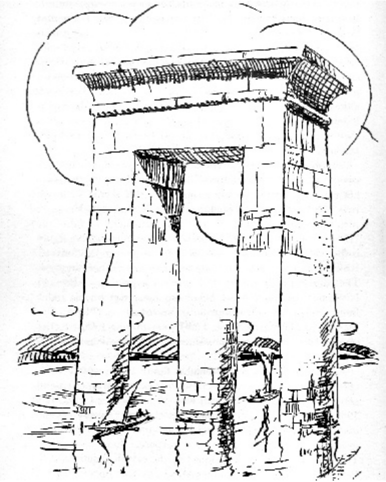
Automata, Languages and Computation

Chapter 9: Undecidability

Master Degree in Computer Engineering
University of Padua
Lecturer: Giorgio Satta

Lecture based on material originally developed by:
Gösta Grahne, Concordia University

Undecidability



Recursively enumerable languages

From now onward: Modern computers = Turing machines.

A language L is **recursively enumerable** (RE) if $L = L(M)$ for some TM M.

Given an input string w, M halts if $w \in L(M)$, but M **may not halt** if $w \notin L(M)$.

Recursive languages

A language L is **recursive** (REC) or, equivalently, the decision problem L represents is **decidable**, if $L = L(M)$ for a TM M that halts for **every** input.

A recursive/decidable language corresponds to the definition of **algorithm**, for which we impose that computation halts both for positive and negative instances of the problem.

String indexing

Let us sort all strings in $\{0,1\}^*$:

- by length;
- lexicografically, for strings of the same length.

i	string
1	ϵ
2	0
3	1
4	00
5	01
⋮	⋮

We associate with each string a positive integer i called **index**.

String indexing

We write w_i to denote the i-th string.

We can easily verify that, for each $w \in \{0,1\}^*$, we have

$$w = w_i \quad \Leftrightarrow \quad i = 1w \ .$$

Encoding of TM

We now want to encode a TM **with binary input alphabet** $M = (Q, \{0, 1\}, \Gamma, \delta, q_1, B, F\}$ by means of a binary string, which we denote $\text{enc}(M)$.

We need to assign integers to each state, tape symbol, and symbols L and R indicating directions.

We rename the states as $q_1, q_2, \ldots, q_r$. Initial state: q_1, final state: q_2 (unique).

We rename the tape symbols as $X_1, X_2, \ldots, X_s$. Also: $0 = X_1$, $1 = X_2$, $B = X_3$.

$L = D_1$ and $R = D_2$.

Encoding of TM

For the transition function, if

$$\delta(q_i, X_j) = (q_k, X_l, D_m)$$

the binary code C for the transition is (we use unary notation for i, j, k, l, m)

$$0^i 1 0^j 1 0^k 1 0^l 1 0^m$$

Note : We never have two consecutive occurrences of 1, since $i, j, k, l, m \geqslant 1$ is always satisfied.

Encoding of TM

For a TM, we concatenate the codes C_i for all transitions, separated by 11

$$C_1 11 C_2 11 \cdots 11 C_{n-1} 11 C_n$$

There are several codes for M, obtained by indexing the symbols and/or listing the transitions in different orders.

Many binary strings **do not** correspond to a TM
Example : 11001 or 001110.

Note : In the following we write enc(M) to denote a generic code for M; keep in mind that enc() **is not** a function.

Try to draw a map between set of all TMs and set of binary strings, representing the encoding relation.

Example

Let $M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\})$, where δ is defined as

$$\begin{array}{ll} \delta(q_1, 1) = (q_3, 0, R) & \delta(q_3, 0) = (q_1, 1, R) \\ \delta(q_3, 1) = (q_2, 0, R) & \delta(q_3, B) = (q_3, 1, L) \end{array}$$

Transition encodings C_i

0100100010100	0001010100100
00010010010100	0001000100010010

TM encoding enc(M)

0100100010100**11**0001010100100**11**00010010010100**11**0001000100010010

TM indexing

We can now **enumerate** all TM (with repetition) using positive integers as indices and using our string indexing.

For $i \geqslant 1$, the i-th TM M_i is defined as follows:

- if w_i is a valid encoding representing TM M, then $M_i = M$;
- if w_i is not a valid encoding, then M_i is the TM that halts immediately for any input (only one state and no transition, $L(M_i) = \emptyset$).

Diagonalization language

The **diagonalization language** is the set

$$L_d = \{w \mid w = w_i,\ w_i \notin L(M_i)\}$$

In words, L_d contains all binary strings w_i such that the i-th TM does not accept w_i.

Diagonalization language

The following table reports whether M_i accepts (1) or rejects (0) w_j.

i \ j	1	2	3	4	...
1	0	1	1	0	...
2	1	1	0	0	...
3	0	0	1	1	...
4	0	1	0	1	...
⋮	⋮	⋮	⋮	⋮	⋱

Diagonal

The table is a fake: the rows at the top should all be 0's.

Diagonalization language

We can interpret the i-th row of the table as the **characteristic vector** of language $L(M_i)$: an entry is 1 iff the corresponding string belongs to the language.

Observation : The table represents the entire class RE. In fact, a language is in RE if and only if its characteristic vector is a row of the table.

Diagonalization language

The following statements are logically equivalent:

- the i-th element of the diagonal is 0;
- $w_i \notin L(M_i)$;
- $w_i \in L_d$.

This means that, if we **complement** the diagonal, we obtain the characteristic vector of language L_d.

This vector cannot be a row of the table, because the diagonal element of each row does not match with at least one position of the characteristic vector of language L_d.

Diagonalization language

Theorem L_d is not in RE.

Proof Let us assume that there is a TM M such that $L_d = L(M)$. Choose i such that $M_i = M$. Does the string w_i belong to L_d?

If $w_i \in L_d$, then M_i accepts w_i because $L_d = L(M_i)$. But by definition of L_d, the i-th element of the diagonal is 0 and therefore M_i does not accept w_i.

If $w_i \notin L_d$, then M_i does not accept w_i. But by definition of L_d, the i-th element of the diagonal is 1 and therefore M_i accepts w_i.

We have therefore obtained a **contradiction**. □

Recursive languages

A language L is **recursive** (REC) if $L = L(M)$ for some TM M such that:

- if $w \in L$, then M halts in a final state;
- if $w \notin L$, then M halts in a non-final state.

If we think of L as a decision problem P_L, then we say that P_L is **decidable** whenever L is recursive, and P_L is **undecidable** otherwise.

Decidability corresponds to the notion of **algorithm**: we have a sequence of steps that always ends and produces some answer.

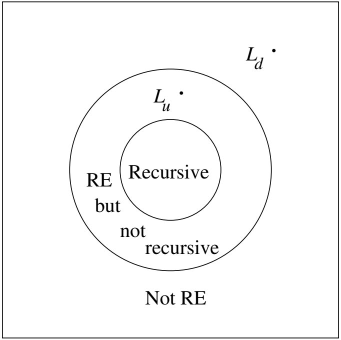
REC vs. RE$\setminus$REC

Comparison:

- recursive language means that there is an algorithm for **solving** the associated decision problem, that is, we always have an answer;
- language in RE that is non-recursive means that we can **enumerate** the positive instances of the problem, but we cannot conclude in a finite amount of time that an instance has a negative answer.

The distinction between decidable / undecidable problems is often more important than the distinction between RE / non-RE problems.

Language classes

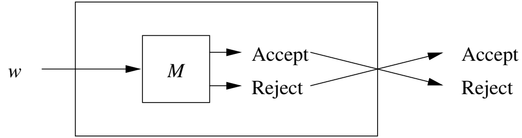


- recursive = decidable = M always halts;
- RE = M halts upon acceptance;
- non-RE = we cannot compute; **Example** : L_d.

Properties of recursive languages

Theorem If L is recursive, then $\overline{L}$ is recursive.

Proof If L is recursive, there is a TM M that always halts, such that $L(M) = L$. We construct a TM M' such that M' accepts when M does not, and *vice versa*. M' always halts and $L(M') = \overline{L}$. □



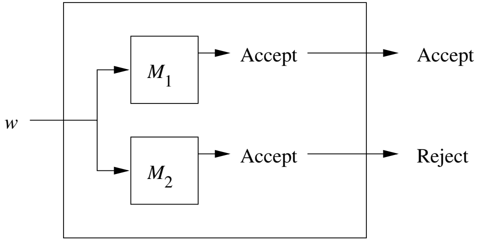
Corollary If L is in RE and $\overline{L}$ is not in RE, then L cannot be a recursive language.

Properties of RE languages

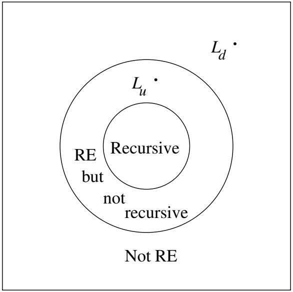
Theorem If L and $\overline{L}$ are in RE, then L is recursive.

Proof Let $L = L(M_1)$ and $\overline{L} = L(M_2)$. We build a multi-tape TM M that simulates M_1 and M_2 in parallel.

If the input is in L, M_1 accepts and halts, then also M accepts and halts. If the input is not in L, then M_2 accepts and halts, so M rejects and halts. □



L and $\overline{L}$



Where can L and $\overline{L}$ be placed?

Combinatorially, there are 9 possible arrangements, but the theory allows only 4 of them.

L and $\overline{L}$

Possible arrangements for L and $\overline{L}$:

- both L and $\overline{L}$ are recursive;
- both L and $\overline{L}$ are not in RE;
- L is RE but not recursive, and $\overline{L}$ is not RE;
- $\overline{L}$ is RE but not recursive, and L is not RE.

It is not possible that a language is recursive and the complement is RE but not recursive or not RE.

It is not possible that a language and its complement are both RE but not recursive.

Example

Let us consider the language $\overline{L_d}$, which contains the strings w_i such that M_i accepts w_i.

Since L_d is not RE, $\overline{L_d}$ is not recursive. It is possible that $\overline{L_d}$ is not RE, or alternatively RE but not recursive.

We will prove later that $\overline{L_d}$ is RE but not recursive.

Universal language

We want to encode pairs (M, w) consisting of:

- one TM M with binary input alphabet;
- one binary string w.

We use $\text{enc}(M)$ followed by 111, followed by w, and write $\text{enc}(M, w)$.

Note : the sequence 111 never appears in $\text{enc}(M)$.

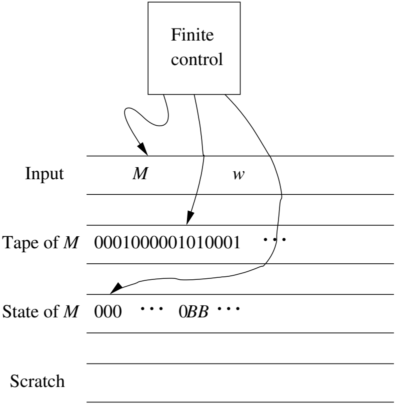
The language L_u, called **universal language**, is the set

$$L_u = \{\text{enc}(M, w) \mid w \in L(M)\} .$$

In words, L_u is the set of binary strings that encode a pair (M, w) such that $w \in L(M)$.

Universal TM

There exists a TM U, called **universal** TM, such that $L(U) = L_u$:



Universal TM

U (multi-tape version) has four tapes:

- tape 1 contains the input string enc(M, w);
- tape 2 simulates M's tape, using the 0^j format for each X_j tape symbol, and 1 as cell separator;
- tape 3 records M's state, using the 0^j format for each state q_j;
- tape 4: auxiliary copying tape, used to "enlarge" or "shrink" the available space for the 0^j representations in tape 2.

Universal TM

Strategy exploited by U:

- if enc(M) is invalid, U halts and rejects (in this case $L(M) = \emptyset$);
- write w on tape 2 using 1 as separator, 0^1 for $0 = X_1$, and 0^2 for $1 = X_2$;

  No encoding for B, use U's blank
- write the initial state on tape 3, using 0 for q_1, and place the tape head of tape 2 on the first cell;
- search on tape 1 for a transition of the form $0^i10^j10^k10^l10^m$, where
  - 0^i is the state on tape 3;
  - 0^j is M's tape symbol under the tape head of tape 2;

Universal TM

Strategy exploited by U (cont'd):

- in order to simulate trasition $0^i 1 0^j 1 0^k 1 0^l 1 0^m$, the TM U
  - replaces the content of tape 3 with 0^k (new state);
  - replaces 0^j on tape 2 with 0^l (new tape symbol); if needed, we can "enlarge" or "shrink" U's tapes using the auxiliary tape (tape 4);
  - move the tape head of tape 2 to the left if $m = 1$ or to the right if $m = 2$, until the next 1 is reached (separator);
- if there is no transition $0^i 1 0^j 1 0^k 1 0^l 1 0^m$, M halts and U halts as well;
- if M reaches a final state, then U halts and accepts.

Universal language

Theorem L_u is in RE but is not recursive.

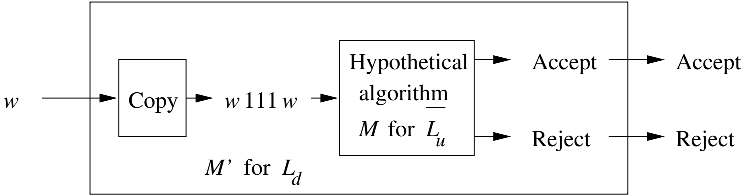
Proof L_u is in RE, since we have built the TM U.

Let us assume that L_u is recursive. Then $\overline{L_u}$ is also recursive.

Let M be a TM such that $L(M) = \overline{L_u}$ and M always halts. We will build a new TM M' for L_d, which is a contradiction.

This is an example of a reduction ($L_d <=_m \bar{L_u}$) a notion which we will introduce in the next section.

Universal language



On input $w = w_i$, M' builds $\text{enc}(M_i, w_i) = w_i111w_i$.

M always halts, and accepts if and only if $w_i \notin L(M_i)$. As a consequence, M' always halts, and $L(M') = L_d$.

We have a **contradiction**, since L_d is not recursive. □

The halting problem

Given a TM M, we define $H(M)$ the set of strings w such that M **halts** with input w.

Let us consider the language L_h, called the **halting problem**

$$L_h = \{\text{enc}(M, w) \mid w \in H(M)\} .$$

There exists a TM M such that $L(M) = L_h$: M takes as input a pair $\text{enc}(M', w)$ and simulates a computation of M' on w.

M accepts whenever M' halts on w.

Therefore L_h is a RE language.

The halting problem

We can prove that L_h is not recursive (proof omitted).

Hence there is **no algorithm** that can state whether a given program ends or not on a given input.

However, there exists a procedure that:

- halts, if a given program ends on a given input;
- cycles, if a given program does not end on a given input.

Reduction

Given a problem P_1 known to be "difficult", we want to know whether a second problem P_2 under investigation is as hard as, or even **harder** than, P_1.

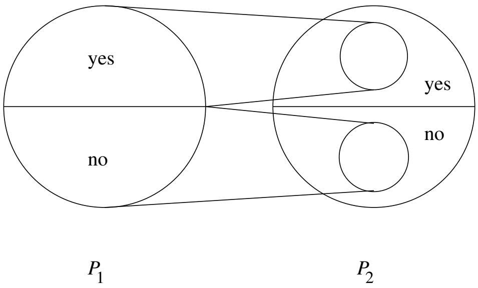
To this end we show that, if we could solve P_2, then we could also solve P_1, written

$$P_1 \leqslant_m P_2; .$$

This notation is not used in the book.

This technique is called **reduction** of P_1 to P_2.

Reduction



A **reduction** from P_1 to P_2 is an **algorithm** that converts an **instance** x of P_1 into an instance y of P_2, such that:

- if x has positive answer then y has positive answer;
- if x has negative answer then y has negative answer.

Reduction

Let $P_1 \leqslant_m P_2$, and assume there exists an algorithm that solves P_2. Given an instance x for P_1:

- we use the reduction to convert x to an instance y for P_2;
- we use the algorithm for P_2 to decide whether y is in P_2 or not.

Whatever the answer is, it is also valid for x in P_1.

We have built an algorithm that solves P_1. Thus solving P_2 is **at least as difficult** as solving P_1.

In other words, solution of P_2 requires enough computational resources to solve P_1.

Reduction

Theorem If $P_1 \leqslant_m P_2$, then:

- if P_1 is undecidable, so is P_2;
- if P_1 is not RE, so is P_2.

Proof (First part) Let us assume that P_2 is decidable:

- we apply the reduction to transform instance x of P_1 into instance y of P_2;
- we apply on y the algorithm to decide P_2.

We found an algorithm to decide P_1, which is a **contradiction**.

Reduction

(Second part) Let us assume that P_2 is RE:

- we apply the reduction to transform instance x of P_1 into instance y of P_2;
- we apply on y the algorithm to accept P_2 (it does not halt if y is a negative instance).

We have found a TM to accept P_1 (which does not halt if x is a negative instance). But this is a **contradiction**. □

This theorem implicitly used in the rest of the lecture whenever we construct a reduction.

TM accepting non-empty languages

We consider two languages formed by TM encodings

$$
\begin{aligned}
L_e &= \{\text{enc}(M) \mid L(M) = \emptyset\} \; ; \\
L_{ne} &= \{\text{enc}(M) \mid L(M) \neq \emptyset\} \; .
\end{aligned}
$$

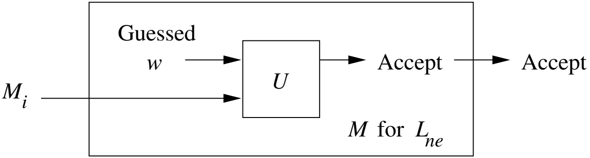
Note : $\overline{L_e} = L_{ne}$.

We want to find out whether these languages are recursive, or RE but not recursive, or else non-RE.

TM accepting non-empty languages

Theorem L_{ne} is RE.

Proof We construct a nondeterministic TM M with $L(M) = L_{ne}$.



Given M_i as input, M implements the following strategy:

- using nondeterminism, **guess** a string w;
- simulate U on M_i and w.

TM accepting non-empty languages

M accepts M_i if and only if there exists w such that $w \in L(M_i)$.

The theorem then follows from the equivalence between nondeterministic TM and TM. $\square$

TM accepting non-empty languages

Theorem L_{ne} is non-recursive.

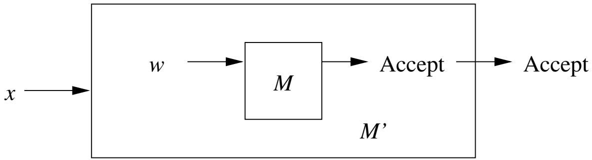
Proof We show that $L_u \leqslant_m L_{ne}$. Since L_u is non-recursive, it follows that even L_{ne} is non-recursive.

The reduction uses as **target** instances only (the encoding of) two languages in L_{ne}:

- the language Σ^* (positive instance);
- the empty language $\emptyset$ (negative instance).

TM accepting non-empty languages

Let us transform any instance enc(M, w) of L_u into an instance M' of L_{ne} defined as follows



M' ignores its input and uses its finite control to simulate a computation of M on w:

- if M accepts w, then M' accepts any input, that is, $L(M') = \Sigma^*$; thus $L(M') \neq \emptyset$;
- if M does not accept w, then M' does not accept any input, that is, $L(M') = \emptyset$. □

TM accepting empty languages

Theorem L_e is not in RE.

Proof We have already observed that $\overline{L_e} = L_{ne}$.

Since L_{ne} is RE but is not recursive, L_e cannot be in RE (if it were, then L_e and L_{ne} would both be recursive). □

Properties of the languages generated by TMs

Languages L_e and L_{ne} are associated with decision problems related to **properties** of RE languages (languages generated by TMs).

Instances of these decision problems are TMs, not languages, since the former are finite objects and the latter are infinite objects.

Our computations take as input finite objects.

In what follows, we will be concerned with more general properties of RE languages, and the associated decision problems.

The fact that L_e and L_{ne} are undecidable is a special case of a more general theorem, known as Rice's Theorem.

Properties of the languages generated by TMs

A property of the RE languages is **trivial** if it is satisfied by all or by none of the RE languages.

Rice's theorem states that all properties $\mathcal{P}$ of the RE languages that are nontrivial are undecidable.

This means that, for any nontrivial property $\mathcal{P}$, there is no TM that:

- always halts;
- given as input $\text{enc}(M_i)$, decides whether the language $L(M_i)$ satisfies $\mathcal{P}$.

Example

Checking whether a TM accepts a context-free language is undecidable.

In fact, the property of the RE languages "to be CFL" is nontrivial:

- some RE languages are CFL;
- not all RE languages are CFL.

Therefore the above statement follows from Rice's theorem.

Properties of the languages generated by TMs

We identify a property of the RE languages with the subset of RE languages that satisfy $\mathcal{P}$.

The language $L_{\mathcal{P}}$ is the set of encodings $\text{enc}(M_i)$ of all TMs M_i such that $L(M_i) \in \mathcal{P}$

$$L_{\mathcal{P}} = \{\text{enc}(M_i) \mid L(M_i) \in \mathcal{P}\} .$$

Note that we are representing RE languages by means of encodings of TMs.

$\mathcal{P}$ is decidable if and only if $L_{\mathcal{P}}$ is recursive.

Rice's theorem

Theorem Any nontrivial property of RE languages is undecidable.

Proof Let $\mathcal{P}$ be a nontrivial property of the RE languages. Let us assume by now that $\emptyset \notin \mathcal{P}$.

Let $L \in \mathcal{P}$ and let M_L be a TM such that $L(M_L) = L$.

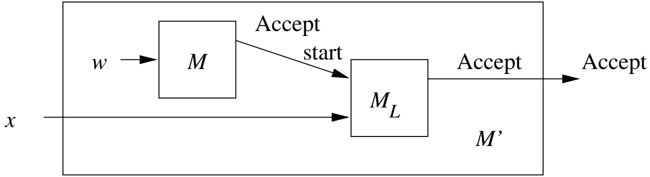
We prove that $L_u \leqslant_m L_{\mathcal{P}}$ using as **target** instances only (the encoding of) two languages:

- $L(M_L)$ (positive instance);
- $\emptyset$ (negative instance).

Then the theorem follows from the fact that L_u is undecidable.

Rice's theorem

Given an instance enc(M, w) for L_u, we produce an instance enc(M') of $L_{\mathcal{P}}$.



- if M does not accept w, M' does not accept any input string, and thus $L(M') = \emptyset \notin \mathcal{P}$;
- if M accepts w, M' simulates M_L on x, and thus $L(M') = L \in \mathcal{P}$.

Rice's theorem

Let us now assume that $\emptyset \in \mathcal{P}$. We consider $\overline{\mathcal{P}}$, the set of RE languages that do not satisfy the property $\mathcal{P}$.

Since $\emptyset \notin \overline{\mathcal{P}}$, the above argument proves that $L_u \leqslant_m L_{\overline{\mathcal{P}}}$. Therefore $L_{\overline{\mathcal{P}}}$ is not recursive.

Each TM accepts some RE language. Therefore we have

$$\overline{L_{\mathcal{P}}} = L_{\overline{\mathcal{P}}} \,.$$

If $L_{\mathcal{P}}$ were recursive, then $L_{\overline{\mathcal{P}}}$ would be recursive as well. This is a **contradiction** with respect to what we have previously asserted. □

Example

From Rice's theorem we have that the following problems are undecidable:

- is the language accepted by a TM the empty language? (already seen)
- is the language accepted by a TM a finite language?
- is the language accepted by a TM a regular language?
- is the language accepted by a TM a context-free language?
- does the language accepted by a TM contain the string 01?
- does the language accepted by a TM contain all even numbers?

Properties not inherent to the accepted language

In contrast with properties of RE languages, not all problems regarding TM are undecidable.

Problems that concern the states or the transitions of a TM, and not the accepted language, can be decided.

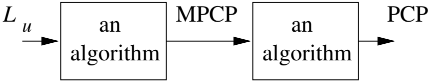
Example : the following problems can be decided:

- does a TM have five states?
- is there any input such that the TM performs at least five steps before halting?
- does a TM contain a certain transition?
- starting with the empty tape, does the TM reach state p in at most 5 steps?

Post's correspondence problem

We now investigate "real" problems, i.e., problems that do not concern TMs

We show that Post's correspondence problem, which refers to **strings**, is undecidable, using the following reductions



Later we will use this result to show that other real-world problems are undecidable

Post's correspondence problem

An instance of **Post's correspondence problem**, or PCP for short, is formed by two **equal length** lists of strings

$$\begin{aligned} A &= w_1, w_2, \ldots, w_k \\ B &= x_1, x_2, \ldots, x_k \end{aligned}$$

where $w_i, x_j \in \Sigma^+$ and Σ is an alphabet with at least two symbols

Instance (A, B) has a solution if there are $m \geqslant 1$ indices $i_1, i_2, \ldots, i_m$ such that

$$w_{i_1} w_{i_2} \cdots w_{i_m} = x_{i_1} x_{i_2} \cdots x_{i_m}$$

Example

PCP instance with $\Sigma = \{0, 1\}$

	A	B
i	w_i	x_i
1	1	111
2	10111	10
3	10	0

A possible solution is provided by the indices:
$m = 4,\ i_1 = 2,\ i_2 = 1,\ i_3 = 1,\ i_4 = 3$

$$w_2 w_1 w_1 w_3 = x_2 x_1 x_1 x_3 = 101111110$$

Possible solutions are also all **repetitions** of 2,1,1,3

Example

PCP instance with $\Sigma = \{0, 1\}$

	A	B
i	w_i	x_i
1	10	101
2	011	11
3	101	011

This instance has no solution. To prove this, let us assume $i_1, i_2, \ldots, i_m$ is a solution

If $i_1 = 2$ or $i_1 = 3$ we have a **mismatch** at the first position. Then we must have $i_1 = 1$

If $i_2 = 1$ or $i_2 = 2$ we still have a **mismatch**. Then we must have $i_2 = 3$

Example

We thus have the **partial solution**

$$\begin{array}{lcllllll} w_1 w_3 & = & 1 & 0 & 1 & 0 & 1 & \cdots \\ x_1 x_3 & = & 1 & 0 & 1 & 0 & 1 & 1 \quad \cdots \end{array}$$

If $i_3 = 1$ or $i_3 = 2$ we still have a **mismatch**. Then we must have $i_3 = 3$, providing the partial solution

$$\begin{array}{lcllllllllll} w_1 w_3 w_3 & = & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & \cdots \\ x_1 x_3 x_3 & = & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \quad \cdots \end{array}$$

We are now back to the previous scenario, forcing us to choose $i_4 = 3$, $i_5 = 3$, ... and we will never reach a complete solution

Modified Post's correspondence problem

An instance of the **modified** PCP, MPCP for short, is an instance (A, B) of PCP

(A, B) has a solution if there are $m \geqslant 0$ indices $i_1, i_2, \ldots, i_m$ such that

$$w_1 w_{i_1} w_{i_2} \cdots w_{i_m} \quad = \quad x_1 x_{i_1} x_{i_2} \cdots x_{i_m}$$

Note : (w_1, x_1) must be the starting choice, and m can be 0

Reduction

We present a transformation from instances (M, w) of L_u to instances (A, B) of the MPCP problem. We will later prove that this transformation is a reduction

Idea

- we assume semi-infinite tape TM with ID's without any blank, as in a previous theorem
- we represent M's computations as strings of the form

  $$\#\alpha_1\#\alpha_2\#\alpha_3\#\cdots$$

  where each α_i is an ID
- we use fictitious ID's that erase the tape when a final state is reached (needed to realign)

Reduction

Idea (cont'd)

- partial solutions of (A, B) simulate computations of M on w
- in a partial solution, the list obtained by A is always one ID **behind** with respect to the list obtained by B

$$\begin{array}{lllll} \ell_A: & \#\alpha_1 & \cdots & \#\alpha_{i-1} & \\ \ell_B: & \#\alpha_1 & \cdots & \#\alpha_{i-1} & \#\alpha_i \end{array}$$

- the pairs (w_i, x_i) are used, through several steps, to
  - copy $\#\alpha_i$ from ℓ_B into ℓ_A
  - add to ℓ_B the new string $\#\alpha_{i+1}$, which simulates the next move of M

Reduction

Transformation: input (M, w), $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

- Pairs of type 1: initial ID

A	B
$\#$	$\#q_0w\#$

- Pairs of type 2: copy tape symbols and $\#$

A	B	
X	X	for each $X \in \Gamma$
$\#$	$\#$	

Reduction

Transformation (cont'd)

- Pairs of type 3: simulate next move for $q \in Q \backslash F$

A	B	
qX	Yp	if $\delta(q, X) = (p, Y, R)$
ZqX	pZY	if $\delta(q, X) = (p, Y, L)$
$q\#$	$Yp\#$	if $\delta(q, B) = (p, Y, R)$
$Zq\#$	$pZY\#$	if $\delta(q, B) = (p, Y, L)$

Reduction

Transformation (cont'd)

- Pairs of type 4: for $q \in F$, erase working tape

A	B
XqY	q
Xq	q
qY	q

- Pairs of type 5: align the two lists, after the tape has been erased

A	B
$q\#\#$	$\#$

Example

Instance of L_u: $(M, 01)$

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

q_i	$\delta(q_i, 0)$	$\delta(q_i, 1)$	$\delta(q_i, B)$
$\rightarrow q_1$	$(q_2, 1, R)$	$(q_2, 0, L)$	$(q_2, 1, L)$
q_1	$(q_3, 0, L)$	$(q_1, 0, R)$	$(q_2, 0, R)$
$\star\ q_3$	—	—	—

Example

List of pairs

type	w_i	x_i	derived from
(1)	#	$\#q_1 01\#$	
(2)	0	0	
	1	1	
	#	#	

Example

List of pairs (cont'd)

type	w_i	x_i	derived from
(3)	$q_1 0$	$1 q_2$	from $\delta(q_1, 0)(q_2, 1, R)$
	$0 q_1 1$	$q_2 00$	from $\delta(q_1, 1)(q_2, 0, L)$
	$1 q_1 1$	$q_2 10$	from $\delta(q_1, 1)(q_2, 0, L)$
	$0 q_1 \#$	$q_2 01\#$	from $\delta(q_1, B)(q_2, 1, L)$
	$1 q_1 \#$	$q_2 11\#$	from $\delta(q_1, B)(q_2, 1, L)$
	$0 q_2 0$	$q_3 00$	from $\delta(q_2, 0)(q_3, 0, L)$
	$1 q_2 0$	$q_3 10$	from $\delta(q_2, 0)(q_3, 0, L)$
	$q_2 1$	$0 q_1$	from $\delta(q_2, 1)(q_1, 0, R)$
	$q_2 \#$	$0 q_2 \#$	from $\delta(q_2, B)(q_2, 0, R)$

Example

List of pairs (cont'd)

type	w_i	x_i	derived from
(4)	$0q_30$	$q_3\#$	
	$0q_31$	$q_3\#$	
	$1q_30$	$q_3\#$	
	$1q_31$	$q_3\#$	
	$0q_3$	$q_3\#$	
	$1q_3$	$q_3\#$	
	q_30	$q_3\#$	
	q_31	$q_3\#$	
(5)	$q_3\#\#$	$\#$	

Example

M accepts input 01 through the following computation

$$q_1 01 \underset{M}{\vdash} 1 q_2 1 \underset{M}{\vdash} 10 q_1 \underset{M}{\vdash} 1 q_2 01 \underset{M}{\vdash} q_3 101$$

We consider the partial solutions of MPCP associated with the above computation

First pair is mandatory, and simulates the initial ID

$$\begin{array}{ll} \ell_A : & \# \\ \ell_B : & \# q_1 01 \# \end{array}$$

We have only one way to expand the partial solution, that is, use the pair $(q_1 0, 1 q_2)$ which simulates the first move

$$\begin{array}{ll} \ell_A : & \# q_1 0 \\ \ell_B : & \# q_1 01 \# 1 q_2 \end{array}$$

Example

We apply three pairs for copying, in order to reach the next state

$$\begin{aligned} \ell_A &: \quad \#q_1 01\#1 \\ \ell_B &: \quad \#q_1 01\#1q_2 1\#1 \end{aligned}$$

We apply pair $(q_2 1, 0q_1)$ to simulate the second move

$$\begin{aligned} \ell_A &: \quad \#q_1 01\#1q_2 1 \\ \ell_B &: \quad \#q_1 01\#1q_2 1\#10q_1 \end{aligned}$$

And so forth ...

PCP

Theorem $L_u \leqslant_m$ MPCP

Proof (sketch) We need to show that, for the previous transformation, (M, w) has a solution if and only if (A, B) has a solution

(*only if*) If $w \in L(M)$ there exists an accepting computation. Then the partial solution ℓ_A reaches ℓ_B and (A, B) has a solution

(*if*) Every solution of (A, B) starts with the initial ID of M on w, proceeds with the simulation of some moves of M, and stops when M reaches an accepting state. Therefore $w \in L(M)$ □

PCP

Theorem MPCP $\leqslant_m$ PCP

Proof not required

Theorem PCP is undecidable

Proof From $L_u \leqslant_m$ MPCP and from MPCP $\leqslant_m$ PCP, we conclude that $L_u \leqslant_m$ PCP □

Composition of two reductions is still a valid reduction

CFG ambiguity

We assume a binary **encoding** for CFGs, similar to the one used for TM

We write enc(G) for the encoding of CFG G

The **ambiguity problem** for a CFG is defined as follows

- the instances are the strings enc(G) where G is a CFG
- the answer is positive if G is ambiguous

We define the corresponding language

$$L_{AMB} = \{\text{enc}(G) \mid G \text{ is ambiguous}\}$$

Reduction

We present a transformation from PCP to instances of the L_{AMB} problem. We will later prove that this transformation is a reduction

Let (A, B) be an instance of PCP over the alphabet Σ, where $A = w_1, w_2, \ldots, w_k$ and $B = x_1, x_2, \ldots, x_k$

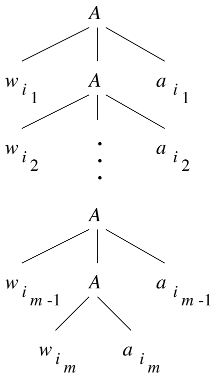
Let G_A be a CFG defined as

- nonterminal set $\{A\}$
- alphabet $\Sigma \cup \{a_i \mid 1 \leqslant i \leqslant k\}$, where a_i is an alias for the pair w_i, x_i
- production set

$$
\begin{aligned}
A &\rightarrow w_1 A a_1 \mid w_2 A a_2 \mid \cdots \mid w_k A a_k \\
&\rightarrow w_1 a_1 \mid w_2 a_2 \mid \cdots \mid w_k a_k
\end{aligned}
$$

Example

Strings generated by G_A have the form $w_{i_1} w_{i_2} \cdots w_{i_m} a_{i_m} \cdots a_{i_2} a_{i_1}$, with $m \geqslant 1$



Reduction

Symmetrically, let G_B be a CFG defined as

- nonterminal set $\{B\}$
- alphabet $\Sigma \cup \{a_i \mid 1 \leqslant i \leqslant k\}$
- production set

$$B \rightarrow x_1 B a_1 \mid x_2 B a_2 \mid \cdots \mid x_k B a_k$$
$$\rightarrow x_1 a_1 \mid x_2 a_2 \mid \cdots \mid x_k a_k$$

Reduction

We observe that G_A and G_B are unambiguous

We define $L_A = L(G_A)$ and $L_B = L(G_B)$

G_{AB} is the CFG that generates the language $L_A \cup L_B$

- nonterminal set $\{S, A, B\}$
- alphabet $\Sigma \cup \{a_i \mid 1 \leqslant i \leqslant k\}$
- production set $S \rightarrow A \mid B$ and in addition all productions of G_A and G_B

L_{AMB}

Theorem PCP $\leqslant_m L_{AMB}$

Proof (sketch) We need to show that, for the given reduction, $\text{enc}(G_{AB}) \in L_{AMB}$ if and only if (A, B) has a solution

(If part) Let $i_1, i_2, \ldots, i_m$ be a solution for (A, B). Then G_{AB} has two derivations for the same string

$$
\begin{aligned}
S &\Rightarrow A \Rightarrow w_{i_1} A a_{i_1} \Rightarrow w_{i_1} w_{i_2} A a_{i_2} a_{i_1} \Rightarrow \cdots \\
&\Rightarrow w_{i_1} w_{i_2} \cdots w_{i_m} a_{i_m} \cdots a_{i_2} a_{i_1} \\
S &\Rightarrow B \Rightarrow x_{i_1} B a_{i_1} \Rightarrow x_{i_1} x_{i_2} B a_{i_2} a_{i_1} \Rightarrow \cdots \\
&\Rightarrow x_{i_1} x_{i_2} \cdots x_{i_m} a_{i_m} \cdots a_{i_2} a_{i_1}
\end{aligned}
$$

L_{AMB}

(Only if part) Assume G_{AB} is ambiguous. Consider an ambiguous string in $L(G_{AB})$, having the form

$$za_{i_m} \cdots a_{i_2} a_{i_1}$$

with $z \in \Sigma^+$

Since G_A and G_B are not ambiguous, the ambiguous string must have two leftmost derivations starting with $S \Rightarrow A$ and $S \Rightarrow B$

Then $i_1, i_2, \ldots, i_m$ is a solution for (A, B) □

CFG problems

Let G_1 and G_2 be CFGs, and let R be a regular expression. The following problems are undecidable

- $L(G_1) \cap L(G_2) = \emptyset$?
- $L(G_1) = L(G_2)$?
- $L(G_1) = L(R)$?
- $L(G_1) = T^*$, for a fixed alphabet T?
- $L(G_1) \subseteq L(G_2)$?
- $L(R) \subseteq L(G_1)$?