Aalto University
School of Science

# CS-C2160 Theory of Computation

Lecture 10: More on Undecidability

Pekka Orponen
Aalto University
Department of Computer Science

Spring 2021

---

Topics:

- Undecidability of some Turing machine properties
  - The halting problem
  - The non-emptiness problem
- Other undecidability results
  - Undecidability in logic and algebra
  - Post's correspondence problem
  - The Chomsky hierarchy
- Computable functions and reductions
  - Computable functions
  - Reductions between languages
- * Excursion: Post's correspondence problem

Material:

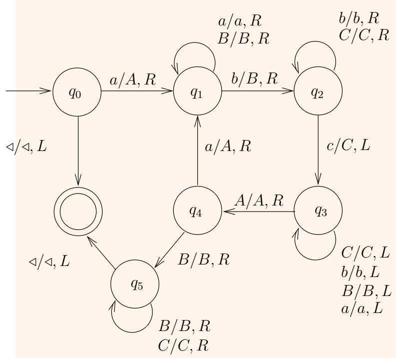
- In Finnish: Sections 6.5, 6.7 (beginning) and 6.9–6.10 (reductions) in the Finnish lecture notes
- In English: Sections 5.1–5.3 in the Sipser book



---

## Recap from previous lectures

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




Computation on the input $aabbcc$:

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



---

- 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

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

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

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

- The "*universal language*" (over the binary alphabet $\{0, 1\}$):

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

- The corresponding decision problem is:

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

- Language $U$ is semi-decidable. Turing machines that recognise ("semi-decide") $U$ are called *universal Turing machines*.
- On the other hand, $U$ is not decidable ...
- ... meaning that there is no Turing machinethat could *always* decide, given another Turing machine $M$ and an input $w$, whether $M$ accepts the input $w$.



## Undecidability of Some Turing Machine Properties



# 10.1 The halting problem

**Theorem 10.1**

The language

$$H = \{c_M w \mid \text{Turing machine } M \text{ halts on input } w\}$$

is semi-decidable but not decidable.

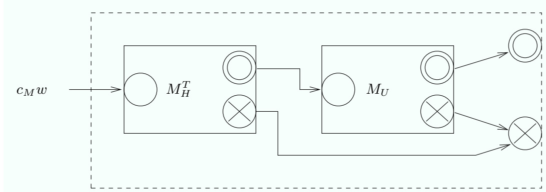
**Proof**

Let us first verify that $H$ is semi-decidable. It is easy to modify the universal Turing machine $M_U$ presented in the proof of Theorem 9.8 to a Turing machine that, on input $c_M w$ simulates the computation of machine $M$ on input $w$ and accepts if and only if the simulated computation halts (in either reject or accept state).



We next show that $H$ is not decidable. Suppose that it were and that $H = \mathcal{L}(M_H^T)$ for some total Turing machine $M_H^T$. Suppose that $M_H^T$ is such that when it halts, it leaves its original input on the tape (possibly extended with blank symbols). Let $M_U$ be the universal Turing machine designed in the proof of Theorem 9.8.

We could now design a *total* Turing machine recognising $U$ by combining the machines $M_H^T$ and $M_U$ is follows:




```
def U(m,w):
  wcopy = w
  if not M_H(m,w):
    reject
  return M_U(m,wcopy)
```

But according to Theorem 9.9 such a total Turing machine recognising $U$ cannot exist. This contradiction means that our assumption must be wrong and $H$ is not decidable.

## Corollary 10.2

The language

$$\tilde{H} = \{c_M w \mid M \text{ does } \textit{not} \text{ halt on the input } w\}$$

is not semi-decidable.



# 10.2 The non-emptiness problem

- Consider the following *non-emptiness problem* for Turing machines:

  *Given a Turing machine $M$.*
  *Does $M$ accept any string?*

- This problem corresponds to the formal language:

$$\mathsf{NE} = \{c \in \{0,1\}^* \mid \mathcal{L}(M_c) \neq \emptyset\}.$$

## Theorem 10.3

The language NE is semi-decidable but not decidable.



- We again establish the proof in two parts: semi-decidability and undecidability.

## Lemma 10.4

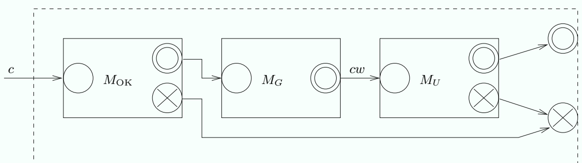
The language NE is semi-decidable.

## Proof

- We prove the claim by designing a Turing machine $M_{\mathsf{NE}}$ that recognises the language.
- The designed machine $M_{\mathsf{NE}}$ is nondeterministic.
- We use the following "sub-machines":
  - $M_{\mathsf{OK}}$ tests whether the input is a valid Turing machine code.
  - $M_G$ nondeterministically writes an arbitrary binary string $w$ at the end of the current tape contents.



- We design $M_{\mathsf{NE}}$ by combining the machines $M_{\mathsf{OK}}$, $M_G$, and the universal Turing machine $M_U$ as follows:




The idea as a nondeterministic "Python" program:

```
def NE(m):
    if not M_ok(m):
        return False
    w = choose_string_nondeterministically()
    return M_U(m,w)
```

Clearly

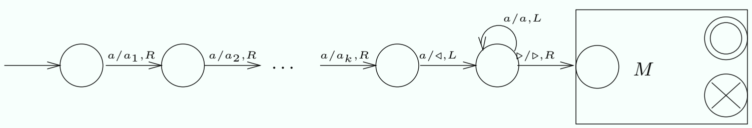
$$\begin{aligned} c \in \mathcal{L}(M_{\mathsf{NE}}) &\Leftrightarrow c \text{ is a valid TM code and } \exists w \text{ s.t. } cw \in U \\ &\Leftrightarrow c \text{ is a valid TM code and } \exists w \text{ s.t. } w \in \mathcal{L}(M_c) \\ &\Leftrightarrow \mathcal{L}(M_c) \neq \emptyset. \end{aligned}$$

## Lemma 10.5

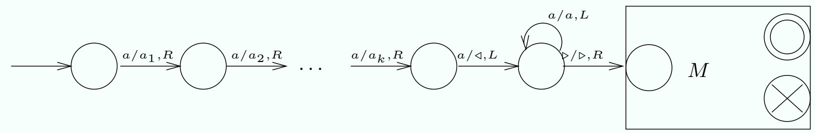
The language NE is not decidable.

## Proof

- Suppose that NE were decidable and let $M_{\text{NE}}^T$ be a total Turing machine recognising it. By using $M_{\text{NE}}^T$, we design a total Turing machine $M_U^T$ recognising the language $U$ and thus obtain a contradiction.
- The design is based on coding input strings as "constant strings" in Turing machines.
- Let $M$ be a Turing machine whose behaviour on an input $w = a_1 a_2 \ldots a_k$ we wish to study.
- Let $M^w$ be the machine that always replaces its own "real" input with the string $w$ and then behaves like $M$:






The behaviour of the machine $M^w$ does thus not depend on its real input at all, but it either accepts or rejects *all* strings depending on how $M$ behaves on $w$:

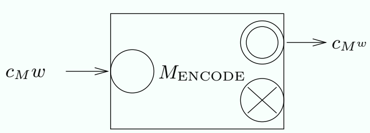
$$\mathcal{L}(M^w) = \begin{cases} \{0,1\}^* & \text{if } w \in \mathcal{L}(M) \\ \emptyset & \text{if } w \notin \mathcal{L}(M) \end{cases}$$

A Python program corresponding to $M^w$:

```
def mw(x):
    # w is a constant
    w = '010...01'
    return m(w)
```



- Now let $M_{\text{ENCODE}}$ be a Turing machine that, given a string $c_M w$ consisting of the code $c_M$ of a Turing machine $M$ and a binary string $w$, writes the code $c_{M^w}$ of the above described machine $M^w$ on the tape and halts:



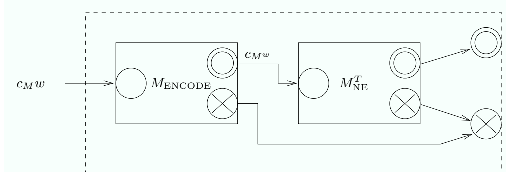

As Python:

```
def encode(m,w):
    if not isValidTM(m):
        return False
    return asTM("""
def mw(x):
w = '"""+w+"""'
return """+m+"""(w)
""")
```

- If the input is not of form $cw$ for a valid TM code $c$, the machine $M_{\text{ENCODE}}$ halts in the reject state.
- Thus the machine $M_{\text{ENCODE}}$ operates on the codes of Turing machines: the code of a machine $M$ is extended with transitions and the numbering of the states is changed so that the result is the code of the machine $M^w$.



- By combining the machine $M_{\text{ENCODE}}$ and the hypothetical total machine $M_{\text{NE}}^T$, we could now build a total Turing machine $M_U^T$ recognising $U$ as follows:




```
def MTU(m,w):
    mw = encode(m,w)
    if mw == False:
        return False
    return MTNE(mw)
```

- The machine $M_U^T$ is total as $M_{\text{NE}}^T$ is, and $\mathcal{L}(M_U^T) = U$ because

$$c_M w \in \mathcal{L}(M_U^T) \Leftrightarrow c_{M^w} \in \mathcal{L}(M_{\text{NE}}^T) = \text{NE} \Leftrightarrow \mathcal{L}(M^w) \neq \emptyset \Leftrightarrow w \in \mathcal{L}(M).$$

- But the language $U$ is not decidable, and thus such a total Turing machine $M_U^T$ recognising $U$ cannot exist.
- From the contradiction we deduce that the language NE cannot be recognised by any total Turing machine $M_{\text{NE}}^T$, and is thus not decidable.

## Other Undecidability Results



## 10.3 Undecidability in logic and algebra

**Theorem 10.6 (Undecidability of FO logic; Church/Turing 1936)**

There is no algorithm that, given a formula $\phi$ in first-order logic, decides whether the formula is valid (i.e. true in all possible interpretations).

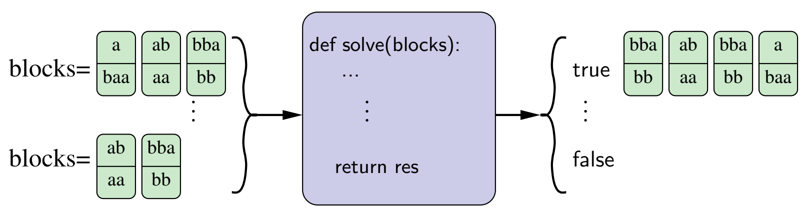
**Theorem 10.7 ("Hilbert's tenth problem"; Matijasevitsh/Davis/Robinson/Putnam 1953–70)**

There is no algorithm that, given a multivariate polynomial $P(x_1,\ldots,x_n)$ with integer coefficients, decides whether the polynomial has integer-valued zero points (i.e. tuples $(m_1,\ldots,m_n) \in \mathbf{Z}^n$ for which $P(m_1,\ldots,m_n) = 0$). The problem is undecidable already when $n = 15$ or $\deg(P) = 4$.



## 10.4 Post's correspondence problem

Given a finite set of domino block types (we can have arbitrarily many blocks of each type), can we have a finite sequence of blocks so that the upper and lower rows contain the same string?




**Theorem 10.8**

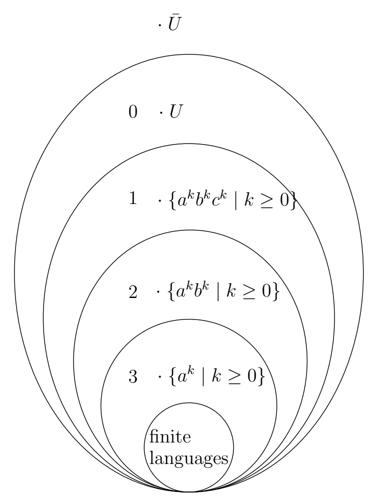
Post's correspondence problem is undecidable.

**Proof**

In Section 10.8.



## 10.5 The Chomsky hierarchy




A classification of grammars, languages generated by grammars and recogniser automata classes:[a]

**Type-0:** unrestricted grammars / semi-decidable languages / Turing machines

**Type-1:** context-sensitive grammars / context-sensitive languages / linear bounded automata

**Type-2:** context-free grammars / context-free languages / pushdown automata

**Type-3:** right and left linear grammars / regular languages / finite automata

[a] Type 0 and Type 1 grammars at Lecture 11.

## Undecidability in the Chomsky hierarchy

The decidability and undecidability of some problems related to grammars, when given grammars $G$ and $G'$ of type $i$ in Chomsky hierarchy and a string $w$. The abbreviations mean $D \sim$ "decidable", $U \sim$ "undecidable", $T \sim$ "always true".

| | Type $i$: | | | |
|---|---|---|---|---|
| Problem: is... | 3 | 2 | 1 | 0 |
| $w \in \mathcal{L}(G)$? | $D$ | $D$ | $D$ | $U$ |
| $\mathcal{L}(G) = \emptyset$? | $D$ | $D$ | $U$ | $U$ |
| $\mathcal{L}(G) = \Sigma^*$? | $D$ | $U$ | $U$ | $U$ |
| $\mathcal{L}(G) = \mathcal{L}(G')$? | $D$ | $U$ | $U$ | $U$ |
| $\mathcal{L}(G) \subseteq \mathcal{L}(G')$? | $D$ | $U$ | $U$ | $U$ |
| $\mathcal{L}(G) \cap \mathcal{L}(G') = \emptyset$? | $D$ | $U$ | $U$ | $U$ |
| $\mathcal{L}(G)$ regular? | $T$ | $U$ | $U$ | $U$ |
| $\mathcal{L}(G) \cap \mathcal{L}(G')$ of type $i$? | $T$ | $U$ | $T$ | $T$ |
| $\overline{\mathcal{L}(G)}$ of type $i$? | $T$ | $U$ | $T$ | $U$ |



## Computable Functions and Reductions



- In this Section we discuss:
- How to use Turing machines to compute more complicated functions than just yes/no answers.
- How to use Turing-computable *reductions between languages* ($\sim$ decision problems) to establish undecidability.



## 10.6 Computable functions

- We define the *partial function*

$$f_M : \Sigma^* \to \Gamma^*$$

*computed* by a Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ as:

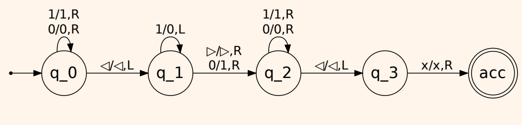
$$f_M(x) = \begin{cases} u, \text{ if } (q_0, \underline{x}) \underset{M}{\vdash^*} (q, u\underline{a}v) \text{ where } q \in \{q_{\text{acc}}, q_{\text{rej}}\}, av \in \Gamma^*; \\ \text{undefined, if } M \text{ does not halt on input } x. \end{cases}$$

A partial function $f : \Sigma^* \to A$ is:

- *partially computable* (historically: *partially recursive*) if it can be computed by some Turing machine, and
- *computable* (historically: *recursive*) if it can be computed by some *total* Turing machine.
- *Note:* We could equivalently define that a partially computable function $f$ is computable if its value $f(x)$ is defined for all $x$.

**Example:**

A Turing machine that computes the successor (modulo $2^n$) of an $n$-bit binary number (in the most significant bit first presentation)

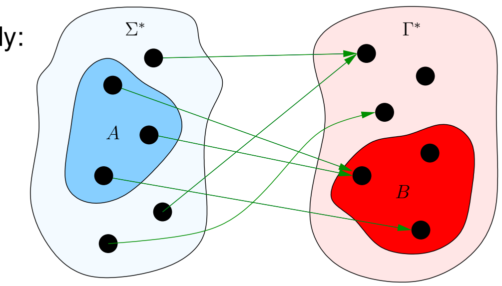



- By the Church-Turing thesis, all total functions that can be computed by computer programs are also Turing-machine computable.



## 10.7 Reductions between languages

- A language $A \subseteq \Sigma^\star$ can be *(computably) reduced* to a language $B \subseteq \Gamma^\star$, denoted as
$$A \leq_m B,$$
if there is a computable function $f : \Sigma^\star \to \Gamma^\star$ such that
$$x \in A \quad \Leftrightarrow \quad f(x) \in B, \quad \text{for all } x \in \Sigma^\star.$$
Such a function is called a *(computable many-one) reduction* from $A$ to $B$.
- Graphically:






- Reductions can be used to translate solution methods (∼recognising/deciding automata) from one problem to another.

**Lemma 10.8**

If $A \leq_m B$ and $B$ is a decidable language, then $A$ is decidable as well.

**Proof**

Let $M_B$ be a total Turing machine recognising language $B$, and $M_f$ a total Turing machine that computes the reduction $f$ from language $A$ to language $B$.

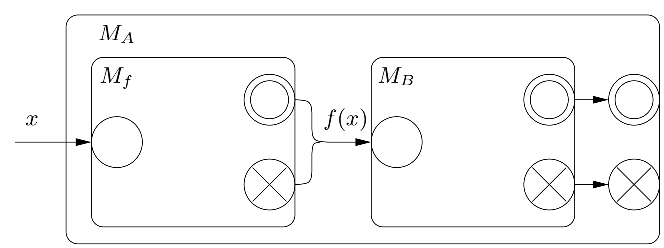
We can combine machines $M_f$ and $M_B$ into a total Turing machine $M_A$ recognising $A$ as follows: On input string $w$,

- first compute the value $f(w)$ using $M_f$, and
- then run machine $M_B$ on input string $f(w)$.

The combined machine $M_A$ is clearly total and accepts input string $w$ if and only if $f(w) \in B$ , i.e. $w \in A$.



- Graphically, the total Turing machine $M_A$ recognising language $A$ can be illustrated as below, where:
  - $M_f$ is the total Turing machine computing reduction $f$, and
  - $M_B$ is the total Turing machine recognising $B$.



- The idea in Python:

```
def solveB(y):
    """Returns true iff y \in B."""
    ...
    return result

def f(z):
    """Returns a string z' such that z \in A <=> z' \in B."""
    ...
    return result

def solveA(x):
    """Returns true iff x \in A."""
    inputForB = f(x)
    return solveB(inputForB)
```



**Example:**

Consider the decision problem DFA-EQ:

> *Given a pair of deterministic finite automata $(M_1, M_2)$ over an alphabet $\Sigma$. Do $M_1$ and $M_2$ recognise the same language?*

This problem is decidable, since we can minimise the automata and check whether the results are the same (up to a renaming of the states).

Consider then the decision problem REX-COMP:

> *Given two regular expressions $r_1$ and $r_2$ over an alphabet $\Sigma$. Is $r_1$ equivalent to the complement of $r_2$, i.e., is $\mathcal{L}(r_1) = \Sigma^* \setminus \mathcal{L}(r_2)$?*

Also this is decidable, since we can reduce it to problem DFA-EQ:

- Given a pair $(r_1, r_2)$ of regular expressions.
- The reduction produces a pair $(M_1, M_2)$ of DFA, where:
  - $M_1$ is a DFA for the language $\mathcal{L}(r_1)$, and
  - $M_2$ is the "state-complement" of a DFA for the language $\mathcal{L}(r_2)$.



- By using reductions, we can also prove that some languages are *not* decidable:

**Corollary 10.9**

If $A \leq_m B$ and $A$ is not decidable, then $B$ is not decidable.

**Proof**

Assume that $A \leq_m B$ and that $A$ is not decidable.

Now if $B$ were decidable, then (by Lemma 10.8) also $A$ should be decidable, which would be a contradiction.

- Showing that a language $B$ is undecidable:
  - Choose a previously-known undecidable language $A$.
  - Design a reduction from language $A$ to language $B$.
  - Conclude by Corollary 10.9 that $B$ is undecidable as well.



**Example: Proving undecidability of the halting problem by reduction**

- We design a reduction mapping $f$ from the undecidable universal language $U$ to the "halting language" $H$.
- Given an arbitrary input $c_M w$ to problem $U$, the reduction $f$ produces a string $f(c_M w) = c_{M'} w'$ with the property that:

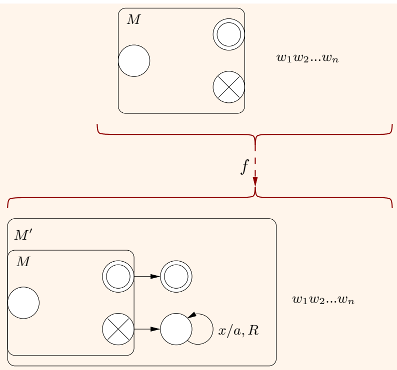
$$c_M w \in U \Leftrightarrow c_{M'} w' \in H.$$

  In other words, the reduction will satisfy:

  *$M$ accepts input $w$ if and only if $M'$ halts on input $w'$*

- ☞ If we can solve the problem "Does $M'$ halt on input $w'$", we can also solve the problem "Does $M$ accept input $w$".
- ☞ As the language $U$ is undecidable and we can reduce it to language $H$, language $H$ must also be undecidable (Cor 10.9)

- More concretely, the reduction $f$ produces:
  - The code $c_{M'}$ for a machine $M'$ that equals $M$ except that the reject state of $M$ is replaced by a state in which the computation never terminates.
  - The string $w'$ simply as $w' = w$.




- Now:
  - $M$ accepts input $w \Rightarrow M'$ halts on input $w'$
  - $M$ rejects input $w \Rightarrow M'$ does not halt on $w'$
  - $M$ does not halt on input $w \Rightarrow M'$ does not halt on $w'$
- ☞ $M$ accepts input $w$ if and only if $M'$ halts on input $w'$.



## Example: Proving undecidability of the non-emptiness problem by reduction

- We design a reduction mapping $f$ from the undecidable universal language $U$ to the "non-emptiness" language NE.
- Given an arbitrary input $c_M w$ to problem $U$, the reduction $f$ produces a string $f(c_M w) = c_{M'}$, with the property that:

$$c_M w \in U \Leftrightarrow c_{M'} \in \text{NE}.$$

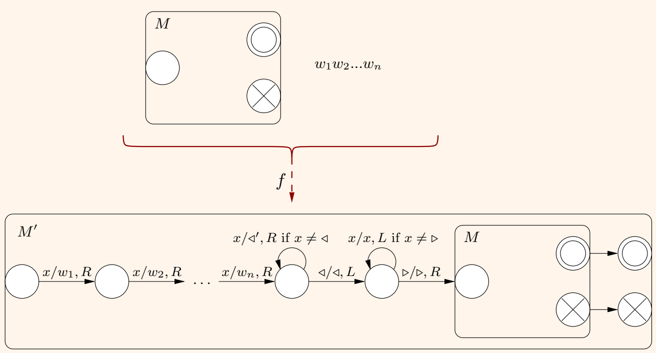
  In other words, the reduction will satisfy:

  *$M$ accepts input string $w$ if and only if $M'$ accepts some input string*

- ☞ If we can solve the problem "Does $M'$ accept *some* input string", we can also solve the problem "Does $M$ accept input string $w$".
- ☞ As the language $U$ is undecidable and we can reduce it to language NE, language NE must also be undecidable (Cor 10.9)



- The reduction function $f$ produces a (code for a) machine $M'$ similarly as in the previous example:
  - $M'$ first overwrites its own input with the (constant) string $w$, and
  - then operates as $M$ would.[a]




- ☞ $M$ accepts input string $w \Rightarrow M'$ accepts all input strings
- ☞ $M$ does not accept $w \Rightarrow M'$ does not accept any input string
- ☞ $M$ accepts input string $w \Leftrightarrow M'$ accepts some input string.

[a] The machine $M$ is modified so that it works on the symbol $\lhd'$ as it would on $\lhd$



## * Excursion: Post's Correspondence Problem

## 10.8 Post's correspondence problem

- An undecidable problem with a simple definition
- A *domino* block is a pair $(t,b)$ of strings, graphically $\boxed{\frac{t}{b}}$

  Here $t$ is the top row and $b$ the bottom row of the domino
- Given a finite set $P$ of dominos, a *match* is a finite sequence $D_1 D_2 \ldots D_n$ of dominos in $P$ such that the top and bottom rows of the sequence contain the same string
- Note that the a domino may occur many times in a match!

**Example:**

For the domino set $\left\{\boxed{\frac{b}{ca}}, \boxed{\frac{a}{ab}}, \boxed{\frac{ca}{a}}, \boxed{\frac{abc}{c}}\right\}$, there is a match $\boxed{\frac{a}{ab}}\boxed{\frac{b}{ca}}\boxed{\frac{ca}{a}}\boxed{\frac{a}{ab}}\boxed{\frac{abc}{c}}$.



**Definition 10.1 (Post's Correspondence Problem)**

Given a finite set $P = \left\{\boxed{\frac{t_1}{b_1}}, \ldots, \boxed{\frac{t_n}{b_n}}\right\}$ of dominos. Does $P$ have a match?
As a language: PCP $= \{P \mid P$ is a domino set that has a match$\}$.

**Example:**

The domino set $\left\{\boxed{\frac{b}{ca}}, \boxed{\frac{a}{ab}}, \boxed{\frac{ca}{a}}, \boxed{\frac{abc}{c}}\right\}$ is in the language PCP as it has a match $\boxed{\frac{a}{ab}}\boxed{\frac{b}{ca}}\boxed{\frac{ca}{a}}\boxed{\frac{a}{ab}}\boxed{\frac{abc}{c}}$.



**Example:**

The domino set $\left\{\boxed{\frac{ab}{aa}}, \boxed{\frac{bba}{bb}}\right\}$ is not in the language PCP as it has no matches.



**Theorem 10.8**

PCP is undecidable.

We prove this in two parts:

1. We reduce the undecidable language $U$ to the "modified Post's correspondence problem" MPCP (defined in a while)
   ☞ MPCP is undecidable
2. We reduce the undecidable language MPCP to the language PCP.
   ☞ PCP is undecidable, too.

## Definition 10.2 MPCP

Given a domino set $P = \left\{\boxed{\begin{matrix} t_1 \\ \hline b_1 \end{matrix}}, \ldots, \boxed{\begin{matrix} t_n \\ \hline b_n \end{matrix}}\right\}$ and a *start domino* $\boxed{\begin{matrix} t_1 \\ \hline b_1 \end{matrix}} \in P$.

Does $P$ has a match that starts with the start domino?

## Lemma 10.11

$U \leq_m$ MPCP.

## Proof

- A sketch, a bit more detailed version is presented in section 5.2 of Sipser's book
- Given an input $c_M w$, i.e., a Turing machine $M$ and a string $w$, for which we wish to find out if $c_m w \in U$
- We design a domino set $P$ and a start domino $\boxed{\begin{matrix} t_1 \\ \hline b_1 \end{matrix}} \in P$ such that

  $M$ accepts the string $w$ if and only if the set $P$ has a match starting with the start domino

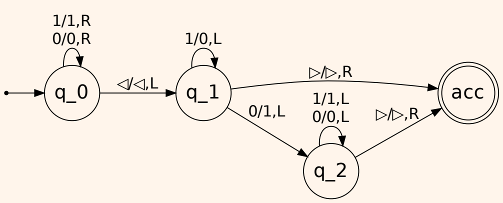


- The idea: the computation (i.e., sequence of configurations) of $M$ on $w$ can be described as a sequence of strings of form $\alpha q \beta \triangleleft$, where
  - $\alpha$ gives the symbols on the left of the tape head,
  - $q$ is the state of the machine, and
  - $\beta$ gives the symbols below and on the right of the the tape head
- Separate these strings with a special symbol # and start with a special symbol $\blacktriangleright$
- In addition, extend the sequence with "configurations" in which the symbols next to the accept state $q_{\text{acc}}$ can be removed one-by-one and require that finally the "configuration" consists only of the state $q_{\text{acc}}$.



## Example:

The computation of the Turing machine




on the input $10$ seen as an above described sequence:

$\blacktriangleright\#\triangleright q_0 10\triangleleft\#\triangleright 1 q_0 0\triangleleft\#\triangleright 10 q_0\triangleleft\#\triangleright 1 q_1 0\triangleleft\#\triangleright q_2 11\triangleleft\# q_2\triangleright 11\triangleleft\#\triangleright q_{\text{acc}} 11\triangleleft$
$\#\triangleright q_{\text{acc}} 1\triangleleft\#\triangleright q_{\text{acc}}\triangleleft\#\triangleright q_{\text{acc}}\# q_{\text{acc}}\#\#$



- We design a domino set whose matches are strings of the above described form
- The start domino contains the start configuration of $M$ on $w$ on the bottom row

## Example:

For the Turing machine above and the input $10$, we make the start domino $\boxed{\begin{matrix} \blacktriangleright\# \\ \hline \blacktriangleright\#\triangleright q_0 10\triangleleft\# \end{matrix}}$

- In order to be able to copy symbols further away from the tape head to the successor configuration on the bottom row, we make dominos of form $\left[\frac{x}{x}\right]$ for all tape symbols and the special symbol #

**Example:**

For the Turing machine above, we make the dominos

$\left[\frac{0}{0}\right] \left[\frac{1}{1}\right] \left[\frac{\triangleleft}{\triangleleft}\right] \left[\frac{\triangleright}{\triangleright}\right] \left[\frac{\#}{\#}\right]$



- We next build dominos that
  - make the top row match the bottom row, and
  - at the same time construct the successor configuration on the bottom row
- For each transition moving the tape head to the right, $\delta(q,a) = (r,b,R)$, we make the domino $\left[\frac{q\,a}{b\,r}\right]$ and for the transition $\delta(q,\triangleleft) = (r,b,R)$ the domino $\left[\frac{q\,a}{b\,r\,\triangleleft}\right]$

**Example:**

For the Turing machine above, we make the dominos

$\left[\frac{q_2 \triangleright}{\triangleright\, q_{\mathrm{acc}}}\right] \left[\frac{q_1 \triangleright}{\triangleright\, q_{\mathrm{acc}}}\right] \left[\frac{q_0\, 1}{1\, q_0}\right] \left[\frac{q_0\, 0}{0\, q_0}\right]$



- Similarly, for the transitions $\delta(q,a) = (r,b,L)$ moving the tape head left, we make the domino $\left[\frac{c\,q\,a}{r\,c\,b}\right]$ for every symbol $c$ (excluding $\triangleleft$ which would be redundant)

**Example:**

For the Turing machine above, we make the dominos

$\left[\frac{0\,q_2\,1}{q_2\,0\,1}\right] \left[\frac{1\,q_2\,1}{q_2\,1\,1}\right] \left[\frac{\triangleright\,q_2\,1}{q_2\,\triangleright\,1}\right] \left[\frac{0\,q_2\,0}{q_2\,0\,0}\right] \left[\frac{1\,q_2\,0}{q_2\,1\,0}\right] \left[\frac{\triangleright\,q_2\,0}{q_2\,\triangleright\,0}\right] \left[\frac{0\,q_0\,\triangleleft}{q_1\,0\,\triangleleft}\right] \left[\frac{1\,q_0\,\triangleleft}{q_1\,1\,\triangleleft}\right] \left[\frac{\triangleright\,q_0\,\triangleleft}{q_1\,\triangleright\,\triangleleft}\right]$

$\left[\frac{0\,q_1\,0}{q_2\,0\,1}\right] \left[\frac{1\,q_1\,0}{q_2\,1\,1}\right] \left[\frac{\triangleright\,q_1\,0}{q_2\,\triangleright\,1}\right] \left[\frac{0\,q_1\,1}{q_1\,0\,0}\right] \left[\frac{1\,q_1\,1}{q_1\,1\,0}\right] \left[\frac{\triangleright\,q_1\,1}{q_1\,\triangleright\,0}\right]$



- The new “configurations” at the end of the string, in which the accept state is already reached and we must remove the other symbols, can be produced with the dominos of form $\left[\frac{x\,q_{\mathrm{acc}}}{q_{\mathrm{acc}}}\right]$ and $\left[\frac{q_{\mathrm{acc}}\,x}{q_{\mathrm{acc}}}\right]$, where $x$ is a tape symbol

**Example:**

For the Turing machine above, we make the dominos

$\left[\frac{0\,q_{\mathrm{acc}}}{q_{\mathrm{acc}}}\right] \left[\frac{q_{\mathrm{acc}}\,0}{q_{\mathrm{acc}}}\right] \left[\frac{1\,q_{\mathrm{acc}}}{q_{\mathrm{acc}}}\right] \left[\frac{q_{\mathrm{acc}}\,1}{q_{\mathrm{acc}}}\right] \left[\frac{q_{\mathrm{acc}}\,\triangleleft}{q_{\mathrm{acc}}}\right] \left[\frac{\triangleright\,q_{\mathrm{acc}}}{q_{\mathrm{acc}}}\right]$

- Finally we ensure that the string ends in the “configuration” consisting only of the accept state by the domino $\left[\dfrac{q_{acc}\#\#}{\#}\right]$
- This is the only domino that can fix the inbalance in the number of # symbols in the top and bottom rows caused by the start domino



**Example:**

For the Turing machine above and the input $10$ we thus produced the following dominos:

$\left[\dfrac{\blacktriangleright\#}{\blacktriangleright\#\triangleright q_0 10\triangleleft\#}\right]$ $\left[\dfrac{q_2\triangleright}{\triangleright q_{acc}}\right]$ $\left[\dfrac{q_1\triangleright}{\triangleright q_{acc}}\right]$ $\left[\dfrac{q_0 1}{1 q_0}\right]$ $\left[\dfrac{q_0 0}{0 q_0}\right]$ $\left[\dfrac{0 q_2 1}{q_2 0 1}\right]$ $\left[\dfrac{1 q_2 1}{q_2 1 1}\right]$ $\left[\dfrac{\triangleright q_2 1}{q_2 \triangleright 1}\right]$

$\left[\dfrac{0 q_2 0}{q_2 0 0}\right]$ $\left[\dfrac{1 q_2 0}{q_2 1 0}\right]$ $\left[\dfrac{\triangleright q_2 0}{q_2 \triangleright 0}\right]$ $\left[\dfrac{0 q_0 \triangleleft}{q_1 0 \triangleleft}\right]$ $\left[\dfrac{1 q_0 \triangleleft}{q_1 1 \triangleleft}\right]$ $\left[\dfrac{\triangleright q_0 \triangleleft}{q_1 \triangleright \triangleleft}\right]$ $\left[\dfrac{0 q_1 0}{q_2 0 1}\right]$ $\left[\dfrac{1 q_1 0}{q_2 1 1}\right]$ $\left[\dfrac{\triangleright q_1 0}{q_2 \triangleright 1}\right]$

$\left[\dfrac{0 q_1 1}{q_1 0 0}\right]$ $\left[\dfrac{1 q_1 1}{q_1 1 0}\right]$ $\left[\dfrac{\triangleright q_1 1}{q_1 \triangleright 0}\right]$ $\left[\dfrac{0}{0}\right]$ $\left[\dfrac{1}{1}\right]$ $\left[\dfrac{\triangleleft}{\triangleleft}\right]$ $\left[\dfrac{\triangleright}{\triangleright}\right]$ $\left[\dfrac{\#}{\#}\right]$ $\left[\dfrac{0 q_{acc}}{q_{acc}}\right]$ $\left[\dfrac{q_{acc} 0}{q_{acc}}\right]$ $\left[\dfrac{1 q_{acc}}{q_{acc}}\right]$ $\left[\dfrac{q_{acc} 1}{q_{acc}}\right]$

$\left[\dfrac{q_{acc}\triangleleft}{q_{acc}}\right]$ $\left[\dfrac{\triangleright q_{acc}}{q_{acc}}\right]$ $\left[\dfrac{q_{acc}\#\#}{\#}\right]$



**Example:**

A match:

$\left[\dfrac{\blacktriangleright\#}{\blacktriangleright\#\triangleright q_0 10\triangleleft\#}\right]$ $\left[\dfrac{\triangleright}{\triangleright}\right]$ $\left[\dfrac{q_0 1}{1 q_0}\right]$ $\left[\dfrac{0}{0}\right]$ $\left[\dfrac{\triangleleft}{\triangleleft}\right]$ $\left[\dfrac{\#}{\#}\right]$ $\left[\dfrac{\triangleright}{\triangleright}\right]$ $\left[\dfrac{1}{1}\right]$ $\left[\dfrac{q_0 0}{0 q_0}\right]$ $\left[\dfrac{\triangleleft}{\triangleleft}\right]$ $\left[\dfrac{\#}{\#}\right]$ $\left[\dfrac{\triangleright}{\triangleright}\right]$ $\left[\dfrac{1}{1}\right]$ $\left[\dfrac{0 q_0 \triangleleft}{q_1 0 \triangleleft}\right]$

$\left[\dfrac{\#}{\#}\right]$ $\left[\dfrac{\triangleright}{\triangleright}\right]$ $\left[\dfrac{1 q_1 0}{q_2 1 1}\right]$ $\left[\dfrac{\triangleleft}{\triangleleft}\right]$ $\left[\dfrac{\#}{\#}\right]$ $\left[\dfrac{\triangleright q_2 1}{q_2 \triangleright 1}\right]$ $\left[\dfrac{1}{1}\right]$ $\left[\dfrac{\triangleleft}{\triangleleft}\right]$ $\left[\dfrac{\#}{\#}\right]$ $\left[\dfrac{q_2\triangleright}{\triangleright q_{acc}}\right]$ $\left[\dfrac{1}{1}\right]$ $\left[\dfrac{1}{1}\right]$ $\left[\dfrac{\triangleleft}{\triangleleft}\right]$ $\left[\dfrac{\#}{\#}\right]$ $\left[\dfrac{\triangleright}{\triangleright}\right]$ $\left[\dfrac{q_{acc} 1}{q_{acc}}\right]$

$\left[\dfrac{1}{1}\right]$ $\left[\dfrac{\triangleleft}{\triangleleft}\right]$ $\left[\dfrac{\#}{\#}\right]$ $\left[\dfrac{\triangleright}{\triangleright}\right]$ $\left[\dfrac{q_{acc} 1}{q_{acc}}\right]$ $\left[\dfrac{\triangleleft}{\triangleleft}\right]$ $\left[\dfrac{\#}{\#}\right]$ $\left[\dfrac{\triangleright}{\triangleright}\right]$ $\left[\dfrac{q_{acc}\triangleleft}{q_{acc}}\right]$ $\left[\dfrac{\#}{\#}\right]$ $\left[\dfrac{\triangleright q_{acc}}{q_{acc}}\right]$ $\left[\dfrac{\#}{\#}\right]$ $\left[\dfrac{q_{acc}\#\#}{\#}\right]$

The string on the top and bottom rows is:
$\blacktriangleright\#\triangleright q_0 10\triangleleft\#\triangleright 1 q_0 0\triangleleft\#\triangleright 10 q_0\triangleleft\#\triangleright 1 q_1 0\triangleleft\#\triangleright q_2 11\triangleleft\# q_2\triangleright 11\triangleleft\#\triangleright q_{acc} 11\triangleleft$
$\#\triangleright q_{acc} 1\triangleleft\#\triangleright q_{acc}\triangleleft\#\triangleright q_{acc}\# q_{acc}\#\#$



- We now only have to show how to get rid of the start domino requirement in MPCP, i.e., prove the following:

**Lemma 10.12**

MPCP $\leq_m$ PCP.

**Proof**

- Let $u = u_1 \ldots u_n$ be a non-empty string (that is, $n \geq 1$).
- Define the following notation:

$$\begin{aligned} \star u &= \star u_1 \star u_2 \ldots \star u_n \\ u\star &= u_1 \star u_2 \ldots \star u_n \star \\ \star u \star &= \star u_1 \star u_2 \ldots \star u_n \star \end{aligned}$$

where $\star$ is a new, unused symbol

- Given a domino set $P = \left\{ \boxed{\frac{t_1}{b_1}}, \boxed{\frac{t_2}{b_2}}, \ldots, \boxed{\frac{t_n}{b_n}} \right\}$ and a start domino $\boxed{\frac{t_1}{b_1}}$, we make a domino set

$$P' = \left\{ \boxed{\frac{\star t_1}{\star b_1 \star}}, \boxed{\frac{\star t_2}{b_1 \star}} \ldots, \boxed{\frac{\star t_n}{b_n \star}}, \boxed{\frac{\star \diamond}{\diamond}} \right\}$$

  where $\diamond$ is a new symbol and $\boxed{\frac{\star \diamond}{\diamond}}$ enables the introduction of the last $\star$ symbol in the top row
- Clearly $\boxed{\frac{\star t_1}{\star b_1 \star}}$ is now the only domino that can start a match
- ... and for each original MPCP match (and only for them) is a new PCP match in which every second symbol is $\star$.



- More on algorithms and reductions in courses:
  - CS-E3190 Principles of Algorithmic Techniques
  - CS-E4530 Computational Complexity Theory