Corollary G \( \tilde{U} = \{ \langle M, w \rangle \mid w \in L(M) \} \) is not Turing-recognizable.

Proof. \( \tilde{U} = \tilde{U} \cup \text{ERR} \), where ERR is the easy to decide language:
\[
\text{ERR} = \{ x \in \{0, 1\}^* \mid x \text{ does not have a prefix that is a valid code for a Turing machine} \}.
\]

Counter-assumption: \( \tilde{U} \) is Turing-recognizable

- Then by Theorem B, \( \tilde{U} \cup \text{ERR} = \tilde{U} \) is Turing-recognizable.
- \( U \) is known to be Turing-recognizable (Th. E) and now also \( \tilde{U} \) is Turing-recognizable. Hence, by Theorem 4.22, \( U \) is decidable.

This is a contradiction with Theorem F and the counter-assumption does not hold. i.e., \( \tilde{U} \) is not Turing-recognizable. \( \Box \)

The Halting Problem is Undecidable

- Analogously to the acceptance problem of DFAs, we can pose the halting problem of Turing machines:
  
  \[ \text{Does the given Turing machine } M \text{ halt on input } w? \]
  
- This is an undecidable problem. If it could be decided, we could easily decide also the universal language

**Theorem 5.1** \( \text{HALT}_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and halts on input } w \} \) is Turing-recognizable, but not decidable.

**Proof.** \( \text{HALT}_{TM} \) is Turing-recognizable: The universal Turing machine \( M_u \) of Theorem E is easy to convert into a TM that simulates the computation of \( M \) on input \( w \) and accepts if and only if the computation being simulated halts.
HALT_TM is not decidable: counter-assumption: \( \text{HALT}_TM = \mathcal{L}(M_{\text{HALT}}) \) for the total Turing machine \( M_{\text{HALT}} \).

Now we can compose a decider for the language \( U \) by combining machines \( M_U \) and \( M_{\text{HALT}} \) as shown in the next figure.

The existence of such a TM is a contradiction with Theorem F. Hence, the counter-assumption cannot hold and \( \text{HALT}_TM \) is not decidable.

Corollary H \( \hat{\mathcal{H}} = \{ \langle M, w \rangle \mid M \text{ is a TM and does not halt on input } w \} \) is not Turing-recognizable.

Proof. Like in corollary G.

A total TM for the universal language \( U \)
Chomsky hierarchy

- A formal language $L$ can be recognized with a Turing machine if and only if it can be generated by an unrestricted grammar.
- Hence, the languages generated by unrestricted grammars are Turing-recognizable languages.
- They constitute type 0 languages of Chomsky hierarchy.
- Chomsky’s type 1 languages are the context-sensitive ones. It can be shown that they are all decidable.
- On the other hand, there exists decidable languages, which cannot be generated by context-sensitive grammars.
Halting Problem in Programming Language

The correspondence between Turing machines and programming languages:

- All TMs ~ programming language
- One TM ~ program
- The code of a TM ~ representation of a program in machine code
- Universal TM ~ interpreter for machine language

The interpretation of the undecidability of the halting problem in programming languages:

```
There does not exist a Java method, which could decide whether any given Java method M halts on input w”.
```

Let us assume that there exists a total Java method h that returns true if the method represented by string m halts on input w and false otherwise:

```java
boolean h(String m, String w)
```

Now we can program the method hHat

```java
boolean hHat( String m )
{ if (h(m,m))
    while (true) ;}
```

Let H be the string representation of hHat. hHat works as follows:

hHat (H) halts ↔ h (H, H) = false ↔ hHat (H) does not halt
5. Reducibility

- The proof of unsolvability of the halting problem is an example of a reduction:
  - a way of converting problem $A$ to problem $B$ in such a way that a solution to problem $B$ can be used to solve problem $A$
  - If the halting problem were decidable, then the universal language would also be decidable
  - Reducibility says nothing about solving either of the problems alone; they just have this connection
  - We know from other sources that the universal language is not decidable
  - When problem $A$ is reducible to problem $B$, solving $A$ cannot be harder than solving $B$ because a solution to $B$ gives one to $A$
  - If an unsolvable problem is reducible to another problem, the latter also must be unsolvable

Non-emptiness Testing for TMs

(Observe that the book deals with $E_{TM}$)

The following decision problem is undecidable:
``Does the given Turing machine accept any inputs?``

$NE_{TM} = \{ \langle M \rangle | M \text{ is a Turing machine and } L(M) \neq \emptyset \}$

**Theorem (5.2)** $NE_{TM}$ is Turing-recognizable, but not decidable

**Proof.** The fact that $NE_{TM}$ is Turing-recognizable will be shown in the exercises.
- Let us assume that $NE_{TM}$ has a decider $M'_{NE}$
- Using it we can construct a total Turing machine for the language $U$
- Let $M$ be an arbitrary Turing machine, whose operation on input $w$ is under scrutiny
Let $M^w$ be a Turing machine that replaces its actual input with the string $w = a_1a_2...a_k$ and then works as $M$.

Operation of $M^w$ does not depend in any way about the actual input.

- The TM either accepts or rejects all inputs:

$$L(M^w) = \begin{cases} 
\{0,1\}^+, & \text{if } w \in L(M) \\
\emptyset, & \text{if } w \notin L(M) 
\end{cases}$$
Let $M_{ENC}$ be a TM, which

- Inputs the concatenation of the code $\langle M \rangle$ for a Turing machine $M$ and a binary string $w$, $\langle M, w \rangle$, and
- Leaves to the tape the code $\langle M^w \rangle$ of the TM $M^w$

By combining $M_{ENC}$ and the decider $M_{TNE}$ for the language $\text{NE}_{TM}$ we are now able to construct the following Turing machine $M_{UT}$

A decider $M_{UT}$ for the universal language $U$
• $M_U^T$ is total whenever $M_{NE}^T$ is, and $L(M_U^T) = U$ because

\[
\begin{align*}
\langle M, w \rangle &\in L(M_U^T) \\
\Leftrightarrow \langle M^w \rangle &\in L(M_{NE}^T) = NE_{TM} \\
\Leftrightarrow L(M^w) &\neq \emptyset \\
\Leftrightarrow w &\in L(M) \\
\Leftrightarrow \langle M, w \rangle &\in U
\end{align*}
\]

• However, by Theorem F $U$ is not decidable, and the existence of the TM $M_U^T$ is a contradiction

• Hence, the language $NE_{TM}$ cannot have a total recognizer $M_{NE}^T$ and we have, thus, proved that the language $NE_{TM}$ is not decidable.

---

**TM Recognizing Regular Languages**

• Similarly, we can show that recognizing those Turing machines that accept a regular language is undecidable by reducing the decidability of the universal language into this problem

The decision problem is:

"Does the given Turing machine accept a regular language?"

\[
REG_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) \text{ is a regular language} \}
\]

**Theorem 5.3** $REG_{TM}$ is undecidable.

**Proof.**

• Let us assume that $REG_{TM}$ has a decider $M_{REG}^e$
Using $M_{\text{REG}}$, we could construct a decider for the universal language $U$.

Let $M$ be an arbitrary Turing machine, whose operation on input $w$ we are interested in.

The language corresponding to balanced pairs of parentheses $\{0^n1^n | n \geq 0 \}$ is not regular, but easy to decide using a TM.

Let $M_{\text{parenth}}$ be a decider for the language.

Now, let $M_{\text{ENC}}$ be a TM, which on input $(M, w)$ composes an encoding for a TM $M^w$, which on input $x$

- First works as $M_{\text{parenth}}$ on input $x$.
- If $M_{\text{parenth}}$ rejects $x$, $M^w$ operates as $M$ on input $w$.
- Otherwise $M^w$ accepts $x$.

Deciding a regular language: the TM $M^w$. 

\[ \text{start} \]

\[ M \]

\[ \text{parenth} \]

\[ w \]

\[ x \]
Thus, $M^w$ either accepts the regular language $\{0, 1\}^*$ or non-regular $\{0^n1^n \mid n \geq 0\}$

Accepting/rejecting the string $w$ on $M$ reduces to the question of the regularity of the language of the TM $M^w$

$$L(M^w) = \begin{cases} \{0, 1\}^* & \text{if } w \in L(M) \\ \{0^n1^n \mid n \geq 0\} & \text{if } w \notin L(M) \end{cases}$$

Let $M_{\text{ENC}}$ be a TM, which

- inputs the concatenation of the code $\langle M \rangle$ for a Turing machine $M$ and a binary string $w$, $\langle M, w \rangle$, and
- Leaves to the tape the code $\langle M^w \rangle$ of the TM $M^w$

Now by combining $M_{\text{ENC}}$ and $M_{\text{REG}}^\tau$ would yield the following Turing machine $M_U^\tau$

A decider $M_U^\tau$ for the universal language $U$
**Rice’s Theorem**

- Any property that only depends on the language recognized by a TM, not on its syntactic details, is called a *semantic property* of the Turing machine.
- E.g.
  - "$M$ accept the empty string", *(NE)*
  - "$M$ accepts some string" (NE).
  - "$M$ accept infinitely many strings",
  - "The language of $M$ is regular" (REG) etc.

- If two Turing machines $M_1$ and $M_2$ have $L(M_1) = L(M_2)$, then they have exactly the same semantic properties.
• More abstractly: a semantic property $S$ is any collection of Turing-recognizable languages over the alphabet $\{0, 1\}$

• Turing machine $M$ has property $S$ if $L(M) \in S$.
• Trivial properties are $S = \emptyset$ and $S = TR$
• Property $S$ is solvable, if language $codes(S) = \{ \langle M \rangle | L(M) \in S \}$ is decidable.

Rice's theorem  All non-trivial semantic properties of Turing machines are unsolvable

Computation Histories

• The computation history for a Turing machine on an input is simply the sequence of configurations that the machine goes through as it processes the input.
• An accepting computation history for $M$ on $w$ is a sequence of configurations $C_1, C_2, \ldots, C_l$, where
  • $C_1$ is the start configuration $q_0 w$,
  • $C_l$ an accepting configuration of $M$, and
  • each $C_i$ legally follows from $C_{i-1}$ according to the rules of $M$

• Similarly one can define a rejecting computation history
• Computation histories are finite sequences — if $M$ doesn’t halt on $w$, no accepting or rejecting computation history exists for $M$ on $w$
Linear Bounded Automata

- A linear bounded automaton (LBA) is a Turing machine that cannot use extra working space.
- It can only use the space taken up by the input.
- Because the tape alphabet can, in any case, be larger than the input alphabet, it allows the available memory to be increased up to a constant factor.
- Deciders for problems concerning context-free languages.
- If a LBA has
  - $q$ states
  - $g$ symbols in its tape alphabet, and
  - an input of length $n$,
  then the number of its possible configurations is $q \cdot n \cdot g^n$.

Theorem 5.9
The acceptance problem for linear bounded automata

$A_{LBA} = \{ \langle M, w \rangle \mid M \text{ is an LBA that accepts string } w \}$

is decidable.

Proof. As $M$ computes on $w$, it goes from configuration to configuration. If it ever repeats a configuration, it will go on to repeat this configuration over and over again and thus be in a loop.

Because an LBA has only $q \cdot n \cdot g^n$ distinct configurations, if the computation of $M$ has not halted in so many steps, it must be in a loop.

Thus, to decide $A_{LBA}$ it is enough to simulate $M$ on $w$ for $q \cdot n \cdot g^n$ steps or until it halts. □
Theorem 5.10
The emptiness problem for linear bounded automata

\[ E_{LBA} = \{ \langle M \rangle \mid M \text{ is an LBA and } L(M) = \emptyset \} \]

is undecidable.

Proof. Reduction from the universal language (acceptance problem for general TMs).

Counter-assumption: \( E_{LBA} \) is decidable; i.e., there exists a decider \( M_{ET} \) for \( E_{LBA} \).

Let \( M \) be an arbitrary Turing machine, whose operation on input \( w \) is under scrutiny. Let us compose an LBA \( B \) that recognizes all accepting computation histories for \( M \) on \( w \).

Now we can reduce the acceptance problem for general Turing machines to the emptiness testing for LBAs:

\[
\begin{align*}
L(B) &\neq \emptyset \quad \text{if } w \in L(M) \\
L(B) &\neq \emptyset \quad \text{if } w \notin L(M)
\end{align*}
\]

The LBA \( B \) must accept input string \( x \) if it is an accepting computation history for \( M \) on \( w \).

Let the input be presented as \( x = C_1\#C_2\#\cdots\#C_l \).
$B$ checks that $x$ satisfies the conditions of an accepting computation history:

- $C_1 = q_0 w$,
- $C_i$ is an accepting configuration for $M$; i.e. accept is the state in $C_i$, and
- $C_{i+1} \Rightarrow_M C_i$:
  - configurations $C_i$ and $C$ are identical except for the position under and adjacent to the head in $C_{i+1}$, and
  - the changes correspond to the transition function of $M$.

Given $M$ and $w$ it is possible to construct LBA $B$ mechanically.

By combining machines $B$ and $M^U$ as shown in the following figure, we obtain a decider for the acceptance problem of general Turing machines (universal language).

$$(M, w) \in L(M^U)$$

$\Leftrightarrow \{B\} \notin L(M^U)$

$\Leftrightarrow L(B) \neq \emptyset$

$\Leftrightarrow w \in L(M)$

$\Leftrightarrow \{M, w\} \in U$

This is a contradiction, and the language $E_{LBA}$ cannot be decidable.
A decider $M^U$ for the universal language $U$

5.3 Mapping Reducibility

- Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \text{accept, reject})$ be an arbitrary standard Turing machine.
- Let us define the partial function $f_M: \Sigma^* \rightarrow \Gamma^*$ computed by the TM as follows:

\[
  f_M(w) = \begin{cases} 
  u, & \text{if } q_0 \xrightarrow{w} u q, \\
  q \in \{ \text{accept, reject} \} \\
  \text{undefined, otherwise}
  \end{cases}
\]

- Thus, the TM $M$ maps a string $w \in \Sigma^*$ to the string $u$, which is the contents of the tape, if the computation halts on $w$.
- If it does not halt, the value of the function is not defined in $w$. 

Definition 5.20

- Partial function $f$ is **computable**, if it can be computed with a total Turing machine. I.e. if its value $f(w)$ is defined for every $w$

- Let us formulate the idea that problem $A$ is "at most as difficult as" problem $B$ as follows:

  - Let $A \subseteq \Sigma^*$, $B \subseteq \Gamma^*$ be two formal languages
  - $A$ is **mapping reducible to** $B$, written
    $$A \leq_m B,$$
    if there is a computable function $f: \Sigma^* \rightarrow \Gamma^*$ s.t.
    $$w \in A \iff f(w) \in B \quad \forall w \in \Sigma^*$$
  - The function $f$ is called the **reduction** of $A$ to $B$

- Mapping an instance $w$ of $A$ computably into an instance $f(w)$ of $B$ and
  
  "does $w$ have property $A$?" $\iff$
  "does $f(w)$ have property $B$?"