What do we know?

- Finite languages $\subseteq$ Regular languages $\subseteq$ Context-free languages $\subseteq$ Turing-decidable languages
- There are automata and grammar descriptions for all these language classes
- Different variations of Turing machines and unrestricted grammars (e.g.) are universal models of computation
- A Turing-decidable language has a total TM that halts on each input
- A Turing-recognizable language has a TM that halts on all positive instances, but not necessarily on the negative ones
- Turing-recognizable languages that are not decidable are semidecidable
- There are languages that are not even Turing-recognizable

Encoding Turing Machines

Standard Turing machine

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \text{accept, reject})$$

where $\Sigma = \{0, 1\}$, can be represented as a binary string as follows:

- $Q = \{q_0, q_1, \ldots, q_n\}$, where $q_{n-1} = \text{accept}$ and $q_n = \text{reject}$
- $\Gamma = \{a_0, a_1, \ldots, a_m\}$, where $a_0 = 0, a_1 = 1, a_2 = \Box$
- Let $\Delta_0 = L$ and $\Delta_1 = R$
- The code for the transition function $\delta$ rule

$$\delta(q_0, a_j) = (q_\rho, a_\sigma, \Delta_\gamma)$$

is

$$c_\delta = 0^{i+1}10^{j+1}10^{i+1}10^{j+1}10^{i+1}10^{j+1}$$
The code for the whole machine $M$, $\langle M \rangle$, is

$$111c_{00}11c_{01}11 \ldots 11c_{0m}11c_{10}11 \ldots 11c_{1n-2,0}11 \ldots 11c_{1n-2,m}11$$

For example, the code for the following TM is

$$\langle M \rangle = 111010100101001001001000100100010010001001000100100010010001000100\ldots 111$$

$\delta(q_0, 0) = (q_1, 0, R)$

$\delta(q_0, 1) = (q_1, 1, R)$

Thus, every standard Turing machine $M$ recognizing some language over the alphabet $\{0, 1\}$ has a binary code $\langle M \rangle$.

On the other hand, we can associate some Turing machine $M_b$ to each binary string $b$.
• However, all binary strings are not codes for Turing machines
  • For instance, 00, 011110, 111000111 and 1110101010111 are not legal codes for TMs according to the chosen encoding
  • We associate with illegal binary strings a trivial machine, $M_{\text{triv}}$, rejecting all inputs:

    $$M_b = \begin{cases} M, & \text{if } b = \langle M \rangle \text{ is the code for Turing machine } M \\ M_{\text{triv}}, & \text{otherwise} \end{cases}$$

• Hence, all Turing machines over \{0, 1\} can be enumerated:
  $$M_e, M_0, M_1, M_{00}, M_{01}, M_{10}, M_{000}, \ldots$$

• At the same time we obtain an enumeration of the Turing-recognizable languages over \{0, 1\}:
  $$L(M_e), L(M_0), L(M_1), L(M_{00}), L(M_{01}), L(M_{10}), \ldots$$

• A language can appear more than once in this enumeration

• By diagonalization we can prove that the language $D$ corresponding to the decision problem
  
  $\text{Does the Turing machine } M \text{ reject its own description } \langle M \rangle?$$

is not Turing-recognizable

• Hence, the decision problem corresponding to $D$ is unsolvable.
Lemma E Language \( D = \{ b \in \{ 0, 1 \}^* \mid b \notin L(M_b) \} \) is not Turing-recognizable

Proof. Let us assume that \( D = L(M) \) for some standard Turing machine \( M \).

Let \( d = \langle M \rangle \) be the binary code of \( M \); i.e., \( D = L(M_d) \).

However,
\[
d \in D \iff d \in L(M) = D.
\]

We have a contradiction and the assumption cannot hold. Hence, there cannot exist a standard Turing machine \( M \) s.t. \( D = L(M) \).

Therefore, \( D \) cannot be Turing-recognizable. \( \square \)

4 Decidability

- The acceptance problem for DFAs: Does a particular DFA \( B \) accept a given string \( w \)?
- Expressed as a formal language
  \[
  A_{DFA} = \{ \langle B, w \rangle \mid B \text{ is a DFA that accepts string } w \}
  \]
- We simply need to represent a TM that decides \( A_{DFA} \)
- The Turing machine can easily simulate the operation of the DFA \( B \) on input string \( w \)
- If the simulation ends up in an accept state, accept
  - If it ends up in a nonaccepting state, reject
- DFAs can be represented (encoded) in a similar vain as Turing machines
Theorem 4.1 \( A_{DFA} \) is a decidable language.

- NFAs can be converted to equivalent DFAs, and hence a corresponding theorem holds for them
- Regular expressions, on the other hand, can be converted to NFAs, and therefore a corresponding result also holds for them
- A different kind of a problem is \textit{emptiness testing}: recognizing whether an automaton \( A \) accepts any strings at all

\[
E_{DFA} = \{ \langle A \rangle | A \text{ is a DFA and } L(A) = \emptyset \}.
\]

Theorem 4.4 \( E_{DFA} \) is a decidable language.

- The TM can work as follows on input \( \langle A \rangle \):
  1. Mark the start state of \( A \)
  2. Repeat until no new states get marked:
     - Mark any state that has a transition coming in from any state that is already marked
  3. If no accept state is marked, accept; otherwise, reject
- Also determining whether two DFAs recognize the same language is decidable
- The corresponding language is \( EQ_{DFA} = \{ \langle A, B \rangle | A \text{ and } B \text{ are DFAs and } L(A) = L(B) \} \)
- The decidability of \( EQ_{DFA} \) follows from Theorem 4.4 by turning to consider the \textit{symmetric difference} of \( L(A) \) and \( L(B) \)
\[ L(C) = (L(A) \setminus L(B)) \cup (L(B) \setminus L(A)) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B)) \]

- \( L(C) = \emptyset \) if and only if \( L(A) = L(B) \)
- Because the class of regular languages is closed under complementation, union, and intersection, we (the TM) can construct automaton \( C \) given \( A \) and \( B \)
- We can use Theorem 4.4 to test whether \( L(C) \) is empty

**Theorem 4.5** \( EQ_{DFA} \) is a decidable language.

- Turning to context-free grammars we cannot go through all derivations because there may be an infinite number of them

In the acceptance problem we can consider the grammar converted into Chomsky normal form in which case any derivation of a string \( w \), \(|w| = n\), has length \( 2n-1 \)
- The emptiness testing of context-free grammars can be decided in a different manner (see the book)
- The equivalence problem for context-free grammars, on the other hand, is not decidable
- Because of the decidability of the acceptance problem, all context-free languages are decidable
- Hence, regular languages are a proper subset of context-free languages, which are decidable
- Furthermore, decidable languages are a proper subset of Turing-recognizable languages
4.2 The Halting Problem

- The technique of diagonalization was discovered in 1873 by Georg Cantor who was concerned with the problem of measuring the sizes of infinite sets.

- For finite sets we can simply count the elements.

- Do infinite sets \( \mathbb{N} = \{0, 1, 2, \ldots\} \) and \( \mathbb{Z}^+ = \{1, 2, 3, \ldots\} \) have the same size?

  - \( \mathbb{N} \) is larger because it contains the extra element 0 and all other elements of \( \mathbb{Z}^+ \).

  - The sets have the same size because each element of \( \mathbb{Z}^+ \) can be mapped to an unique element of \( \mathbb{N} \) by \( f(z) = z - 1 \).

- We have already used this comparison of sizes:
  - \( |A| \leq |B| \) iff there exists a one-to-one function \( f: A \rightarrow B \).
  - One-to-one (injection): \( a_1 \neq a_2 \Rightarrow f(a_1) \neq f(a_2) \).

- We have also examined the equal size of two sets \( |A| = |B| \) through a bijective \( f: A \rightarrow B \) mapping:
  - Bijection (correspondence) = one-to-one + onto.
  - Onto (surjection): \( f(A) = B \) or \( \forall b \in B: \exists a \in A: b = f(a) \).
  - A bijection uniquely pairs the elements of the sets \( A \) and \( B \).

- \( |\mathbb{N}| = |\mathbb{Z}^+| \).

- An infinite set that has the same size as \( \mathbb{N} \) is countable.
The set of rational numbers

\[ \mathbb{Q} = \{ \frac{m}{n} \mid m, n \in \mathbb{Z} \land n \neq 0 \} \]

- In between any two integers there is an infinite number of rational numbers.
- Nevertheless, \( |\mathbb{Q}| = |\mathbb{N}| \).
- Mapping \( f : \mathbb{Q} \to \mathbb{Z}^2, f(m/n) = (m, n) \)
  - Integers \( m \) and \( n \) have no common factors!
  - Mapping \( f \) is a one-to-one. Hence, \( |\mathbb{Q}| \leq |\mathbb{Z}^2| = |\mathbb{N}| \).
- On the other hand, \( \mathbb{N} \subseteq \mathbb{Q} \Rightarrow |\mathbb{N}| \leq |\mathbb{Q}| \)

\[ \therefore |\mathbb{Q}| = |\mathbb{N}| \]
Let us assume that the interval \([0, 1]\) is countable and apply
Cantor’s diagonalization to the numbers \(x_1, x_2, x_3, \ldots\) in \([0, 1]\).
Let the decimal representations of the numbers within the
interval be (excluding infinite sequences of 9s)
\[ x_i = \sum_{j=1}^{\infty} d_{ij} \cdot 10^{-j} \]

Let us construct a new real number
\[ x = \sum_{j=1}^{\infty} d_j \cdot 10^{-j} \]
such that
\[ d_j = \begin{cases} 0, & \text{if } d_{ij} > 0 \\ 1, & \text{if } d_{ij} = 0 \end{cases} \]

If, for example
\begin{align*}
x_1 &= 0.23246\ldots \\
x_2 &= 0.30589\ldots \\
x_3 &= 0.21754\ldots \\
x_4 &= 0.05424\ldots \\
x_5 &= 0.99548\ldots \\
\end{align*}
then \(x = 0.01000\ldots\)

Hence, \(x \neq x_i\) for all \(i\)
The assumption about the countability of the numbers within the
interval \([0, 1]\) is false
\[ |\mathbb{R}| = |[0, 1]| \neq |\mathbb{N}| \]
Universal Turing Machines

- The universal language $U$ over the alphabet $\{0, 1\}$ is
  $$U = \{ \langle M, w \rangle \mid w \in L(M) \}.$$  

- The language $U$ contains information on all Turing-recognizable languages over $\{0, 1\}$:
  - Let $A \subseteq \{0, 1\}^*$ be some Turing-recognizable language and $M$ a standard TM recognizing $A$. Then
    $$A = \{ w \in \{0, 1\}^* \mid \langle M, w \rangle \in U \}.$$  

- Also $U$ is Turing-recognizable.
- Turing machines recognizing $U$ are called universal Turing machines.

Theorem F Language $U$ is Turing-recognizable.

Proof. The following three-tape TM $M_U$ recognizes $U$

1. First $M_U$ checks that the input $cw$ in tape1 contains a legal encoding $c$ of a Turing machine. If not, $M_U$ rejects the input.
2. Otherwise $w = a_1a_2...a_k \in \{0, 1\}^*$ is copied to tape 2 in the form $00010^{a_1+1}010^{a_2+1}1...10^{a_k+1}10000$.
3. Now $M_U$ has to find out whether the TM $M (c = \langle M \rangle)$ would accept $w$. Tape 1 contains the description $c$ of $M$, tape 2 simulates the tape of $M$, and tape 3 keeps track of the state of the TM $M$:
   $$q_i \sim 0^{i+1}$$
4. $M_U$ works in phases, simulating one transition of $M$ at each step

1. First $M_U$ searches the position of the encoding of $M$ (tape 1) that corresponds to the simulated state (tape 3) of $M$ and the symbol in tape 2 at the position of the tape head.

2. Let the chosen sequence of encoding be
   \[0^{i+1}10^{j+1}10^{r+1}10^{s+1}10^{t+1},\]
   which corresponds to the transition function $\delta$ rule
   \[\delta(q, a) = (q, a', \Delta).\]
   
   tape 3: $0^{i+1} \rightarrow 0^{i+1}$
   tape 2: $0^{r+1} \rightarrow 0^{s+1}$

   In addition, the head of tape 2 is moved to the left so that the code of one symbol is passed, if $\varepsilon = 0$, and to the right otherwise.

3. When tape 1 does not contain any code for the simulated state $q_i$, $M$ has reached a final state. Now $i = k+1$ or $i = k+2$, where $q_k$ is the last encoded state. The TM $M_U$ transitions to final state accept or reject.

   Clearly the TM $M_U$ accepts the binary string $\langle M, w \rangle$ if and only if $w \in L(M)$.
Theorem G \textit{Language U is not decidable.}

Proof. Let us assume that \textit{U} has a total recognizer \(M^T_{UT}\).

Then we could construct a recognizer \(M_D\) for the "diagonal language" \(D\), which is not Turing-recognizable (Lemma E), based on \(M^T_{UT}\) and the following total Turing machines:

- \(M_{OK}\) tests whether the input binary string is a valid encoding of a Turing machine
- \(M_{DUP}\) duplicates the input string \(c\) to the tape: \(cc\)

Combining the machines as shown in the next picture, we get the Turing machine \(M_D\), which is total whenever \(M^T_{UT}\) is. Moreover,

\[
\begin{align*}
    c &\in L(M_D) \\
    \iff c &\in L(M_{OK}) \vee cc \in L(M^T_{UT}) \\
    \iff c &\in L(M_{D}) \\
    \iff c &\in D = \{ c \mid c \in L(M_{D}) \}
\end{align*}
\]

By Lemma E the language \(D\) is not decidable. Hence, we have a contradiction and the assumption must be false. Therefore, there cannot exist a total recognizer \(M^T_{UT}\) for the language \(U\). \(\Box\)
Corollary H \( \hat{U} = \{ \langle M, w \rangle \mid w \in L(M) \} \) is not Turing-recognizable.

Proof. \( \hat{U} = \hat{U} \cup \text{ERR} \), where \( \text{ERR} \) 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: \( \hat{U} \) is Turing-recognizable

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

This is a contradiction with Theorem G and the counter-assumption does not hold. I.e., \( \hat{U} \) is not Turing-recognizable. \( \square \)
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** \( \text{HALT}_{TM} = \{ (M, w) \mid M \text{ is a TM and halts on input } w \} \) is Turing-recognizable, but not decidable.

**Proof.** \( \text{HALT}_{TM} \) is Turing-recognizable: The universal Turing machine \( M_U \) of Theorem F 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.

\( \text{HALT}_{TM} \) is not decidable: counter-assumption: \( \text{HALT}_{TM} = L(M_{\text{HALT}}) \) 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 G. Hence, the counter-assumption cannot hold and \( \text{HALT}_{TM} \) is not decidable.

**Corollary J** \( \hat{U} = \{ (M, w) \mid M \text{ is a TM and does not halt on input } w \} \) is not Turing-recognizable.

**Proof.** Like in corollary H.
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.
0: Turing-recognizable languages

Decidable languages

1: Context-sensitive languages

2: Context-free languages

3: Regular languages

Finite lang.

e.g. \{ a^k b^k c^k \mid k \geq 0 \}

e.g. \{ a^k b^k \mid k \geq 0 \}

e.g. \{ a^k \mid k \geq 0 \}

\[D, \emptyset, \overline{D}, \ldots\]

\[U, \text{HALT}_{TM}, \ldots\]

\[A_{DFA}, E_{DFA}, \ldots\]

---

Halting Problem in Programming Language

The correspondence between Turing machines and programming languages:

All TMs \sim programming language
One TM \sim program
The code of a TM \sim representation of a program in machine code
Universal TM \sim 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 \( \text{true} \) if the method represented by string \( m \) halts on input \( w \) and \( \text{false} \) 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:

\[ hHat(H) \text{ halts} \iff h(H,H) = \text{false} \iff hHat(H) \text{ does not halt} \]