Aalto University
School of Science

CS-C2160 Theory of Computation

Lecture 9: Decidability and Undecidability

Pekka Orponen
Aalto University
Department of Computer Science

Topics:

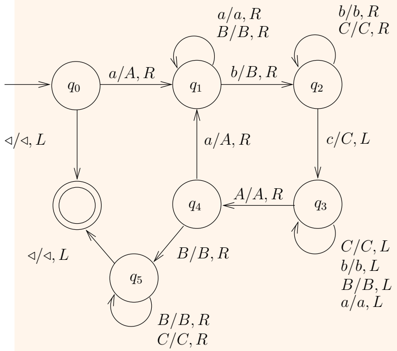
- Decidable and semi-decidable languages and problems
- Background: countable and uncountable sets
- Universal Turing machines and undecidable problems

Material:

- In Finnish: Sections 6.1, 6.2, 1.7, 6.3 and 6.4 in the Finnish lecture notes
- In English: Sections 4.1–4.2 in the Sipser book and these slides

Recap

Example: A Turing machine for the language $\{a^k b^k c^k \mid k \geq 0\}$



Computation on input $aabbcc$:

$$
\begin{aligned}
&(q_0, \underline{a}abbcc) \vdash (q_1, A\underline{a}bbcc) \vdash \\
&(q_1, Aa\underline{b}bcc) \vdash (q_2, AaB\underline{b}cc) \vdash \\
&(q_2, AaBb\underline{c}c) \vdash (q_3, AaB\underline{b}Cc) \vdash \\
&(q_3, Aa\underline{B}bCc) \vdash (q_3, A\underline{a}BbCc) \vdash \\
&(q_3, \underline{A}aBbCc) \vdash (q_4, A\underline{a}BbCc) \vdash \\
&(q_1, AA\underline{B}bCc) \vdash (q_1, AAB\underline{b}Cc) \vdash \\
&(q_2, AABB\underline{C}c) \vdash (q_2, AABBC\underline{c}) \vdash \\
&(q_3, AABB\underline{C}C) \vdash (q_3, AAB\underline{B}CC) \vdash \\
&(q_3, AA\underline{B}BCC) \vdash (q_3, A\underline{A}BBCC) \vdash \\
&(q_4, AA\underline{B}BCC) \vdash (q_5, AAB\underline{B}CC) \vdash \\
&(q_5, AABB\underline{C}C) \vdash (q_5, AABBC\underline{C}) \vdash \\
&(q_5, AABBCC\underline{\lhd}) \vdash \\
&(q_{\text{acc}}, AABBC\underline{C}\lhd).
\end{aligned}
$$

Decidable and Semi-Decidable Languages and Problems

9.1 Turing-recognisable and Turing-decidable languages

- The *Church-Turing thesis:* Any (strong enough) computing model $\equiv$ Turing machines.
- *Computability theory:* The study of what can be, and especially what cannot be computed with Turing machines ($\equiv$ computer programs).
- *Important distinction:* Machines (programs) that always halt and those that don't.

Definition 9.1

A Turing machine

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$$

is *total* if it halts on all input strings. A language A is

- *Turing-recognisable* (historically also called *recursively enumerable*) if it can be recognised with some Turing machine.
- *Turing-decidable* (historically also called *recursive*) if it can be recognised by some *total* Turing machine.

An alternative point of view:

- Recall the connection between languages and decision problems (binary input-output relations): the language A_Π corresponding to a decision problem Π consists of those inputs x for which the answer is “yes” (binary output 1) in the decision problem Π.
- The decision problem Π is
  - *decidable* if its language A_Π is Turing-decidable, and
  - *semi-decidable* if A_Π is Turing-recognisable.
  - A problem that is not decidable is called *undecidable*.
    (Note: An undecidable problem may still be semi-decidable.)
- In other words, a decision problem is (i) decidable if it has an algorithm that solves it correctly and always terminates, and (ii) semi-decidable if it has an algorithm that solves it correctly but may not terminate on some “no” instances.
- In the following, we will use this terminology also for languages, viz. decidable $\equiv$ Turing-decidable, semi-decidable $\equiv$ Turing-recognisable.

Example:

The language

$$\{a^n b^n c^n \mid n \geq 0\}$$

over the alphabet $\{a, b, c\}$ corresponds to the decision problem:

> *Given a string x over the alphabet $\{a, b, c\}$. Is x of form $a^n b^n c^n$ for some $n \geq 0$?*

Example:

The language

$$\{n\#p\#q \mid n,p,q \in \{0,1\}^* \text{ are binary numbers and } n = pq\}$$

over the alphabet $\{0,1,\#\}$ corresponds to the decision problem:

> *Given a string x over the alphabet $\{0,1,\#\}$. Is x of form $n\#p\#q$, where $n,p,q \in \{0,1\}^*$, and $n = pq$ holds if we interpret n,p,q is binary numbers?*

This can be expressed more informally as:

> *Given binary numbers n,p,q. Does it hold that $n = pq$?*

Example:

The decision problem

Given a binary number n. Is n a prime?

is represented by the language

$$\{n \in \{0,1\}^* \mid n \text{ is a prime number written in binary}\}$$

over the alphabet $\{0,1\}$.

Example:

The decision problem

> *Given a multivariate polynomial P.*
> *Does P have integer roots?*

can be expressed as the language

$$\{P \mid P \text{ is a multivariate polynomial having an integer root}\}.$$

The alphabet for P could be for instance the standard ASCII alphabet, in which e.g. the string `x1^3*x2-7*x1` could represent the polynomial $x_1^3x_2 - 7x_1$.

Example:

The language

$$\{p \mid p \text{ is a UTF-8 encoded string representing a Python program whose execution terminates on all possible inputs}\}$$

over the “byte alphabet” $\{0\text{x}00, 0\text{x}01, \ldots, 0\text{xff}\}$ corresponds to the decision problem:

Given a Python program p. Does the execution of p terminate on all inputs?

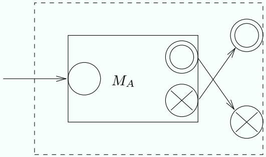
9.2 Basic properties of decidable and semi-decidable languages ($\sim$ problems)

Lemma 9.1

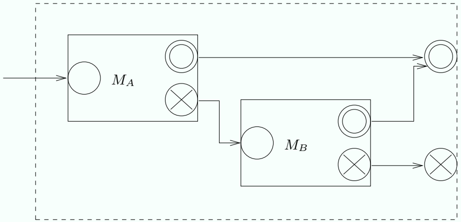
Let $A, B \subseteq \Sigma^*$ be decidable languages. Then $\bar{A} = \Sigma^* - A$, $A \cup B$, and $A \cap B$ are decidable languages as well.

Proof

(i) Let M_A be a total Turing machine with $\mathcal{L}(M_A) = A$. We obtain a total Turing machine $M_{\bar{A}}$ recognising $\bar{A}$ simply by swapping the accept and reject states of M_A:



(ii) Let M_A and M_B be total Turing machines with $\mathcal{L}(M_A) = A$ and $\mathcal{L}(M_B) = B$. We obtain a total Turing machine M recognising the language $A \cup B$ by sequential composition of M_A and M_B: if M_A accepts the input, then M also accepts; if M_A rejects the input, then execute M_B on the input string.



(iii) $A \cap B = \overline{\bar{A} \cup \bar{B}}$.

Lemma 9.2

Let $A, B \subseteq \Sigma^*$ be semi-decidable languages. Then $A \cup B$ and $A \cap B$ are semi-decidable languages, too.

Proof

$A \cap B$ as in Lemma 9.1 (ii), and $A \cup B$ with a construction similar to that in Lemma 9.3 (left as an exercise).

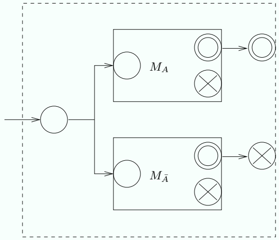
Lemma 9.3

A language $A \subseteq \Sigma^*$ is decidable if and only if both languages A and $\bar{A}$ are semi-decidable.

Proof

By Lemma 9.1(i), if A is decidable, then $\bar{A}$ is also decidable and hence both A and $\bar{A}$ are semi-decidable as well.

We next show that if A and $\bar{A}$ are both semi-decidable, then A is decid-able.



Let M_A and $M_{\bar{A}}$ be Turing machines recognising the languages A and $\bar{A}$, respectively. Then for every $x \in \Sigma^*$, *either* M_A *or* $M_{\bar{A}}$ *halts* and accepts x.

We build a 2-tape Turing machine M by combining M_A and $M_{\bar{A}}$ "in parallel": on tape 1, M simulates machine M_A, and on tape 2 it simulates machine $M_{\bar{A}}$.

If the simulation on tape 1 halts in an accepting configuration, then M accepts the input. If the tape 2 simulation accepts, then M rejects the input.

Corollary 9.4

Let $A \subseteq \Sigma^*$ be a semi-decidable language that is not decidable. Then the language $\bar{A}$ is not semi-decidable.

Background: Countable and Uncountable Sets

9.3 Countable and uncountable sets

Definition 9.2

- A set X is *countably infinite* if there is a bijection $f : \mathbb{N} \to X$.
- A set is *countable* if it is finite or countably infinite.
- A set that is not countable is called *uncountable*.

- Intuitively, a set X is countable if its elements can be ordered and indexed with natural numbers:
  - $X = \{x_0, x_1, x_2, \ldots, x_{n-1}\}$ if X is a finite set with n elements, and
  - $X = \{x_0, x_1, x_2, \ldots\}$ if X is countably infinite.

- Every subset of a countable set is also countable (proof left as an exercise). On the other hand, uncountable sets have both countable and uncountable subsets. Thus uncountable sets are, in some sense, “larger” than countable sets.

Lemma 9.5

Let Σ be a finite alphabet. The set Σ^* of all strings over Σ is countably infinite.

Proof

We construct a bijection $f : \mathbb{N} \to \Sigma^*$ as follows. Let $\Sigma = \{a_1, a_2, \ldots, a_n\}$. We fix some arbitrary "alphabetical order" for the symbols in Σ, for instance, $a_1 < a_2 < \cdots < a_n$.

The strings in Σ^* can now be enumerated in shortlex order with respect to the chosen alphabetical order:

- We first enumerate strings of length 0 (i.e., ε), then those of length 1 (i.e., $a_1, a_2, \ldots, a_n$), then those of length 2, and so on.
- Inside each length group, the strings are enumerated in lexicographic order with respect to the chosen alphabetical order.

The bijection f is thus

$$\begin{aligned}
0 &\mapsto \varepsilon \\
1 &\mapsto a_1 \\
2 &\mapsto a_2 \\
&\vdots \\
n &\mapsto a_n \\
n+1 &\mapsto a_1a_1 \\
n+2 &\mapsto a_1a_2 \\
&\vdots \\
2n &\mapsto a_1a_n
\end{aligned}$$

$$\begin{aligned}
2n+1 &\mapsto a_2a_1 \\
&\vdots \\
3n &\mapsto a_2a_n \\
&\vdots \\
n^2+n &\mapsto a_na_n \\
n^2+n+1 &\mapsto a_1a_1a_1 \\
n^2+n+2 &\mapsto a_1a_1a_2 \\
&\vdots
\end{aligned}$$

- Note that in fact all programs written in any programming language are just strings over the respective base alphabet (ASCII in C, Unicode allowed in some other languages).
- By Lemma 9.5, the set of such strings is countably infinite, and therefore the set of all possible programs in any programming language is countable as well.
- We next prove that the family of all languages ($\sim$ decision problems) over any alphabet is uncountable.
- Thus there are more decision problems than there are possible computer programs!
  $\Rightarrow$ *It is impossible, in any programming language, to write a decision program for every problem.*

Theorem 9.6

The family of all languages over any alphabet Σ is uncountable.

Proof (Cantor's diagonal argument)

Denote the family $\mathcal{P}(\Sigma^*)$ of all languages over Σ by $\mathcal{A}$. Suppose that these languages could be enumerated in some order, say

$$\mathcal{A} = \{A_0, A_1, A_2, \ldots\}.$$

Let the set of all strings over Σ, enumerated in shortlex order, be $\Sigma^* = \{x_0, x_1, x_2, \ldots\}$. Using these orders, define a language $\tilde{A}$ as

$$\tilde{A} = \{x_i \in \Sigma^* \mid x_i \notin A_i\}.$$

Since $\tilde{A}$ belongs to the family $\mathcal{A}$, and we assumed that the languages in $\mathcal{A}$ can be enumerated, it must be the case that $\tilde{A} = A_k$ for some $k \in \mathbb{N}$. But now, according to the definition of $\tilde{A}$, we obtain the contradiction

$$x_k \in \tilde{A} \Leftrightarrow x_k \notin A_k = \tilde{A}.$$

Thus the assumption that $\mathcal{A}$ could be enumerated is wrong and $\mathcal{A}$ is uncountable.

$\tilde{A}$

$$
\begin{array}{c|ccccc}
\searrow & A_0 & A_1 & A_2 & A_3 & \cdots \\
\hline
x_0 & \overset{1}{\cancel{0}} & 0 & 0 & 1 & \cdots \\
x_1 & 0 & \overset{0}{\cancel{1}} & 0 & 0 & \cdots \\
x_2 & 1 & 1 & \overset{0}{\cancel{1}} & 1 & \cdots \\
x_3 & 0 & 0 & 0 & \overset{1}{\cancel{0}} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}
$$

Graphically, the idea of the proof can be illustrated as follows:

- We form the “incidence matrix” of the languages $A_0, A_1, A_2, \ldots$ and the strings $x_0, x_1, x_2, \ldots$.
- The cell at row i, column j has value: 1 if $x_i \in A_j$, 0 if $x_i \notin A_j$.
- Now the language $\tilde{A}$ differs from every language A_k on the “diagonal” of the matrix.

Universal Turing Machines and Undecidable Problems

What is a universal Turing machine?

- In “interpreted” computer programming languages such as Python or Java, programs are not compiled into machine code but executed (simulated) by a systems program called an *interpreter* (cf. Python interpreter, Java virtual machine).
- Similarly a “universal Turing machine” can execute (simulate) any other Turing machine, given its description as an input string.
- In modern terminology, a universal Turing machine is thus an interpreter for the “Turing machines” programming language, written in the same language.

9.4 An encoding for Turing machines

- Without loss of generality, we consider standard, deterministic 1-tape Turing machines whose input alphabet is $\Sigma = \{0, 1\}$.
- Each such machine

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}})$$

  can be encoded into a binary string.
- The idea:
  - fix some ordering for the states and the tape alphabet, and
  - use unary coding for encoding the transitions.

- Number the states and the tape alphabet so that
  - $Q = \{q_0, q_1, \ldots, q_n\}$ with $q_{\text{acc}} = q_{n-1}$ and $q_{\text{rej}} = q_n$
  - $\Gamma \cup \{\triangleright, \triangleleft\} = \{a_0, a_1, \ldots, a_m\}$ with $a_0 = 0$, $a_1 = 1$, $a_2 = \triangleright$, and $a_3 = \triangleleft$
- Furthermore, let $\Delta_0 = L$ and $\Delta_1 = R$.
- The code for transition function entry $\delta(q_i, a_j) = (q_r, a_s, \Delta_t)$ is:

$$c_{ij} = 0^{i+1}10^{j+1}10^{r+1}10^{s+1}10^{t+1}.$$

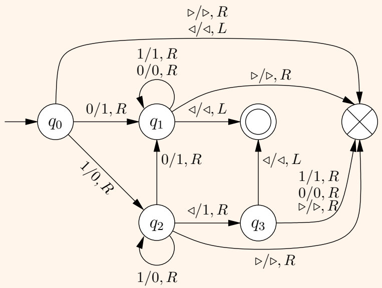
- The code for the whole machine M is:

$$\begin{aligned} c_M \quad = \quad & 111c_{00}11c_{01}11\ldots11c_{0m}11c_{10}11\ldots11c_{1m}11 \\ & \ldots11c_{n-2,0}11\ldots11c_{n-2,m}111. \end{aligned}$$

- *Note:* From the code of a machine it is easy to algorithmically deduce the number of states as well as the size of the tape alphabet.

Example:

Consider the following machine M ($\sim$ machine `succ` from Lecture 8).



$\triangleright/\triangleright, R$
$\triangleleft/\triangleleft, L$
$1/1, R$
$0/0, R$
$\triangleright/\triangleright, R$
$0/1, R$
$\triangleleft/\triangleleft, L$
q_0
q_1
$1/0, R$
$0/1, R$
$\triangleleft/\triangleleft, L$
$1/1, R$
$0/0, R$
$\triangleright/\triangleright, R$
$\triangleleft/1, R$
q_2
q_3
$\triangleright/\triangleright, R$
$1/0, R$

$Q = \{q_0, q_1, q_2, q_3\}$
$\Sigma = \{0, 1\}$
$\Gamma = \{0, 1\}$

Then $c_M = 111c_{0,0}11c_{0,1}11c_{0,2}11c_{0,3}11c_{1,0}11\ldots11c_{3,3}111$, where

$$\begin{aligned}
\delta(q_0, 0) = (q_1, 1, R): &\qquad c_{0,0} = 010100100100 \\
\delta(q_0, 1) = (q_2, 0, R): &\qquad c_{0,1} = 0100100010100 \\
\delta(q_0, \triangleright) = (q_{\text{rej}}, \triangleright, R): &\qquad c_{0,2} = 0100010000001000100 \\
\delta(q_0, \triangleleft) = (q_{\text{rej}}, \triangleleft, L): &\qquad c_{0,3} = 01000010000001000010 \\
\vdots &\qquad \vdots
\end{aligned}$$

9.5 A language that is not semi-decidable

- Previously we showed how to associate each Turing machine M to a binary string c_M encoding it.
- In reverse, we can associate each binary string c to a Turing machine M_c.
- Some binary strings do not encode any Turing machine in the way described above — to such strings, we associate some trivial Turing machine M_{triv} that *rejects all input strings*.
- That is, we define:

$$M_c = \begin{cases} \text{the machine } M \text{ for which } c_M = c \text{ if } c \text{ is a valid encoding,} \\ \text{the machine } M_{\text{triv}} \text{ otherwise.} \end{cases}$$

- As a result, we obtain:
  - an enumeration of all Turing machines (with input alphabet $\{0,1\}$), and also
  - an enumeration of all semi-decidable languages over the alphabet $\{0,1\}$.
- The machines are:

$$M_\varepsilon, M_0, M_1, M_{00}, M_{01}, \ldots$$

(Indices in shortlex order.)

- The semi-decidable languages are:

$$\mathcal{L}(M_\varepsilon), \mathcal{L}(M_0), \mathcal{L}(M_1), \mathcal{L}(M_{00}), \mathcal{L}(M_{01}), \ldots$$

Each language can appear multiple times in this list, as different machines may recognise the same language.

Lemma 9.7

The "diagonal language"

$$D = \{c \in \{0,1\}^* \mid c \notin \mathcal{L}(M_c)\}$$

is not semi-decidable.

Proof (Cantor-style diagonal argument)

Suppose that D is semi-decidable; then $D = \mathcal{L}(M)$ for some Turing machine M. Let d be the binary encoding of M, i.e. $D = \mathcal{L}(M_d)$. Now

$$d \in D \quad \Leftrightarrow \quad d \notin \mathcal{L}(M_d) = D.$$

From the contradiction, we deduce that the assumption must be wrong and thus D is not semi-decidable.

- The decision problem corresponding to the language D is: “Given a binary string c, is it the case that the Turing machine associated to c does not accept the string c?” More natural examples of undecidable languages will be seen later.
- A graphical view of language D: if the characteristic functions of the languages $\mathcal{L}(M_\varepsilon)$, $\mathcal{L}(M_0)$, $\mathcal{L}(M_1)$, $\ldots$ are represented as an infinite array, then the language D is the one that is obtained by “flipping” the language obtained from the diagonal:

D $\searrow$	$\mathcal{L}(M_\varepsilon)$	$\mathcal{L}(M_0)$	$\mathcal{L}(M_1)$	$\mathcal{L}(M_{00})$	$\cdots$
ε	$\cancel{0}^1$	0	0	0	$\cdots$
0	0	$\cancel{1}^0$	1	0	$\cdots$
1	0	0	$\cancel{1}^0$	1	$\cdots$
00	0	0	0	$\cancel{0}^1$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

9.6 A universal language and universal Turing machines

- Let us define a *universal language* U [1] (over the binary alphabet $\{0,1\}$) as

$$U = \{c_M w \mid \text{Turing machine } M \text{ accepts string } w\}.$$

- The corresponding decision problem is:

  Given a Turing machine M and a string w.
  Does M accept the string w?

- If A is a semi-decidable language and M is a Turing machine recognising A, then

$$A = \{w \in \{0,1\}^* \mid c_M w \in U\}.$$

- The language U itself is semi-decidable, too. The machines recognising U are called *universal Turing machines*.

[1] Language A_{TM} in Sipser's book.

Theorem 9.8

The language U is semi-decidable.

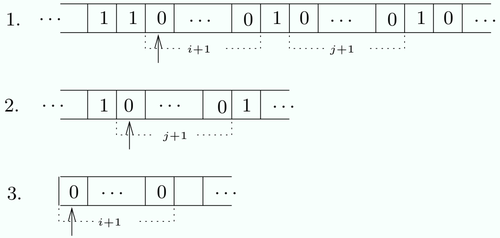
Proof

It is easiest to describe a Turing machine M_U recognising U in the 3-tape machine model. This can then be transformed into a standard 1-tape machine as explained in Lecture 8.

In the beginning, the input is placed on tape 1 as usual. After this, the machine works as follows:

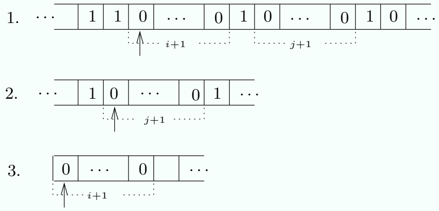
1. First, M_U checks whether the input is of form cw, where c is a valid code for a Turing machine. If the input is not of that form, M_U rejects it (and thus works as expected for M_{triv}). Otherwise, it copies the binary input sub-string $w = a_1 a_2 \dots a_k \in \{0,1\}^*$ to tape 2 in the unary form

$$00010^{a_1+1}10^{a_2+1}1\dots10^{a_k+1}10000.$$



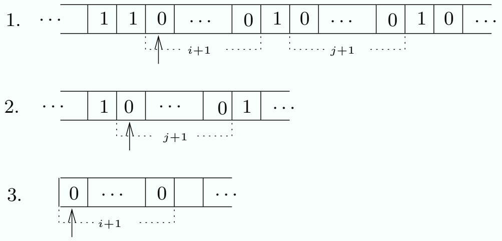
2. Now it is known that the input is of form cw, where $c = c_M$ for some machine M and M_U has to check whether M accepts w.

   To do this, M_U keeps the description c of M on tape 1, uses tape 2 to simulate the tape of M (in unary coding), and stores the current simulated state of M on tape 3 in the unary form $q_i \sim 0^{i+1}$ (in the beginning, M_U thus writes the code 0 for the state q_0 on tape 3).



3. After the initial preparations, M_U works in phases and simulates the action of one transition of M in each phase.

   In the beginning of each phase, M_U searches the place on the description of M on tape 1 that corresponds to the current simulated state q_i (tape 3) and symbol a_j under the simulated tape head (tape 2).
   Let that place on the tape 1 be $0^{i+1}10^{j+1}10^{r+1}10^{s+1}10^{t+1}$.
   Now M_U replaces the sub-string 0^{i+1} on tape 3 with 0^{r+1} and the sub-string 0^{j+1} on tape 2 with 0^{s+1}, and moves the tape head on tape 2 one simulated symbol left if $t = 0$ and right if $t = 1$.



If the description on tape 1 does not contain a place corresponding to the simulated state q_i, the simulated machine M has reached the accept or the reject state.

Thus $i = k+1$ or $i = k+2$, where q_k is the last state having transitions encoded in the description of M, and machine M_U enters the state q_{acc} or q_{rej}, correspondingly.

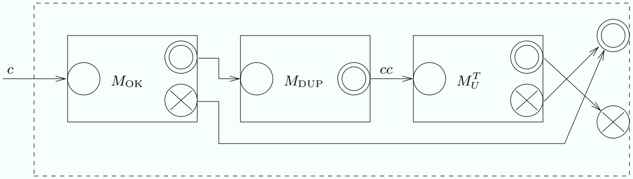
Theorem 9.9

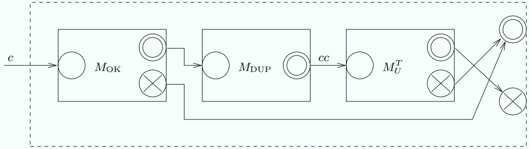
The language U is not decidable.

Proof

Suppose that U would be decidable. Then there is a total Turing machine M_U^T recognising U. We can now build a total Turing machine M_D that recognises the diagonal language D of Lemma 9.7 as follows.

Let M_{OK} be a total Turing machine that checks whether a string is a valid code for a Turing machine. Similarly, let M_{DUP} be a total Turing machine that transforms a string c into the string cc. We build the machine M_D by combining the machines M_U^T, M_{OK} and M_{DUP}:





Now clearly M_D is total whenever the machine M_U^T is, and M_D recognises D:

$$\begin{aligned} c \in \mathcal{L}(M_D) &\Leftrightarrow c \notin \mathcal{L}(M_{\mathsf{OK}}) \vee cc \notin \mathcal{L}(M_U^T) \\ &\Leftrightarrow c \notin \mathcal{L}(M_c) \\ &\Leftrightarrow c \in D. \end{aligned}$$

But by Theorem 9.7, the language D is not decidable. We have thus obtained a contradiction, and our assumption that U is decidable must be wrong.

Corollary 9.10

The language

$$\tilde{U} = \{c_M w \mid w \notin \mathcal{L}(M)\}$$

is not semi-decidable.

Proof

The language $\tilde{U}$ is effectively the same as the complement $\bar{U}$ of the universal language U. More precisely, $\bar{U} = \tilde{U} \cup \text{ERR}$, where ERR is the (obviously) decidable language

$$\text{ERR} = \{x \in \{0,1\}^* \mid x \text{ does not start with a valid code for a Turing machine}\}.$$

If the language $\tilde{U}$ were semi-decidable, then so would the language $\bar{U}$. But as the language U is semi-decidable, it would follow (by Lemma 9.3) that U is decidable. This contradicts Theorem 9.7 and therefore we must conclude that $\tilde{U}$ is not semi-decidable.