts. If $t_1 < t < t_2$, then $M_1$ accepts and and $M$ runs $M_2$ until $M_2$ also accepts, and $M$ accepts. If $t > t_1, t_2$, both $M_1$ and $M_2$ accept in time $t$ and $M$ accepts.

(b) Next, suppose $w \notin L(M_1) = L(M_2)$. Then neither machine accepts $w$ in $t$ steps, so $M$ accepts. Thus in these two cases when $(M_1, M_2) \in EQ_{TM}$, $M$ halts.

2. For the other direction, suppose $(M_1, M_2) \notin EQ_{TM}$. To show that $M$ is not a decider, we want to find an input $(w, t)$ such that $M$ loops forever. Without loss of generality, let $w$ be a string such that $w \in L(M_1)$ and $w \notin L(M_2)$. Let $t$ be the number of steps it takes $M_1$ to accept $w$. Then on input $(M, w)$, $M$ sees that $M_1$ accepts but $M_2$ has not yet accepted (so $M_2$ either rejects or loops). Since $M_2$ will not accept $w$, $M$ loops forever. In particular, $M$ is not a decider.

Thus, $(M_1, M_2) \in EQ_{TM}$ if and only if $M$ is a decider. Since $EQ_{TM} \leq_m \text{DECIDER}_{TM}$, and $EQ_{TM}$ is not Turing-recognizable, it follows that $\text{DECIDER}_{TM}$ is not Turing-recognizable. Moreover, we can use the same function $f$ to show that $EQ_{TM}^{C} \leq_m \text{DECIDER}_{TM}^{C}$, so $\text{DECIDER}_{TM}^{C}$ is not Turing-recognizable. In other words, $\text{DECIDER}_{TM}$ is neither Turing-recognizable nor co-Turing-recognizable. 

3 More CFG Languages

The language $\text{ALL}_{DF,A} = \{ (M) \mid M \text{ is a DFA that accepts everything}; i.e. L(M) = \Sigma^* \}$. This is decidable, as we can check whether any path in the DFA leads to a reject state. However, the language $\text{ALL}_{CFG} = \{ (G) \mid G \text{ is a CFG such that } L(G) = \Sigma^* \}$ is undecidable.

**Proof that $\text{ALL}_{CFG}$ is Undecidable** This can be shown by a reduction from $A_{TM}$. The computable function $f$, on input $(M, w)$, outputs the description of a CFG $G$ that generates any string over $\Gamma \cup Q \cup \{ \# \}$ that is not an encoding of an accepting CH of $M$ on $w$. At a high level, a computation history looks something like:

$$
\#q_0w_1w_2\ldots w_n\#b_1q_2w_2\ldots w_n\#\ldots\#
$$

Let $C_1 = q_0w_1w_2\ldots w_n$, $C_2 = b_1q_2w_2\ldots w_n$, and so on. We then have a CH, and format it so that every other $C_i$ is reversed:

$$
s = \#C_1\#C_2^R\#C_3\#C_4^R\#\ldots\#C_{\ell}\#
$$

Consider the conditions in which $s$ does not represent an accepting configuration:
1. It does not start with \( #C_1 \) or does not end in \\#.  
2. Some \( C_i \) does not encode a valid configuration.  
3. \( C_\ell \) does not contain \( q_{\text{accept}} \).  
4. For some \( i \), \( C_{i+1} \) does not follow from \( C_i \).

We can generate \( s \) by taking the union of four CFGs:

1. Generate all strings that don’t have the form \( #C_1 \ldots # \).  
2. Only make invalid configurations.  
3. Only make \( C_\ell \) not containing \( q_{\text{accept}} \).  
4. Generate configurations such that \( C_{i+1} \) does not follow from \( C_i \). This is tricky, and it relies on having reversed every other \( C_i \) in \( s \).

When we take the union of these four CFGs, the result is a CFG \( G \) such that if \( M \) does accept \( w \), then \( G \) cannot generate \( w \). If \( M \) does not accept \( w \), then there is a rejecting computation history, which \( G \) can generate. Thus, \( A_{TM} \leq_m \text{AllCFG} \), so \( \text{AllCFG} \) is undecidable. ■

Another CFG language, \( EQ_{CFG} = \{ \langle G_1, G_2 \rangle \mid G_1, G_2 \text{ are CFGs for the same language} \} \) is also undecidable.