Decidability and Undecidability

Classes of languages/problems:

1) Recursive languages: class of languages accepted by T.M.s that always halt

T.M. test always halts = algorithm
(belts on all inputs in finite time, either accepting or rejecting)

<= decidable problems/languages

problems/languages that are non recursive are called undecidable
=> they don't have algorithms

Note: regular and context-free languages are special cases of recursive languages

2) Recursively enumerable (R.E.) languages:

class of languages defined by T.M. (or procedures)

arbitrary T.M. (that may not halt) = procedure

3) non-R.E. languages

languages/problems for which there is no T.M./procedure

Pictorially:

1) Algorithm (recursive)

\[ w \rightarrow A \rightarrow \text{yes} \quad (w \in L) \]

\[ w \rightarrow \text{no} \quad (w \notin L) \]

2) Procedure (R.E.)

\[ w \rightarrow P \rightarrow \text{yes} \quad (w \in L) \]

3) non-R.E.

\[ w \rightarrow ?? \]
To sum up

**Theorem:** \( L \) is recursive \( \Rightarrow \bar{L} \) is recursive

**Proof:** given algorithm \( A \) for \( L \) construct alg. \( \bar{A} \) for \( \bar{L} \)

Intuitively:

\[
\begin{array}{cccc}
\text{input} & \xrightarrow{\text{yes}} & \text{yes} \\
\bar{A} & \xrightarrow{\text{no}} & \text{no}
\end{array}
\]

More precisely: \( A = (Q, \Sigma, \Gamma, \delta, q_0, \emptyset, F) \)

\[
\bar{A} = (Q \cup \{\bar{F}\}, \Sigma, \Gamma, \bar{\delta}, q_0, \emptyset, \{\bar{F}\})
\]

yes \( \rightarrow \) no: we assume that final states of \( F \) are blocking.
So we get that \( \bar{A} \) blocks in a non-final state.

no \( \rightarrow \) yes: if \( \delta(q, \varepsilon) \) is undefined
then set \( \bar{\delta}(q, \varepsilon) = (\bar{F}, \varepsilon, R) \)

Since \( L \) is recursive, \( A \) always halts (with output \text{yes} or \text{no}),

hence \( \bar{A} \) also halts always (with output \text{no} or \text{yes})

q.e.d.
**Theorem:** Both $L$ and $\overline{L}$ are R.E.

$\Rightarrow$ both $L$ and $\overline{L}$ are recursive.

**Proof:** Let $P$, $\overline{P}$ be procedures (i.e. TMs) for $L$, $\overline{L}$.

We run them in parallel, to get an alg. $A$ for $L$.

**Intuitively:**

```
A

<table>
<thead>
<tr>
<th>Input</th>
<th>P</th>
<th>Yes</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>\overline{P}</th>
<th>No</th>
<th></th>
</tr>
</thead>
</table>
```

Note: we assume that $\overline{P}$ is non-blocking in non-final states.

Note: for every $w$, one of $P$, $\overline{P}$ will halt and give the right answer.

**More precisely:**

For $A$ use 2 tapes, one simulating that of $P$ and one $\overline{P}$.

**States of $A$:** for each state $p$ of $P$, state $\langle p, q \rangle$ of $\overline{P}$

**Transitions:**

- For each transition $\delta(p, \sigma) = (p', \delta, D_1)$ in $P$
- $\overline{P}$

we have a transition in $A$

$\delta(\langle p, q \rangle, \sigma, \delta) = (\langle p', q' \rangle, (\delta', D_1), (\delta', D_2))$

**Final states:** every $\langle p, q \rangle$ s.t. $p$ is final in $P$

Note: if $\overline{P}$ accepts in $q$, then $q$ is final, and we examine it is halting. Hence, if $A$ needs $\langle p, q \rangle$, $p$ cannot be final, and since $q$ in $\overline{P}$ is halting, $A$ rejects.

q.e.d.
The two previous results imply that, for every language \( L \), we have that:

- either both \( L \) and \( \overline{L} \) are recursive
- or at least one of \( L, \overline{L} \) is non-R.E.

<table>
<thead>
<tr>
<th>( L ) rec.</th>
<th>( L ) R.E. but not rec.</th>
<th>( L ) non-R.E.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( L ) rec.</td>
<td>( \checkmark )</td>
<td>( \times )</td>
</tr>
<tr>
<td>( L ) R.E. but not rec.</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>( L ) non-R.E.</td>
<td>( \times )</td>
<td>( \checkmark )</td>
</tr>
</tbody>
</table>

(\( \times \) means that this case is not possible)

**Theorem:** \( L_1, L_2 \) rec. \( \Rightarrow L_1 \cup L_2 \) rec.

(recursive languages are closed under union)

**Proof:** let \( A_1, A_2 \) be algorithms for \( L_1, L_2 \)

![Diagram of \( A_1 \) and \( A_2 \) algorithms]

Output "no" of \( A_1 \) triggers \( A_2 \) means:
- if \( A_1 \) halts in a non-final state \( q \) (i.e. \( w \notin L_1 \)), then we have a transition from \( q \) to the initial state of \( A_2 \) (to feed \( w \) to \( A_2 \), we can store it on a second tape before running \( A_1 \)).

\( \text{Q.E.D.} \)
Theorem: \( L_1, L_2 \text{ R.E. } \Rightarrow \text{ L}_1 \cup \text{L}_2 \text{ R.E.} \)

(R.E. languages are closed under union)

Group: let \( P_1, P_2 \) be procedures (i.e. TMs) for \( L_1, L_2 \)
we run \( P_1, P_2 \) in parallel

\[
\text{input} \quad w \quad \rightarrow \quad P_1 \quad \rightarrow \quad\text{yes} \quad \rightarrow \text{yes} \quad \rightarrow \quad P_{1 \cup L_2} \\
\text{input} \quad w \quad \rightarrow \quad P_2 \quad \rightarrow \quad\text{yes} \quad \rightarrow \text{yes} \quad \rightarrow \quad P_{1 \cup L_2}
\]

Note: we assume that \( P_1, P_2 \) are non-blocking or non-final states
Note: if \( w \in L_1 \cup L_2 \), one of \( P_1 \) or \( P_2 \) will halt and answer \text{yes}

Exercise: work out the details

Exercise: prove/disprove closure under intersection, reversal

Sharing languages to be undecidable/non-R.E.

To show languages to be undecidable (non-R.E. we make use of
the basic idea of feeding the encoding of a T.M. as input
to a T.M.

E.g.: Universal T.M. (UTM):

- input: T.M. \( M \)
  \( M \)'s input string \( w \) \( \langle M, w \rangle \)

- output: UTM accepts \( \langle M, w \rangle \) \( \iff \) \( M \) accepts \( w \)

- Diagonalization: consider what happens when certain T.M.s
  are fed their own encoding as input.
In both cases we need a suitable way of encoding T.M.s by means of strings.

We consider encoding T.M.s by means of strings over \{0, 1\}, i.e., as binary integers.

Set \( M = (Q, \Sigma, \Gamma, \delta, q_0, q_f, F) \) be a T.M.

We assume \( \Sigma = \{0, 1\} \), i.e., we consider only T.M.s over the binary alphabet (this is no limitation).

\( \Gamma = \{0, 1, \#\} \)

\( Q = \{q_0, q_1, \ldots, q_n\} \)

- initial state \( q_0 \)
- single final state \( q_f \)

We use the following notation:

\( k_0 = 0, \quad k_2 = 1, \quad k_3 = \# \)

\( d_1 = \text{left}, \quad d_2 = \text{right} \)

Encoding of transitions: \( \delta(q_i, k_j) = (q_k, k_l, d_m) \)

with \( i, k \in \{1, \ldots, n\} \)

\( j, l \in \{1, 2, 3\} \)

\( m \in \{1, 2\} \)

We encode the transition as \( 0^i 10^j 10^{k_l} 10^m 10 \)

Encoding \( E(M) \) of entire T.M. \( M \):

Set \( c_1, \ldots, c_n \) be the encodings of the transitions of \( M \).

We encode \( M \) as:

\[ E(M) = c_1, 11, c_2, 11, c_3, 11, \ldots, 11, c_n, 11 \]

Encoding of \( M \) with its input, \( w \): \( E(M) 111 w \)

(note: the first 111 indicates that the encoding \( E(M) \) is finished)
Note:
- Each bit string encodes a unique T.M.
- Either it is a valid encoding and encodes a unique T.M.
- Or it is not a valid encoding according to our rules.
- In this case we assume that it encodes the particular machine $M_0$ with 1 state and no transitions ($I(M_0) = \emptyset$)
- Each T.M. admits at least 1 encoding (possibly many)

**Enumerating binary strings:**

We define an ordering on binary strings:
- In increasing order of length
- Strings of the same length are ordered lexicographically

$\Rightarrow$: $\varepsilon$, 0, 1, 00, 01, 10, 11, 000, 001, ...

Set $w_i$ be the i-th string in this ordering
(startering with $w_1 = \varepsilon$)

We define $M_i$ as the T.M. encoded by $w_i$, i.e. $w_i = \varepsilon(M_i)$

$\Rightarrow$ we get an ordering of T.M.'s:
- Each T.M. appears at least once in the ordering
- May appear many times

Note: Each binary string $w_i$ can also be viewed:
- as a string fed as input to a T.M.
- as the encoding $w_i = \varepsilon(M_i)$ of a T.M. $M_i$
The diagonalization language

Exploiting the ordering/enumeration of $w_i/M_i$, we can consider the infinite table $T$ so. $\forall i, j > 1$:

$$T(i, j) = \begin{cases} 1 & \text{if } w_i \in L(M_j) \\ 0 & \text{if } w_i \notin L(M_j) \end{cases}$$

| $w_1$ | $w_2$ | $w_3$ | $w_4$ | ...
|-------|-------|-------|-------|-----
| $M_1$ | 0     | 1     | 1     | 0   |
| $M_2$ | 1     | 1     | 0     | 1   |
| $M_3$ | 0     | 0     | 1     | 1   |
| $M_4$ | 1     | 0     | 1     | 0   |

Each row of $T$ is a characteristic vector of $L(M_i)$, specifying which strings belong to $L(M_i)$.

**Definition:** The diagonalization language

$$L_d = \{ w_i \mid T(i, i) = 0 \} = \{ w_i \mid w_i \in L(M_i) \}$$

In other words: $L_d$ is defined as the language whose characteristic vector is the bit by bit complementation of the diagonal of $T$.

**Theorem:** $L_d$ is non-R.E.

**Proof:** By contradiction, assume $L_d$ is R.E. and has a T.M. that accepts it.

Then $\exists k > 1$ so: $L(M_k) = L_d$
Question: is $w_k \in L_d$?

Case 1: $w_k \in L_d \Rightarrow w_k \in L(M_k)$
\[ \Rightarrow T(h, h) = 1 \]
\[ \Rightarrow w_k \notin L_d \quad \text{contradiction} \]

Case 2: $w_k \notin L_d \Rightarrow w_k \notin L(M_k)$
\[ \Rightarrow T(h, h) = 0 \]
\[ \Rightarrow w_k \in L_d \quad \text{contradiction} \]

Intuition: $L_d$ is defined so that it disagrees with each $L(M_i)$ on at least string $w_i$.
\[ \Rightarrow \text{no } M_i \text{ can have } L_d \text{ as its language} \]
But all T.M. s appear in the enumeration
\[ \Rightarrow \text{no T.M. can accept } L_d \]

**Universal T.M.**

**UTM:** Input $\langle \bar{e}(M), w \rangle$ with $\bar{e}(M)$: encoding of e T.M. M
- w input string for M

Action: UTM simulates M on w, and accepts
\[ \langle \bar{e}(M), w \rangle \quad \text{if and only if } M \text{ accepts } w. \]

Language $L_u$ of UTM

Definition: Universal language
\[ L_u = \{ \langle \bar{e}(M), w \rangle \mid w \in L(M) \} \]
Theorem: \( L_n \) is R.E.

Proof: we construct a T.M. \( U \) s.t. \( L(U) = L_n \)

\( U \) has 4 tapes:

- Tape 1: input tape containing \( \langle \varepsilon(M), w \rangle \) (read-only)
- Tape 2: simulates the tape of \( M \)
- Tape 3: contains the current state \( q_1 \) of \( M \): \( 0 \overline{0} \cdots \overline{0} \) (note: the state of \( M \) cannot be encoded in the state of \( U \), since we have no bound on the number of states that \( M \) could have)

- Tape 4: scratch tape

Transitions: \( U \) simulates the transitions on Tape 1, by modifying Tapes 2 and 3

- Initially, copy \( w \) to Tape 2, and \( q_1 = 0 \) to Tape 3
- Initial state

- To simulate each transition of \( M \), \( U \) uses
  - the current state \( q_i = 0^i \) on Tape 3
  - the current symbol \( k_j \) on Tape 2
  - Looks on Tape 1 for transition \( 0^i 10^k 10^k 10^m \), i.e. \( \delta(q_i, k_j) = (q_k, k_e, d_m) \)

  - and - changes the content of Tape 3 to \( q_k = 0^k \)
  - changes the current symbol on Tape 2 to \( k_e \)
  - moves the head on Tape 2 according to \( d_m \)

\( U \) accepts whenever \( M \) enters final state \( q_2 \)
To show that \( L_1 \) is not recursive, we exploit the notion of reduction:

**Reduction**

**Definition:** \( L_1 \) reduces to \( L_2 \) (denoted \( L_1 < L_2 \)) if there exists a function \( R \) (called the reduction from \( L_1 \) to \( L_2 \)) such that:

1. \( R \) is computed by some T.M. \( M_R \) that takes as input a string \( w \) (an instance of \( L_1 \)) and halts, leaving a string \( R(w) \) on its tape (\( M_R \) is an algorithm!)

2. \( w \in L_1 \iff R(w) \in L_2 \)

Intuitively:

\[
\Sigma^* \xrightarrow{R} \Sigma^*
\]

**R maps:** all strings in \( L_1 \) to a subset of all strings in \( L_2 \)

\[
\Sigma^* \xrightarrow{R} \Sigma^*
\]

**Theorem:** \( L_1 < L_2 \) and \( L_2 \) is recursive \( \Rightarrow \) \( L_1 \) is recursive

**Proof:** given algorithm \( A_2 \) for \( L_2 \)

- \( M_R \) for \( R \)

construct algorithm \( A_1 \) for \( L_1 \) (\( A_1 \) is correct since

\[
\begin{align*}
w &\rightarrow M_R \xrightarrow{R(w)} A_2 \xrightarrow{\text{yes}} R(w) \in L_2 \\
A_1
\end{align*}
\]
We can use the same result to show a language to be non-recursive (i.e., undecidable):

**Corollary:** \( L_1 < L_2 \) and \( L_1 \) is non-recursive

\[ \Rightarrow L_2 \text{ is non-recursive} \]

The above results apply also to R.E.

**Theorem:** \( L_1 < L_2 \) and \( L_2 \) is R.E. \( \Rightarrow \) \( L_1 \) is R.E.

\( L_1 < L_2 \) and \( L_1 \) is non-R.E. \( \Rightarrow \) \( L_2 \) is non-R.E.

Intuitively: \( L_1 < L_2 \) means that \( L_2 \) is at least as difficult as \( L_1 \).

**Theorem:** \( L_0 \) is non-recursive

We show that \( \overline{L_0} \) is R.E. (i.e., \( \overline{L_0} \) reduces to \( L_0 \)).

The claim follows, since \( \overline{L_0} \) is non-recursive

(if \( \overline{L_0} \) were recursive, also \( \overline{L_0} \) = \( L_0 \) would be recursive,

but we know that \( L_0 \) is non-R.E.)

Reduction \( R \): given input string \( w \) for \( \overline{L_0} \)

produce input string \( \langle E(M), w \rangle \) for \( L_0 \)

We define \( R(w) = \langle aw \rangle = w \ 111 \ w \).

Clearly, there exists an algorithm \( M_e \) to convert \( w \) into \( \langle w, w \rangle \).

We need to show: \( w \in \overline{L_0} \iff R(w) \in L_0 \)

\( w_e \in \overline{L_0} \iff \langle w_e \rangle \in E(M_e) \iff \langle \langle E(M_e), w_e \rangle \in L_0 \)

\( \iff \langle aw_e, w_e \rangle \in L_0 \iff R(w_e) \in L_0 \)
To sum up:

$\overline{E_d}$ is non-R.E. - direct proof via diagonalization.

$\overline{E_d}$ is R.E., but non-recursively: exercise.

$E_n$ is R.E., but non-recursively: R.E. by construction of $U$.

$E_n$ is non-R.E.: by inference from previous line.

We can exploit this to show a language $L$ to be non-rec. or non-R.E.

- $\overline{E_d} \leq L$ or $E_n \leq L \Rightarrow L$ is non-recursively.
- $E_d \leq L$ or $\overline{E_n} \leq L \Rightarrow L$ is non-R.E.

(And hence non-recursively).

Consider the following languages over $\Sigma = \{0, 1\}$

$L_e = \{E(M) \mid \not\exists \text{ TM } \text{ s.t. } L(M) = \emptyset\}$

$L_{\neg e} = \{E(M) \mid \exists \text{ TM } \text{ s.t. } L(M) \neq \emptyset\}$

Hence: $L_e$ is the set of all strings that encode TMs that accept the empty language.

$L_{\neg e}$ is the complement of $L_e$.

We have that: $L_{\neg e}$ is R.E. but non-recursively.

$L_e$ is non-R.E.

Proof: see Exercise 10 (10.1, 10.2).
We have shown that a specific property of T.M. languages (namely non-emptiness) is undecidable.

This is just a special case of a much more general result:

All non-trivial properties of R.E. languages are undecidable.

**Property P**: of R.E. languages is a set of R.E. languages of the property of being context-free in the set of all CFLs.

The empty set \( \emptyset \) is the set \( \{ \emptyset \} \) consisting of only \( \emptyset \).

A property is trivial if either all or no R.E. language has it.

\[ \Rightarrow P \text{ is non-trivial if at least one R.E. language has } P \text{ and } \neg P \text{ does not have } P \]

**Note**: e T.M. cannot recognize a property (i.e. a set of languages) by checking if an input string is a language, because a language is typically infinite.

\[ \Rightarrow \text{ we consider instead a property } P \text{ as the language of the codes of those TMs that accept a language that satisfies } P \]

\[ L_P = \{ \langle M \rangle \mid L(M) \text{ has property } P \} \]

**Rice's Theorem**: every non-trivial property of R.E. languages is undecidable.

**Proof**: let \( P \) be a non-trivial property of R.E. languages.

Assume \( \neg P \) does not have \( P \) (otherwise, we can work with \( P \) ).
Since \( L \) is non-trivial, there is some \( L \subseteq P \) with \( L \neq \emptyset \). Let \( M_1 \) be s.t. \( L(M_1) = L \). \( \Rightarrow \mathcal{E}(M_1) \in L_0 \)

We show that \( L_0 \subseteq L_1 \):

**Reduction** \( R \) is an algorithm that:

- Takes as input a pair \( \langle \mathcal{E}(M), w \rangle \) instance of \( L_0 \)
- Produces a code \( \mathcal{E}(M_2) \) for a TM: \( M_2 \)

s.t. \( \langle \mathcal{E}(M), w \rangle \in L_0 \iff \mathcal{E}(M_2) \in L_0 \)

**Idea for \( M_2 \):

\[
\begin{array}{ccc}
w & \xrightarrow{\text{yes}} & \text{triggers} \\
\text{M} & & \\
\text{M}_1 & \xrightarrow{\text{yes}} & \\
\text{M}_2 & \xrightarrow{\text{yes}} &
\end{array}
\]

- \( M_2 \) ignores first \( \varepsilon \) own input \( \chi \), and writes \( w \) on tape 2
- Simulates \( M \) on \( w \) using an UTM (on tape 2)
- If \( M \) accepts \( w \), \( M_2 \) starts simulating \( M_1 \) on \( \chi \)
  and accepts if \( M_1 \) accepts \( \chi \)
  if \( M \) rejects \( w \) on does not halt, \( M_2 \) does the same.

**Note:** Since \( R \) takes as input \( \langle \mathcal{E}(M), w \rangle \), it can
  hack \( \varepsilon \) code \( \varepsilon \) into \( M_2 \).

We get that:

\[
\begin{align*}
\text{w \in L(M) & \Rightarrow L(M_2) = L(M_1) & \Rightarrow \mathcal{E}(M_2) \in L_0} \\
\text{w \notin L(M) & \Rightarrow L(M_2) = \emptyset & \Rightarrow \mathcal{E}(M_2) \notin L_0}
\end{align*}
\]

\[
\Rightarrow \langle \mathcal{E}(M), w \rangle \in L_0 \iff w \in L(M) \iff \mathcal{E}(M_2) \in L_0
\]

\[
\Rightarrow R \text{ reduces } L_0 \text{ to } L_0 \Rightarrow L_0 \text{ is undecidable. q.e.d.}
\]