Theory of Computation

Decidability

Recursive and Recursively Enumerable Languages

Source of Slides: Introduction to Automata Theory, Languages, and Computation
By John E. Hopcroft, Rajeev Motwani and Jeffrey D. Ullman
And
Introduction to Languages and The Theory of Computation by J. C. Martin

Prof. Muhammad Saeed
Basic Mathematical Definitions

- **Numbers:**
  - $\mathbb{N} =$ natural numbers = \{1, 2, 3, …\}
  - $\mathbb{Z} =$ integers = \{…, -2, -1, 0, 1, 2, …\}
  - $\mathbb{Q} =$ rational numbers (expressed in ratios, 1/5, 3/7, 2/9 etc.)
  - $\mathbb{R} =$ real numbers (floating point, 1.5, 2.5674 etc.)
  - $\mathbb{C} =$ complex numbers (2+5i, 27-3i etc.)

- **Countable Set:**
  A set is countable if there is a one-to-one correspondence between the set and $\mathbb{N}$, the natural numbers. It can be enumerated.

\{ a, b, c, d, e, ……… \}

N, Z and Q are countable. R and C are uncountable.
Rational Numbers, $Q$ are countable
Uncountable Set: Cantor’s Diagonalization

\[ s_1 = (0, 0, 0, 0, 0, 0, 0, ...) \]
\[ s_2 = (1, 1, 1, 1, 1, 1, 1, ...) \]
\[ s_3 = (0, 1, 0, 1, 0, 1, 0, ...) \]
\[ s_4 = (1, 0, 1, 0, 1, 0, 1, ...) \]
\[ s_5 = (1, 1, 0, 1, 0, 1, 1, ...) \]
\[ s_6 = (0, 0, 1, 1, 0, 1, 1, ...) \]
\[ s_7 = (1, 0, 0, 0, 1, 0, 0, ...) \]

..........

\[ s_0 = (1, 0, 1, 1, 0, 1, ...) \]
R is uncountable

Proof:

- Suppose $\mathbb{R}$ is countable
- List $\mathbb{R}$ according to the bijection $f$:

<table>
<thead>
<tr>
<th>$n$</th>
<th>$f(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.14159…</td>
</tr>
<tr>
<td>2</td>
<td>5.55555…</td>
</tr>
<tr>
<td>3</td>
<td>0.12345…</td>
</tr>
<tr>
<td>4</td>
<td>0.50000…</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Decidability

R is uncountable

Proof:
  o Suppose R is countable
  o List R according to the bijection f:

<table>
<thead>
<tr>
<th>n</th>
<th>f(n)</th>
<th>set x = 0.a_1a_2a_3a_4…</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.14159…</td>
<td>where digit a_i ≠ i^{th} digit after decimal</td>
</tr>
<tr>
<td>2</td>
<td>5.55555…</td>
<td>point of f(i) (not 0, 9)</td>
</tr>
<tr>
<td>3</td>
<td>0.12345…</td>
<td>e.g. x = 0.2312…</td>
</tr>
<tr>
<td>4</td>
<td>0.50000…</td>
<td>x cannot be in the list!</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>
Decidability

Turing decidability

L is *Turing decidable* (or just decidable) if there exists a Turing machine M that accepts all strings in L and rejects all strings not in L. Note that by rejection we mean that the machine halts after a finite number of steps and announces that the input string is not acceptable. Acceptance, as usual, also requires a decision after a finite number of steps.
Decidability

Turing Recognizability

L is *Turing recognizable* if there is a Turing machine M that recognizes L, that is, M should accept all strings in L and M should not accept any strings not in L. This is not the same as decidability because recognizability does not require that M actually reject strings not in L. M may reject some strings not in L but it is also possible that M will simply "remain undecided" on some strings not in L; for such strings, M's computation never halts.
Decidability

- Recursion and Recursive Functions
- Enumerable Sets
- Recursively Enumerable Languages
- Recursive Languages
- Non-Recursively Enumerable Languages
Decidability

Decidable \( \subset \) RE \( \subset \) All languages
Decidability

A language is *recursively enumerable* if some Turing machine accepts it.

M is a *Turing Machine* and L is a *recursively enumerable language* that M accepts, if a string $w \in L$ then M *halts in a final state* and if $w \notin L$ then M *halts in a non-final state or loops forever*.
Decidability

A language is **recursive** if some Turing machine accepts it and halts on any input string.

OR

A language is **recursive** if there is a membership algorithm for it.

M is a *Turing Machine* and L is a *recursive language* that M accepts,
if a string $w \in L$ then M *halts in a final state* and
if $w \notin L$ then M *halts in a non-final state*
A language is **recursively enumerable** if and only if there is an **enumeration procedure** for it.

If a language $L$ is **recursive** then there is an **enumeration procedure** for it.
If the alphabet is \( \{a, b\} \) then \( \tilde{M} \) can enumerate strings as follows:

\[
\begin{align*}
&a \\
&b \\
&aa \\
&ab \\
&ba \\
&bb \\
&aaa \\
&aab \\
&...
\end{align*}
\]
Decidability

Enumeration procedure

Repeat:

$\hat{M}$ generates a string $w$

$M$ checks if $w \in L$

YES: print $w$ to output

NO: ignore $w$

Example: $L = \{b, ab, bb, aaa, ....\}$

<table>
<thead>
<tr>
<th>$\hat{M}$</th>
<th>$L(M)$</th>
<th>Enumeration Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
</tr>
<tr>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
</tr>
<tr>
<td>$aa$</td>
<td>$ab$</td>
<td>$ab$</td>
</tr>
<tr>
<td>$ab$</td>
<td>$ab$</td>
<td>$ab$</td>
</tr>
<tr>
<td>$ba$</td>
<td>$bb$</td>
<td>$bb$</td>
</tr>
<tr>
<td>$bb$</td>
<td>$bb$</td>
<td>$bb$</td>
</tr>
<tr>
<td>$aaa$</td>
<td>$aaa$</td>
<td>$aaa$</td>
</tr>
<tr>
<td>$aab$</td>
<td>$aaa$</td>
<td>$aaa$</td>
</tr>
<tr>
<td>......</td>
<td>......</td>
<td>......</td>
</tr>
</tbody>
</table>
Decidability

**Theorem:**
S is an **infinite countable** set, the powerset $2^S$ of S is **uncountable**

Since S is countable,
we can write $S = \{ s_1, s_2, s_3, s_4, \ldots \}$
where S consists of $s_1, s_2, s_3, s_4, \ldots$ elements
and the powerset $2^S$ is of the form:

$\{ \{s_1\}, \{s_2\}, \ldots \{s_1, s_2\}, \ldots \{s_1, s_2, s_3, s_4\} \ldots \}$
String Encoding:

We encode each element of the power set with a binary string of 0's and 1's.

<table>
<thead>
<tr>
<th>Powerset element</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>{s_1}</td>
<td>s_1: 1</td>
</tr>
<tr>
<td></td>
<td>s_2: 0</td>
</tr>
<tr>
<td></td>
<td>s_3: 0</td>
</tr>
<tr>
<td></td>
<td>s_4: 0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>{s_2, s_3}</td>
<td>s_1: 0</td>
</tr>
<tr>
<td></td>
<td>s_2: 1</td>
</tr>
<tr>
<td></td>
<td>s_3: 1</td>
</tr>
<tr>
<td></td>
<td>s_4: 0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>{s_1, s_3, s_4}</td>
<td>s_1: 1</td>
</tr>
<tr>
<td></td>
<td>s_2: 0</td>
</tr>
<tr>
<td></td>
<td>s_3: 1</td>
</tr>
<tr>
<td></td>
<td>s_4: 1</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Decidability

\[
\begin{array}{cccccc}
  t_1 & 1 & 0 & 0 & 0 & 0 & \ldots \\
  t_2 & 1 & 1 & 0 & 0 & 0 & \ldots \\
  t_3 & 1 & 1 & 0 & 1 & 0 & \ldots \\
  t_4 & 1 & 1 & 0 & 0 & 1 & \ldots \\
\end{array}
\]

New element: \(0011\ldots\)

The powerset \(2^S\) of \(S\) is uncountable
Example Alphabet: $\{a, b\}$

The set of all Strings:

$$S = \{a, b\}^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$$

infinite and countable

The powerset of $S$ contains all languages:

$$2^S = \{\{\lambda\}, \{a\}, \{a, b\} \{aa, ab, aab\}, \ldots\}$$

$L_1$ $L_2$ $L_3$ $L_4$ $\ldots$

uncountable
Decidability

Languages: uncountable

$L_1 \quad L_2 \quad L_3 \quad \ldots \quad L_k$

$M_1 \quad M_2 \quad M_3 \quad ?$

Turing machines: countable

There are more languages than Turing Machines

There are some languages not accepted by Turing Machines
Decidability

Coding Turing Machines

- We shall assume the states are $q_1, q_2, \ldots, q_r$ for some $r$. The start state will always be $q_1$, and $q_2$ will be the only accepting state. Note that, since we may assume the TM halts whenever it enters an accepting state, there is never any need for more than one accepting state.

- We shall assume the tape symbols are $X_1, X_2, \ldots, X_s$ for some $s$. $X_1$ always will be the symbol 0, $X_2$ will be 1, and $X_3$ will be $B$, the blank. However, other tape symbols can be assigned to the remaining integers arbitrarily.

- We shall refer to direction $L$ as $D_1$ and direction $R$ as $D_2$.

$$\delta(q_i, X_j) = (q_k, X_l, D_m) \quad \text{where} \quad 0^i 10^j 10^k 10^l 10^m$$
**Decidability**

Turing Machine $M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\})$

Transition Rules are:

- $\delta(q_1, 1) = (q_3, 0, R)$
- $\delta(q_3, 0) = (q_1, 1, R)$
- $\delta(q_3, 1) = (q_2, 0, R)$
- $\delta(q_3, B) = (q_3, 1, L)$

**Coding**

<table>
<thead>
<tr>
<th>Coding</th>
<th>010010001010100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00010101001100</td>
</tr>
<tr>
<td></td>
<td>00010010010100</td>
</tr>
<tr>
<td></td>
<td>00010001001100</td>
</tr>
<tr>
<td></td>
<td>00010001000100</td>
</tr>
</tbody>
</table>

Code for M

| 01001000101010011000101001001001100010010010100100110001001001010011000100100101001100010010010100110001001001010011000100100101001100010010010100110001001001010011000100100101001100010010010100110001001001010011000100100101001100010010010100110001001001010011000100100101001100010010010100110001001001010011000100100101001100010010010100110001001001010011000100100101001100010010010100110001001001010011000100100101001100010010010100110001001001010011000100100101001100010010010100110001001001010011000100100101001100010010010100110001001001010011000100100101001100010010010100110001001001010011000100100101001100010010010100110001001001010011000100100101001100010010010100110001001001010011000100100101001100010010010100110001001001010011000100100101001100010010010100110001001001010011000100100101001100010010010100110001001001010011000100100101001100010010010100110001001001010011000100100101001100010010010100110001001001010011000100100101001100010010010100110001001001010011000100100101001100010010010100110001001001010011000100100101001100010010010100110001001001010011000100100101001100010010010100110001001001010011000100100101001100010010010100110001001001010011000100100101001100010010010100110001001001010011000100100101001100010010010100110001001001
Diagonalization Language:

The language $L_d$, the diagonalization language, is the set of strings $w_i$ such that $w_i$ is not in $L(M_i)$. 

![Diagram showing a grid with numbers and 0s and 1s]
**Theorem:** $L_d$ is not a recursively enumerable language. That is there is no Turing Machine that accepts $L_d$

**Proof:** Suppose $L_d$ were $L(M)$ for some TM $M$. Since $L_d$ is a language over alphabet \{0, 1\}, $M$ would be in the list of Turing machines we have constructed, since it includes all TM's with input alphabet \{0, 1\}. Thus, there is at least one code for $M$, say $i$; that is, $M = M_i$.

Now, ask if $w_i$ is in $L_d$.

- If $w_i$ is in $L_d$, then $M_i$ accepts $w_i$. But then, by definition of $L_d$, $w_i$ is not in $L_d$, because $L_d$ contains only those $w_j$ such that $M_j$ does not accept $w_j$.

- Similarly, if $w_i$ is not in $L_d$, then $M_i$ does not accept $w_i$, Thus, by definition of $L_d$, $w_i$ is in $L_d$.

Since $w_i$ can neither be in $L_d$ nor fail to be in $L_d$, we conclude that there is a contradiction of our assumption that $M$ exists. That is, $L_d$ is not a recursively enumerable language.  

□
Theorem: If $L$ is recursive language, so is $\overline{L}$

**Proof:** Let $L = L(M)$ for some TM $M$ that always halts. We construct a TM $\overline{M}$ such that $\overline{L} = L(\overline{M})$ by the construction suggested in Fig. 10. That is, $\overline{M}$ behaves just like $M$. However, $M$ is modified as follows to create $\overline{M}$:

1. The accepting states of $M$ are made nonaccepting states of $\overline{M}$ with no transitions; i.e., in these states $\overline{M}$ will halt without accepting.

2. $\overline{M}$ has a new accepting state $r$; there are no transitions from $r$.

3. For each combination of a nonaccepting state of $M$ and a tape symbol of $M$ such that $M$ has no transition (i.e., $M$ halts without accepting), add a transition to the accepting state $r$.

Since $M$ is guaranteed to halt, we know that $\overline{M}$ is also guaranteed to halt. Moreover, $\overline{M}$ accepts exactly those strings that $M$ does not accept. Thus $\overline{M}$ accepts $\overline{L}$. □
Decidability

**Theorem:** If $L$ is recursive language, so is $\overline{L}$

![Diagram](image-url)
Decidability

**Theorem:** If both a language $L$ and its complement are RE, then $L$ is recursive. ($\overline{L}$ is recursive as well)

Let $L = L(M_1)$ and $\overline{L} = L(M_2)$. Both $M_1$ and $M_2$ are simulated in parallel by a TM $M$. We can make $M$ a two-tape TM, and then convert it to a one-tape TM, to make the simulation easy and obvious. One tape of $M$ simulates the tape of $M_1$, while the other tape of $M$ simulates the tape of $M_2$. The states of $M_1$ and $M_2$ are each components of the state of $M$. 
Decidability

\[ w \]

\[ M_1 \]

Accept

Accept

\[ M_2 \]

Accept

Reject

Dept. of Computer Science & IT, FUUAST
Theory of Computation
Decidability

<table>
<thead>
<tr>
<th>Decidable</th>
<th>Undecidable</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{DFA}$</td>
<td>$\text{EQ}_{\text{CFG}}$</td>
</tr>
<tr>
<td>$A_{NFA}$</td>
<td>$A_{TM}$</td>
</tr>
<tr>
<td>$A_{REX}$</td>
<td>$\text{HALT}_{TM}$</td>
</tr>
<tr>
<td>$E_{DFA}$</td>
<td>$E_{TM}$</td>
</tr>
<tr>
<td>$E_{QDFA}$</td>
<td>$\text{REGULAR}_{TM}$</td>
</tr>
<tr>
<td>$A_{CFG}$</td>
<td>$E_{QTM}$</td>
</tr>
<tr>
<td>$E_{CFG}$</td>
<td>PCP</td>
</tr>
</tbody>
</table>
Decidability

Decidable Languages about DFA

$A_{DFA}$ is a decidable language.

$A_{DFA} = \{ \langle B, w \rangle | B \text{ is a DFA that accepts input string } w \}.$

$M = \text{“On input } \langle B, w \rangle, \text{ where } B \text{ is a DFA and } w \text{ is a string:}
\begin{enumerate}
  \item Simulate } B \text{ on input } w.
  \item If the simulation ends in an accept state, } accept. \text{ If it ends in a nonaccepting state, } reject. \text{”}
Decidability

$A_{\text{NFA}}$ is a decidable language.

$A_{\text{NFA}} = \{ \langle B, w \rangle \mid B \text{ is an NFA that accepts input string } w \}$.

$N = \text{“On input } \langle B, w \rangle \text{ where } B \text{ is an NFA, and } w \text{ is a string:}
\begin{enumerate}
\item Convert NFA $B$ to an equivalent DFA $C$, using the \textbf{last} procedure.
\item Run TM $M$ on input $\langle C, w \rangle$.
\item If $M$ accepts, \textit{accept}; otherwise, \textit{reject}.”
\end{enumerate}$
Decidability

$A_{\text{REX}}$ is a decidable language.

$A_{\text{REX}} = \{\langle R, w \rangle | \ R \text{ is a regular expression that generates string } w \}.$

$P = \text{“On input } \langle R, w \rangle \text{ where } R \text{ is a regular expression and } w \text{ is a string:}
    \begin{enumerate}
    \item Convert regular expression } R \text{ to an equivalent DFA } A
    \item Run TM } N \text{ on input } \langle A, w \rangle.
    \item If } N \text{ accepts, } accept; \text{ if } N \text{ rejects, } reject.\text{”}
Decidability

Halting and Acceptance Problems:

Acceptance Problem: Does a Turing machine accept an input string?

\[ A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}. \]

\( A_{TM} \) is recursively enumerable.

\[ H(\langle M, w \rangle) = \begin{cases} 
    \text{accept} & \text{if } M \text{ accepts } w \\
    \text{reject} & \text{if } M \text{ does not accept } w.
\end{cases} \]

D = “On input \( \langle M \rangle \), where \( M \) is a TM:
1. Run \( H \) on input \( \langle M, \langle M \rangle \rangle \).
2. Output the opposite of what \( H \) outputs; that is, if \( H \) accepts, reject and if \( H \) rejects, accept.”
Decidability

\[ D(\langle M \rangle) = \begin{cases} 
  \text{accept} & \text{if } M \text{ does not accept } \langle M \rangle \\
  \text{reject} & \text{if } M \text{ accepts } \langle M \rangle.
\end{cases} \]

\[ D(\langle D \rangle) = \begin{cases} 
  \text{accept} & \text{if } D \text{ does not accept } \langle D \rangle \\
  \text{reject} & \text{if } D \text{ accepts } \langle D \rangle.
\end{cases} \]
Decidability
Decidability
Decidability
Decidability
Decidability
Decidability
Decidability