Chapter 7
Uncomputability

7.1 Introduction

- Undecidability of concrete problems.
- First undecidable problem obtained by diagonalisation.
- Other undecidable problems obtained by means of the reduction technique.
- Properties of languages accepted by Turing machines.

7.2 Proving undecidability
Undecidability classes

Correspondence between a problem and the language of the encodings of its positive instances.

Definition
The decidability class R is the set of languages that can be decided by a Turing machine.

The class R is the class of languages (problems) that are

- decided by a Turing machine,
- recursive, decidable, computable,
- algorithmically solvable.

Definition
The decidability class RE is the set of languages that can be accepted by a Turing machine.

The class RE is the class of languages (problems) that are

- accepted by a Turing machine,
- partially recursive, partially decidable, partially computable,
- partially algorithmically solvable,
- recursively enumerable.

Lemma
The class R is contained in the class RE ($R \subseteq RE$)

A first undecidable language

A	w_0	w_1	w_2	$\ldots$	w_j	$\ldots$
M_0	Y	N	N	$\ldots$	Y	$\ldots$
M_1	N	N	Y	$\ldots$	Y	$\ldots$
M_2	Y	Y	N	$\ldots$	N	$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	
M_i	N	N	Y	$\ldots$	N	$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\ddots$

- $A[M_i, w_j] =$ Y (yes) if the Turing machine M_i accepts the word w_j ;

- $A[M_i, w_j] =$ N (no) if the Turing machine M_i does not accept the word w_j (loops or rejects the word).

$$L_0 = \{w | w = w_i \wedge A[M_i, w_i] = \mathsf{N}\}.$$

is not in the class RE.

A second undecidable language

Lemma
The complement of a language in the class R is also in the class R.

Lemma
If a language L and its complement $\overline{L}$ are both in the class RE, then both L and $\overline{L}$ are in R.

Three situations are thus possible:

1. L and $\overline{L} \in \mathsf{R}$,

2. $L \notin \mathsf{RE}$ and $\overline{L} \notin \mathsf{RE}$,

3. $L \notin \mathsf{RE}$ and $\overline{L} \in \mathsf{RE} \cap \overline{\mathsf{R}}$.

Lemma

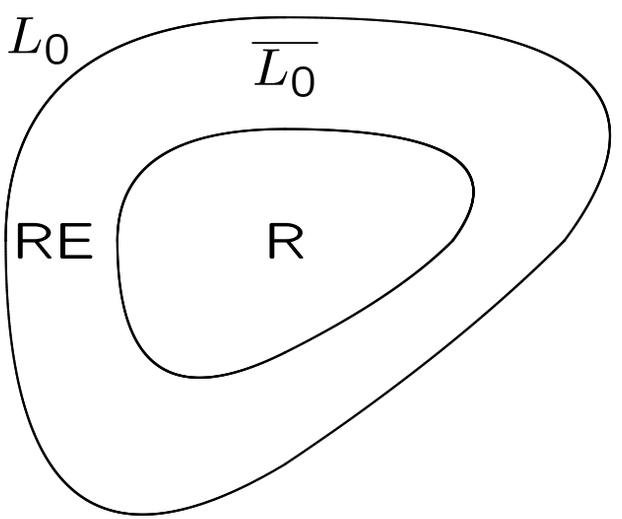
The language

$$\overline{L_0} = \{w | w = w_i \wedge M_i \text{ accepts } w_i\}$$

is in the class RE.

Theorem

The language $\overline{L_0}$ is undecidable (is not in R), but is in RE.



The reduction technique

1. One proves that, if there exists an algorithm that decides the language L_2, then there exists an algorithm that decides the language L_1. This is done by providing an algorithm (formally a Turing machine that stops on all inputs) that decides the language L_1, using as a sub-program an algorithm that decides L_2. This type of algorithm is called a *reduction*from L_1 to L_2. Indeed, it reduces the decidability of L_1 to that of L_2.

2. If L_1 is undecidable, one can conclude that L_2 is also undecidable ($L_2 \notin \mathsf{R}$). Indeed, the reduction from L_1 to L_2 establishes that if L_2 was decidable, L_1 would also be decidable, which contradicts the hypothesis that L_1 is an undecidable language.

The *universal language* UL

$$\text{UL} = \{< M, w > | \ M \text{ accepts } w\}$$

is undecidable.

Reduction from $\overline{L_0}$: to check if a word w is in $\overline{L_0}$, proceed as follows.

1. Find the value i such that $w = w_i$.

2. Find the Turing machine M_i.

3. Apply the decision procedure for UL to the wprd $< M_i, w_i >$: if the result is positive, w is accepted, if not it is rejected.

Note : $\overline{\text{UL}} \notin \text{RE}$

More undecidable problems

The halting problem

$$\mathsf{H} = \{< M, w > |\ M \text{ stops on } w\}$$

is undecidable. Reduction from UL.

1. Apply the algorithm deciding H to $< M, w >$.
2. If the algorithm deciding H gives the answer "no" (*i.e.* the machine M does not stop), answer "no" (in this case, we have indeed that $< M, w > \notin$ UL).
3. If the algorithm deciding H gives the answer "yes, simulate the execution of M on w and give the answer that is obtained (in this case, the execution of M on w terminates and one always obtains an answer).

The problem of determining if a program written in a commonly used programming language (for example C or, Java) stops for given input values is undecidable. This is proved by reduction from the halting problem for Turing machines.

1. Build a C program P that, given a Turing machine M and a word w, simulates the behaviour of M on w.

2. Decide if the program P stops for the input $< M, w >$ and use the result as answer.

The problem of deciding if a Turing machine stops when its input word is the empty word (*the empty-word halting problem*) is undecidable. This is proved by reduction from the halting problem.

1. For an instance $< M, w >$ of the halting problem, one builds a Turing machine M' that has the following behaviour:

   - it writes the word w on its input tape;
   - it then behaves exactly as M.

2. One solves the empty-word halting problem for M' and uses the result as answer.

The problem of deciding if a Turing machine stops for at least one input word (*the existential halting problem*) is undecidable. One proceeds by reduction from the empty-word halting problem.

1. For an instance M of the empty-word halting problem, one builds a Turing machine M'that behaves as follows:

   - it erases the content of its input tape;
   - it then behaves as M.

2. One solve the existential halting problem for M' and uses the result as answer.

The problem of deciding if a Turing machine stops for every input word (*the universal halting problem*) is undecidable. The reduction proceeds from the empty-word halting problem and is identical to the one used for the existential halting problem. The only difference is that one solves the universal halting problem for M', rather than the existential halting problem.

Determining if the language accepted by a Turing machine is empty (*empty accepted language*) is undecidable. Reduction from $\overline{\mathsf{UL}}$.

1. For an instance $< M, w >$ of $\overline{\mathsf{UL}}$, one builds a Turing machine M' that
   - simulates the execution of M on w ignoring its own input word;
   - if M accepts w, it accepts is input word, whatever it is.
   - if M does not accept w (rejects or has an infinite execution) it does not accept any word.
2. One solves the empty accepted language problem for M' and uses the result as answer.

This reduction is correct given that

- $L(M') = \emptyset$ exactly when M does not accept w, i.e., when $< M, w > \in \overline{\mathsf{UL}}$;

- $L(M') = \Sigma^* \neq \emptyset$ exactly when M accepts w, i.e. when $< M, w > \notin \overline{\mathsf{UL}}$.

Determining if the language accepted by a Turing machine is recursive (*recursive accepted language*) is undecidable. Reduction from $\overline{\text{UL}}$.

1. For an instance $< M, w >$ of $\overline{\text{UL}}$, one builds a Turing machine M' that
   - simulates the execution of M on w ignoring its own input word;
   - if M accepts w, it behaves on its own input word as a universal turing machine.
   - if M does not accept w (rejects or has an infinite execution) it does not accept any word.
2. One solves the *recursive accepted language problem* for M' and uses the result as answer.

This reduction is correct since

- $L(M') = \emptyset$ and is recursive exactly when M does not accept w, i. e. when $< M, w > \in \overline{\mathsf{UL}}$;

- $L(M') = \mathsf{UL}$ and is not recursive exactly when M accepts w, i.e. when $< M, w > \notin \overline{\mathsf{UL}}$.

Determining if the language accepted by a Turing machine is not recursive (undecidable) (*undecidable accepted language*) is undecidable. Reduction from $\overline{\text{UL}}$.

1. For an instance $< M, w >$ of $\overline{\text{UL}}$, one builds a Turing machine M' that

   - simulates the execution of M on w, without looking at its own input word x;
   - simultaneously (i.e. interleaving the executions), the machine M' simulates the universal Turing machine on its own input word x;
   - As soon as one of the executions accepts, (i.e., if M accepts w or if the input word is in UL), M' accepts.

2. If neither of the two executions accepts (i.e., if M does not accept w, or if the input word $x \notin$ UL), M' does not accept.

3. One solves the undecidable accepted language problem for M' and uses the result as answer.

This reduction is correct since

- $L(M') = \mathsf{UL}$ and is undecidable exactly when M does not accept w, i.e., when $< M, w > \in \overline{\mathsf{UL}}$;

- $L(M') = \Sigma^*$ and is decidable exactly when M accepts w, i.e. when $< M, w > \notin \overline{\mathsf{UL}}$.

In the preceding reductions, the language accepted by the machine M' is either UL, or $\emptyset$, or Σ^*. These proofs can thus also be used to establish that the problem of determining if the language accepted by a Turing machine is regular (or non regular) is undecidable. Indeed, $\emptyset$ and Σ^* are regular languages, whereas UL is not a regular language.

7.4 Properties of recursively enumerable languages

The recursively enumerable languages are :

- The languages computed by a Turing machine,
- the languages generated by a grammar,
- The languages that can be enumerated by an effective procedure (which explains why they are called "recursively enumerable").

The languages computed by a Turing machine

Definition

Let M be a Turing machine.If M stops on an input word u, let $f_M(u)$ be the word computed by M for u. The language computed by M is then the set of words

$$\{w \mid \exists u \text{ such that } M \text{ stops for } u \text{ and } w = f_M(u)\}.$$

Theorem

A language is computed by a Turing machine if and only if it is recursively enumerable (accepted by a Turing machine).

Let L be a language accepted by a Turing machine M. The Turing machine M' described below computes this language.

1. The machine M' first memorises its input word (one can assume that it uses a second tape for doing this).

2. Thereafter, it behaves exactly as M.

3. If M accepts, M' copies the memorised input word onto its tape.

4. If M does not accept, M' keeps running forever.

Let L be a language computed by a Turing machine M. The nondeterministic Turing machine described below accepts this language.

1. The machine M' first memorises its input word w.

2. Thereafter, it generates nondeterministically a word u.

3. The machine M' then simulates the behaviour of M on u.

4. If M stops on u, M' compares w to $f_M(u)$ and accepts w if $w = f_M(u)$.

5. If M does not stop on u, M' does not accept w.

The languages generated by a grammar

Theprem
A language is generated by a grammar if and only if it is recursively enumerable.

Let $G = (V, \Sigma, R, S)$, The Turing machine M described below accepts the language generated by G.

1. The machine M starts by memorising its input word (we can assume it uses a second tape to do so).
2. Then, it erases its tape and writes on it the start symbol S of the grammar.

3. The following cycle is then repeated :

   (a) nondeterministically, the machine chooses a rule R and a string appearing on its tape;

   (b) if the selected string is identical to the left-hand side of the rule, it is replaced by the right-hand side;

   (c) the content of the tape is compared to the memorised input word, and if they are identical the machine accepts; if not it carries on with its execution.

Let $M = (Q, \Gamma, \Sigma, \delta, s, B, F)$ be a Turing machine. One builds a grammar

$$G_0 = (V_{G_0}, \Sigma_{G_0}, R_{G_0}, S_{G_0})$$

such that $S_{G_0} \stackrel{*}{\Rightarrow} w$ with $w \in (Q \cup \Gamma)^*$ if and only if w describes a configuration (q, α_1, α_2) of M written as $\alpha_1 q \alpha_2$.

The grammar G_0 is defined by

- $V_{G_0} = Q \cup \Gamma \cup \{S_{G_0}, A_1, A_2\}$,
- $\Sigma_{G_0} = \Sigma$,
- R_{G_0} is the set of rules below.

1. Initial configuration of M :

$$\begin{array}{lcll} S_{G_0} & \to & sA_1 & \\ A_1 & \to & aA_1 & \forall a \in \Sigma \\ A_1 & \to & A_2 & \\ A_2 & \to & BA_2 & \\ A_2 & \to & \varepsilon. & \end{array}$$

2. Transitions. For all $p, q \in Q$ and $X, Y \in \Gamma$ such that

$$\delta(q, X) = (p, Y, R)$$

we include the rule

$$qX \to Yp.$$

Similarly, for all $p, q \in Q$ and $X, Y, Z \in \Gamma$ such that

$$\delta(q, X) = (p, Y, L)$$

we include the rule

$$ZqX \to pZY.$$

Problem: the input word is lost.

Solution: simulate a Turing machine with two tapes.

$G_1 = (V_{G_1}, \Sigma_{G_1}, R_{G_1}, S_{G_1})$ where

- $V_{G_1} = \Sigma \cup Q \cup ((\Sigma \cup \{e\}) \times \Gamma) \cup \{S_{G_1}, A_1, A_2\}$ (we represent an element of $((\Sigma \cup \{e\}) \times \Gamma)$ by a pair $[a, X]$),

- $\Sigma_{G_1} = \Sigma$,

- R_{G_1} is the set of rules described below.

1. Initial configuration of M :

$$\begin{array}{lcll} S_{G_1} & \rightarrow & sA_1 & \\ A_1 & \rightarrow & [a,a]A_1 & \forall a \in \Sigma \\ A_1 & \rightarrow & A_2 & \\ A_2 & \rightarrow & [e,B]A_2 & \\ A_2 & \rightarrow & \varepsilon. & \end{array}$$

2. Transitions. For all $p, q \in Q$, $X, Y \in \Gamma$ and $a \in \Sigma \cup \{e\}$ such that

$$\delta(q, X) = (p, Y, R)$$

we include the rule

$$q[a, X] \rightarrow [a, Y]p.$$

Similarly, for all $p, q \in Q$, $X, Y, Z \in \Gamma$ and $a, b \in \Sigma \cup \{e\}$ such that

$$\delta(q, X) = (p, Y, L)$$

we include the rule

$$[b, Z]q[a, X] \rightarrow p[b, Z][a, Y].$$

3. For all $q \in F$, $X \in \Gamma$ and $a \in \Sigma \cup \{e\}$, we include the rules

$$\begin{array}{rcl} q[a, X] & \to & qaq \\ [a, X]q & \to & qaq \end{array}$$

if $a \neq e$ and

$$\begin{array}{rcl} q[a, X] & \to & q \\ [a, X]q & \to & q \end{array}$$

if $a = e$. These rules propagate a copy of q next to each nonterminal $[a, X]$ and extract its first component. Finally, we add

$$q \to \varepsilon$$

that allows the copies of the state q to be removed.

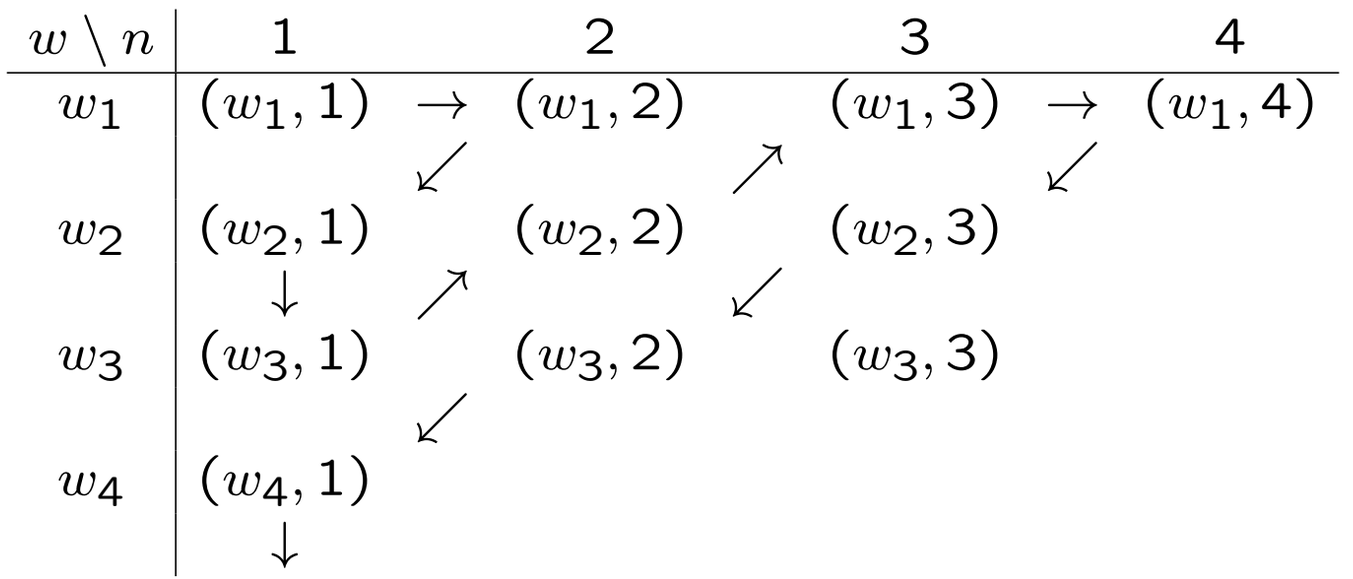
The languages enumerated by an effective procedure

Turing machine that enumerates the words accepted by M.

- Generate all words in lexicographical and increasing length order,
- simulate M on each newly generated word and keep this word only if it is accepted by M.

Incorrect: the Turing machine can have infinite executions.

Solution: other enumeration order.



- One considers the pairs (w, n) in the order of their enumeration.
- For each of these pairs, one simulates the execution of M on w, but limits the execution to n steps. On produces the word w if this execution accepts w.
- On then moves to the next pair (w, n).

7.5 Other undecidable problems

The problem of determining if a word w is in the language generated by a grammar G is undecidable.

Reduction from the problem UL. Let $< M, w >$ be an instance of the problem UL. It can be solved as follows:

1. one builds the grammar G generating the language accepted by M

2. one determines if $w \in L(G)$ and uses the result as answer.

The problem of deciding if two grammars G_1 and G_2 generate the same language is undecidable.

Reduction from the membership problem for the language generated by a grammar. An instance $< w, G >$ of this problem can be solved as follows:

1. Let $G = (V, \Sigma, R, S)$. One builds the grammars $G_1 = G$ and $G_2 = (V, \Sigma, R', S')$, with
$$R' = R \cup \{S' \to S, S' \to w\}.$$
2. One checks if $L(G_1) = L(G_2)$ and uses the result as answer.

One has indeed that $L(G_2) = L(G_1) \cup \{w\}$ and thus that $L(G_2) = L(G_1)$ if and only if $w \in L(G)$.

The problem of determining *validity in the predicate calculus* is undecidable

The problem of determining the *universality of a context-free language*, i.e., the problem of determining if for a context-free grammar G one has $L(G) = \Sigma^*$ is undecidable.

The problem of determining the *emptiness of the intersection of context-free languages* is undecidable.

The problem is to determine if, for two context-free grammars G_1 and G_2, one has $L(G_1) \cap L(G_2) = \emptyset$.

Hilbert's tenth problem is undecidable. This problem is to determine if an equation

$$p(x_1, \ldots, x_n) = 0$$

where $p(x_1, \ldots, x_n)$ is an integer coefficient polynomial, has an integer solution.

Noncomputable functions

A total function

$$f : \Sigma_1^* \to \Sigma_2^*$$

is computable if and only if the following questions are decidable.

1. Given $n \in N$ and $w \in \Sigma_1^*$, do we have that $|f(w)| > n$?

2. Given $k \in N$, $w \in \Sigma_1^*$ and $a \in \Sigma_2$, do we have that $f(w)_k = a$? (is the k^{th} letter of $f(w)$ a?).

The situation is similar in the case of a partial function. A function

$$f : \Sigma_1^* \to \Sigma_2^*$$

is a partially computable function if and only if the following conditions are satisfied.

1. Checking if for a given word w, $f(w)$ is defined is partially decidable.

2. For $n \in N$ and $w \in \Sigma_1^*$ such that $f(w)$ is defined, checking if $|f(w)| > n$ is decidable.

3. For $k \in N$, $a \in \Sigma_2$ and $w \in \Sigma_1^*$ such that $f(w)$ is defined, checking if $f(w)_k = a$ is decidable.