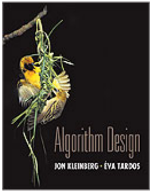
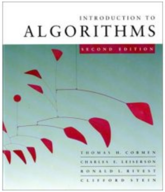
Algorithms and Complexity

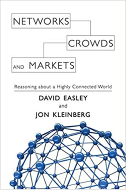
Fall 2023

Algorithmics: Basic references

- Kleinberg, Tardos. Algorithm Design, Pearson Education, 2006.
- Cormen, Leisserson, Rivest and Stein. Introduction to algorithms. Second edition, MIT Press and McGraw Hill 2001.
- Easley, Kleinberg. Networks, Crowds, and Markets: Reasoning About a Highly Connected World, Cambridge University Press, 2010

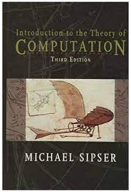


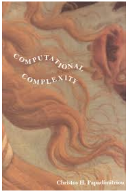


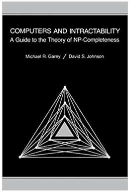


Computational Complexity: Basic references

- Sipser Introduction to the Theory of Computation 2013.
- Papadimitriou Computational Complexity 1994.
- Garey and Johnson Computers and Intractability: A Guide to the Theory of NP-Completeness 1979







Alphabets and languages

- Alphabet: a non-empty finite set.
- Symbol: an element of an alphabet.
- Word: a finite sequence of symbols.
  λ denotes the empty word, a sequence with 0 symbols.
- Language: set of words over an alphabet.

Alphabets and languages: concatenation

- For an alphabet Σ, Σ^* denotes the set of words over Σ.

Alphabets and languages: concatenation

- For an alphabet Σ, Σ^* denotes the set of words over Σ.
- The basic operation on words is the concatenation.
  - For $x, y \in \Sigma^*$, $x \cdot y$ is the word obtained placing the symbols in x followed by the symbols in y.
  - For example, if $\Sigma = \{0, 1\}$, $x = 001000$ and $y = 11101$, $xy = 00100011101$.
  - $(\Sigma^*, \cdot)$ is a non-commutative monoid.

Alphabets and languages: concatenation

- For an alphabet Σ, Σ^* denotes the set of words over Σ.
- The basic operation on words is the concatenation.
  - For $x, y \in \Sigma^*$, $x \cdot y$ is the word obtained placing the symbols in x followed by the symbols in y.
  - For example, if $\Sigma = \{0, 1\}$, $x = 001000$ and $y = 11101$, $xy = 00100011101$.
  - $(\Sigma^*, \cdot)$ is a non-commutative monoid.
- For $x \in \Sigma^*$, the length of x ($|x|$) is the number of symbols in x.
  - $|x \cdot y| = |x| + |y|$
  - $|\lambda| = 0$

Alphabets and languages: concatenation

- For an alphabet Σ, Σ^* denotes the set of words over Σ.
- The basic operation on words is the concatenation.
  - For $x, y \in \Sigma^*$, $x \cdot y$ is the word obtained placing the symbols in x followed by the symbols in y.
  - For example, if $\Sigma = \{0, 1\}$, $x = 001000$ and $y = 11101$, $xy = 00100011101$.
  - $(\Sigma^*, \cdot)$ is a non-commutative monoid.
- For $x \in \Sigma^*$, the length of x ($|x|$) is the number of symbols in x.
  - $|x \cdot y| = |x| + |y|$
  - $|\lambda| = 0$
- A language L is a subset of Σ^*.
  We can extend concatenation to languages in the usual form.

$$L_1 \cdot L_2 = \{x \cdot y \mid x \in L_1, y \in L_2\}$$

Alphabets and languages: enumerability

- Let Σ be an alphabet, Σ^* is enumerable.
- We cannot use alphabetical order $a, aa, aaa, aaaa, \ldots$
- We use lexicographic order
  - Order words by length.
  - Among words with the same length order them according to alphabetical order.

Alphabets and languages: enumerability

- Let Σ be an alphabet, Σ^* is enumerable.
- We cannot use alphabetical order $a, aa, aaa, aaaa, \ldots$
- We use lexicographic order
  - Order words by length.
  - Among words with the same length order them according to alphabetical order.
- For $\Sigma = \{0, 1\}$ we can enumerate $\{0.1\}^*$ as

$$\lambda, 0, 1, 00, 01, 10, 11, 000, \ldots, 111, 0000, \ldots$$

Problem types

- Decision

  Input x

  Property $P(x)$

  Example: Given a graph and two vertices, is there a path joining them?

- Function

  Input x

  Compute y *such that* $Q(x, y)$

  Example: Given a graph and two vertices, compute the minimum distance between them.

Problem types

- Decision

  Input x

  Property $P(x)$

Problem types

- Decision

  Input x

  Property $P(x)$

  By coding inputs on alphabet Σ such a problem identifies a language:

Problem types

- Decision

  Input x

  Property $P(x)$

By coding inputs on alphabet Σ such a problem identifies a language:

$\{\langle x\rangle \mid P(x)\} \in \mathcal{P}(\Sigma^*)$

Problem types

- Decision

  Input x

  Property $P(x)$

  By coding inputs on alphabet Σ such a problem identifies a language:
  $\{\langle x\rangle \mid P(x)\} \in \mathcal{P}(\Sigma^*)$

- Function

  Input x

  Compute y *such that* $Q(x, y)$

Problem types

- Decision
  Input x
  Property $P(x)$

  By coding inputs on alphabet Σ such a problem identifies a language:
  $\{\langle x\rangle \mid P(x)\} \in \mathcal{P}(\Sigma^*)$

- Function
  Input x
  Compute y *such that* $Q(x, y)$

  By coding inputs/outputs on alphabet Σ a deterministic algorithm solving a problem determines a function

Problem types

- Decision

  Input x

  Property $P(x)$

  By coding inputs on alphabet Σ such a problem identifies a language:
  $\{\langle x\rangle \mid P(x)\} \in \mathcal{P}(\Sigma^*)$

- Function

  Input x

  Compute y *such that* $Q(x, y)$

  By coding inputs/outputs on alphabet Σ a deterministic algorithm solving a problem determines a function $f : \Sigma^* \to \Sigma^*$ s.t., for any x, $Q(x, f(x))$ is true.

Decision problem classes

- Undecidable
  No algorithm can solve the problem.
- Decidable
  There is an algorithm solving them.

Turing machines

- A Turing machine (TM) M is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{F}})$, where
  - Q is a finite set of states,
  - Σ is the input alphabet.
  - Γ is the tape alphabet, $\Gamma = \Sigma \cup \{\mathtt{b}, \blacktriangleright\}$, with $\mathtt{b}, \blacktriangleright \notin \Sigma$.
  - $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\mathtt{l}, \mathtt{r}, \mathtt{n}\}$ is the transition function.
  - q_0 is the initial state.
  - q_{F} is the final or accepting state.

Turing machines: Recognizing languages

- Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_\mathrm{F})$ be a TM and $x \in \Sigma^*$
- The computation of M with input x goes as follows:
  - Initially: the state is q_0; the tape has $\blacktriangleright x$ and all remaining cells in the tape hold a $\mathtt{b}$; the head has access to the first symbol of x.
  - While there is a transition in δ for the combination state, symbol accessed by the head, the transition is applied.

Turing machines: Recognizing languages

- Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$ be a TM and $x \in \Sigma^*$
- The computation of M with input x goes as follows:
  - Initially: the state is q_0; the tape has $\blacktriangleright x$ and all remaining cells in the tape hold a $\mathtt{b}$; the head has access to the first symbol of x.
  - While there is a transition in δ for the combination state, symbol accessed by the head, the transition is applied.
- Assuming that Q and Γ are disjoint, the word $\blacktriangleright \alpha q \beta$ is a configuration in which $\blacktriangleright \alpha\beta$ are the tape contents ($\mathtt{b}$ outside), q is a state and the head is accessing the tape cell holding the first symbol in β.

Turing machines: Recognizing languages

- Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$ be a TM and $x \in \Sigma^*$
- The computation of M with input x goes as follows:
  - Initially: the state is q_0; the tape has $\blacktriangleright x$ and all remaining cells in the tape hold a b; the head has access to the first symbol of x.
  - While there is a transition in δ for the combination state, symbol accessed by the head, the transition is applied.
- Assuming that Q and Γ are disjoint, the word $\blacktriangleright \alpha q \beta$ is a configuration in which $\blacktriangleright \alpha\beta$ are the tape contents (b outside), q is a state and the head is accessing the tape cell holding the first symbol in β.
- The computation of M on x is a sequence of configurations, starting with $\blacktriangleright q_0 x$, so that we can pass from one configuration to the next applying δ.

Turing machines: Recognizing languages

- The computation of a TM on input x is a sequence of configurations.
- If the sequence of configurations is finite, we say the M halts on input x, we note this as $M(x) \downarrow$,
  otherwise, the computation diverges or does not halt, $M(x) \uparrow$.

Turing machines: Recognizing languages

- The computation of a TM on input x is a sequence of configurations.
- If the sequence of configurations is finite, we say the M halts on input x, we note this as $M(x)\downarrow$,
  otherwise, the computation diverges or does not halt, $M(x)\uparrow$.
- When $M(x)\downarrow$,
  - the number of configurations in the computation is the computation time.
  - Let $\blacktriangleright \alpha q \beta$ be the last configuration in the computation.
    - $M(x) = \alpha\beta$ is the output,
    - M halts on input x in state q.

Turing machines: Recognizing languages

- The computation of a TM on input x is a sequence of configurations.
- If the sequence of configurations is finite, we say the M halts on input x, we note this as $M(x)\downarrow$,
  otherwise, the computation diverges or does not halt, $M(x)\uparrow$.
- When $M(x)\downarrow$,
  - the number of configurations in the computation is the computation time.
  - Let $\blacktriangleright \alpha q \beta$ be the last configuration in the computation.
    - $M(x) = \alpha\beta$ is the output,
    - M halts on input x in state q.
- $L(M) \subseteq \Sigma^*$ is the set of words that M accepts, i.e, M on input x halts in state q_{F}.
- A language $L \subseteq \Sigma^*$ is recognizable iff there is a TM M with $L = L(M)$.

The Church-Turing thesis: Problems and programs

- TM is a synonym of program. However, a program allows different construction rules than a TM but, so far, programmable as TM.

The Church-Turing thesis: Problems and programs

- TM is a synonym of program. However, a program allows different construction rules than a TM but, so far, programmable as TM.
- The input to a program has to be fitted into main memory, so it can be identified with a string in $\{0,1\}^*$ (or in another suitable alphabet).

The Church-Turing thesis: Problems and programs

- TM is a synonym of program. However, a program allows different construction rules than a TM but, so far, programmable as TM.
- The input to a program has to be fitted into main memory, so it can be identified with a string in $\{0,1\}^*$ (or in another suitable alphabet).
- The set of TM is enumerable (easy to see), so the set of all programs is enumerable.

The Church-Turing thesis: Problems and programs

- TM is a synonym of program. However, a program allows different construction rules than a TM but, so far, programmable as TM.
- The input to a program has to be fitted into main memory, so it can be identified with a string in $\{0, 1\}^*$ (or in another suitable alphabet).
- The set of TM is enumerable (easy to see), so the set of all programs is enumerable.
- So, we can identify TMs with the natural numbers. Let M_x be the x-th TM.

The Church-Turing thesis: Problems and programs

- In fact, a TM/program is a word in an adequate alphabet. So, we can consider a TM/program as the input to a TM/program.

The Church-Turing thesis: Problems and programs

- In fact, a TM/program is a word in an adequate alphabet. So, we can consider a TM/program as the input to a TM/program.
- The universal TM designed by Turing, has input x, y and outputs the result of executing TM x on input y. This has an equivalent in the language interpreter.

The Church-Turing thesis: Problems and programs

- In fact, a TM/program is a word in an adequate alphabet. So, we can consider a TM/program as the input to a TM/program.
- The universal TM designed by Turing, has input x, y and outputs the result of executing TM x on input y. This has an equivalent in the language interpreter.
- Technically x, y is not a well defined input. Nevertheless, $\mathbb{N} \times \mathbb{N}$ is enumerable, so we have a bijection to $\mathbb{N}$ (or Σ^*) that allow us to recover the original words. We denote this effective codification as $\langle x, y\rangle \in \Sigma^*$.

The Entscheidungsproblem.

- David Hilbert, ICM Bolonia-1929, The Entscheidungsproblem: Find a procedure to decide the validity of a given a first-order logic expression, in a finite number of operations.

  Ex: Is it true that
  $\neg\exists x, y, z, n \in \mathbb{N} : (n \geq 3) \wedge (x^n + y^n = z^n)$?

- Reflecting Hilbert's dream of a mechanical procedure that can prove or refute any mathematical claim.

  If *the Entscheidungsproblem* is solved $\Rightarrow$ all mathematics could be mechanized.

Hilbert program fails: Gödel's theorem

- The framework
  - Formal system: a language, a finite set of axioms and a set of inference rules used to derive an expression from a set of axioms.
  - A statement S in a formal system is decidable: if there is a finite proof, according to the inference rules, which will correctly prove that S is true or not. Otherwise the statement is undecidable
  - A formal system is complete: every statement can be proved or disproved. Otherwise the system is incomplete.

- K. Gödel's incompleteness theorem (1931):
  In any formal system sufficiently powerful to include ordinary arithmetic, there will be undecidable statements.

  There are statements in the arithmetic ($\mathbb{N}$ with $+, x$), which can not be proved or disproved.

The Entscheidungsproblem unsolvable: Turing's approach

- Consider the set of TMs, i.e., tuples $(Q, \Sigma, \Gamma, \delta, q_0, q_F)$, this set is enumerable.
- The set of languages is $\mathcal{P}(\Sigma^*)$, which is not enumerable.
- By Cantor's theorem there are languages that are not recognizable.

The Entscheidungsproblem unsolvable: Turing's approach

- Consider the set of TMs, i.e., tuples $(Q, \Sigma, \Gamma, \delta, q_0, q_F)$, this set is enumerable.
- The set of languages is $\mathcal{P}(\Sigma^*)$, which is not enumerable.
- By Cantor's theorem there are languages that are not recognizable.

The Entscheidungsproblem unsolvable: Turing's approach

- A TM M stops always on accepted inputs, but it may stop or not on rejected inputs.

The Entscheidungsproblem unsolvable: Turing's approach

- A TM M stops always on accepted inputs, but it may stop or not on rejected inputs.
- A TM M is a decider if it stops on any input $x \in \Sigma^*$.

The Entscheidungsproblem unsolvable: Turing's approach

- A TM M stops always on accepted inputs, but it may stop or not on rejected inputs.
- A TM M is a decider if it stops on any input $x \in \Sigma^*$.
- A language $L \subseteq \Sigma^*$ is decidable iff there is a decider M such that $L = L(M)$.

An undecidable language

Theorem

The language $A_{TM} = \{\langle x, w\rangle \mid w \in L(M_x)\}$ *is undecidable.*

Proof.

An undecidable language

Theorem

The language $A_{TM} = \{\langle x, w\rangle \mid w \in L(M_x)\}$ *is undecidable.*

Proof.

- Suppose that M_y is a decider for A_{TM}.

An undecidable language

Theorem

The language $A_{TM} = \{\langle x, w\rangle \mid w \in L(M_x)\}$ *is undecidable.*

Proof.

- Suppose that M_y is a decider for A_{TM}.
- On input $\langle x, w\rangle$, M_y halts and accepts if M_x on input w halts and accepts. Furthermore, it halts and rejects if M_x fails to accept w.

An undecidable language

Theorem

The language $A_{TM} = \{\langle x, w\rangle \mid w \in L(M_x)\}$ *is undecidable.*

Proof.

- Suppose that M_y is a decider for A_{TM}.
- On input $\langle x, w\rangle$, M_y halts and accepts if M_x on input w halts and accepts. Furthermore, it halts and rejects if M_x fails to accept w.
- Consider the following TM, M, that on input x:
    - Executes M_y on input $\langle x, x\rangle$
    - If M_y accepts, reject, otherwise, accept

An undecidable language

Theorem

The language $A_{TM} = \{\langle x, w\rangle \mid w \in L(M_x)\}$ is undecidable.

Proof.

- Suppose that M_y is a decider for A_{TM}.
- On input $\langle x, w\rangle$, M_y halts and accepts if M_x on input w halts and accepts. Furthermore, it halts and rejects if M_x fails to accept w.
- Consider the following TM, M, that on input x:
  - Executes M_y on input $\langle x, x\rangle$
  - If M_y accepts, reject, otherwise, accept
- What happens when we run M with its own number z as input?

An undecidable language

Theorem

The language $A_{TM} = \{\langle x, w\rangle \mid w \in L(M_x)\}$ is undecidable.

Proof.

- Suppose that M_y is a decider for A_{TM}.
- On input $\langle x, w\rangle$, M_y halts and accepts if M_x on input w halts and accepts. Furthermore, it halts and rejects if M_x fails to accept w.
- Consider the following TM, M, that on input x:
  - Executes M_y on input $\langle x, x\rangle$
  - If M_y accepts, reject, otherwise, accept
- What happens when we run M with its own number z as input?

  $M = M_z$ accepts z iff M_y rejects $\langle z, z\rangle$ iff M_z rejects z.

A unrecognizable language

Theorem

A language L is decidable iff L and $\overline{L}$ are recognizable.

A unrecognizable language

Theorem

A language L is decidable iff L and $\overline{L}$ are recognizable.

Proof.

- If L is decidable, the TM M deciding L recognizes L. Consider M' that is equal to M but: add a new final state $q'_{\mathtt{F}}$ and, for any missing transition (q, a) in δ with $q \neq q_{\mathtt{F}}$, add transition $\delta'(q, a) = (q'_{\mathtt{F}}, a, \mathtt{n})$.

A unrecognizable language

Theorem

A language L is decidable iff L and $\overline{L}$ are recognizable.

Proof.

- If L is decidable, the TM M deciding L recognizes L. Consider M' that is equal to M but: add a new final state q'_{F} and, for any missing transition (q, a) in δ with $q \neq q_{\mathrm{F}}$, add transition $\delta'(q, a) = (q'_{\mathrm{F}}, a, \mathtt{n})$.
- When L and $\overline{L}$ are recognizable, the trick is consider the two TM and to run one step of each machine until one of them stops and accepts. Determine acceptance according to the machine that accepted.

$\square$

A unrecognizable language

Theorem

A language L is decidable iff L and $\overline{L}$ are recognizable.

Proof.

- If L is decidable, the TM M deciding L recognizes L. Consider M' that is equal to M but: add a new final state q'_{F} and, for any missing transition (q, a) in δ with $q \neq q_{\mathrm{F}}$, add transition $\delta'(q, a) = (q'_{\mathrm{F}}, a, \mathtt{n})$.
- When L and $\overline{L}$ are recognizable, the trick is consider the two TM and to run one step of each machine until one of them stops and accepts. Determine acceptance according to the machine that accepted.

$\square$

$\overline{A_{TM}}$ is not recognizable.

Many-one reducibility

- E. L. Post, 1944

Many-one reducibility

- E. L. Post, 1944
- A TM can be extended to produce an output by interpreting the block of symbols left in the tape between $\blacktriangleright$ and the first $\flat$. $M(w) \in \Sigma^*$ represents the output of M on input w.

Many-one reducibility

- E. L. Post, 1944
- A TM can be extended to produce an output by interpreting the block of symbols left in the tape between $\blacktriangleright$ and the first b. $M(w) \in \Sigma^*$ represents the output of M on input w.
- A function $f : \Sigma^* \to \Sigma^*$ is computable if, for some TM M, on every input w, $M(w)\downarrow$ and $M(w) = f(w)$.

Many-one reducibility

- E. L. Post, 1944
- A TM can be extended to produce an output by interpreting the block of symbols left in the tape between ▶ and the first ␢. $M(w) \in \Sigma^*$ represents the output of M on input w.
- A function $f : \Sigma^* \to \Sigma^*$ is computable if, for some TM M, on every input w, $M(w)\downarrow$ and $M(w) = f(w)$.
- *Problem A is many-one reducible to problem B ($A \leq_m B$)* if there is a computable function f so that $x \in L_A$ iff $f(x) \in L_B$.

Many-one reducibility

- E. L. Post, 1944
- A TM can be extended to produce an output by interpreting the block of symbols left in the tape between $\blacktriangleright$ and the first $\flat$. $M(w) \in \Sigma^*$ represents the output of M on input w.
- A function $f : \Sigma^* \to \Sigma^*$ is computable if, for some TM M, on every input w, $M(w) \downarrow$ and $M(w) = f(w)$.
- *Problem A is many-one reducible to problem B ($A \leq_m B$)* if there is a computable function f so that $x \in L_A$ iff $f(x) \in L_B$.
- f is called the reduction.

Many-one reducibility

- E. L. Post, 1944
- A TM can be extended to produce an output by interpreting the block of symbols left in the tape between ▶ and the first ƀ. $M(w) \in \Sigma^*$ represents the output of M on input w.
- A function $f : \Sigma^* \to \Sigma^*$ is computable if, for some TM M, on every input w, $M(w)\downarrow$ and $M(w) = f(w)$.
- *Problem A is many-one reducible to problem B ($A \leq_m B$)* if there is a computable function f so that $x \in L_A$ iff $f(x) \in L_B$.
- f is called the reduction.

If $A \leq_m B$

- B is decidable $\Rightarrow$ A is decidable.
- A is undecidable $\Rightarrow$ B is undecidable.

Some undecidable problems

- HALT: Given a program P and an input x, does P halt on input x?
- Given a program P, does P halt on input 0?
- Given a program P, is the set of inputs x on which P halts finite?
- Given a program P, is there an input x s.t. $P(x) = x$?

Semi-Thue systems

Introduced by Axel Thue (1863-1922)

- A rewriting system or semi-Thue system is a tuple (Σ, R) where
  - Σ is an alphabet, usually assumed finite.
  - R is a binary relation on strings, i.e., $R \subseteq \Sigma^* \times \Sigma^*$.

Semi-Thue systems

Introduced by Axel Thue (1863-1922)

- A rewriting system or semi-Thue system is a tuple (Σ, R) where
  - Σ is an alphabet, usually assumed finite.
  - R is a binary relation on strings, i.e., $R \subseteq \Sigma^* \times \Sigma^*$.
- Each element $(u, v) \in R$ is called a (rewriting) rule and is usually written $u \to v$.

Semi-Thue systems

Introduced by Axel Thue (1863-1922)

- A rewriting system or semi-Thue system is a tuple (Σ, R) where
  - Σ is an alphabet, usually assumed finite.
  - R is a binary relation on strings, i.e., $R \subseteq \Sigma^* \times \Sigma^*$.
- Each element $(u, v) \in R$ is called a (rewriting) rule and is usually written $u \to v$.
- When R is symmetric the system is called a Thue system.

Semi-Thue systems

Introduced by Axel Thue (1863-1922)

- A rewriting system or semi-Thue system is a tuple (Σ, R) where
  - Σ is an alphabet, usually assumed finite.
  - R is a binary relation on strings, i.e., $R \subseteq \Sigma^* \times \Sigma^*$.
- Each element $(u, v) \in R$ is called a (rewriting) rule and is usually written $u \to v$.
- When R is symmetric the system is called a Thue system.
- The rewriting rules in R are extended to strings, for any strings $s, t \in \Sigma^*$,
  $s \longrightarrow t$ iff there exist $x, y, u, v \in \Sigma^*$ s.t. $s = xuy$, $t = xvy$, and $u \stackrel{R}{\to} v$.

Semi-Thue systems

Introduced by Axel Thue (1863-1922)

- A rewriting system or semi-Thue system is a tuple (Σ, R) where
  - Σ is an alphabet, usually assumed finite.
  - R is a binary relation on strings, i.e., $R \subseteq \Sigma^* \times \Sigma^*$.
- Each element $(u, v) \in R$ is called a (rewriting) rule and is usually written $u \to v$.
- When R is symmetric the system is called a Thue system.
- The rewriting rules in R are extended to strings, for any strings $s, t \in \Sigma^*$,
  $s \underset{R}{\longrightarrow} t$ iff there exist $x, y, u, v \in \Sigma^*$ s.t. $s = xuy$, $t = xvy$, and $u \overset{R}{\to} v$.
- A zero-or-more-steps rewriting is captured by the reflexive transitive closure of $\underset{R}{\longrightarrow}$, $\overset{*}{\underset{R}{\longrightarrow}}$.

The word problem for finite semi-Thue systems

WP: Given a finite list of rewriting rules R and two words u, v, $u \xrightarrow[R]{*} v$?

The word problem for finite semi-Thue systems

WP: Given a finite list of rewriting rules R and two words u, v, $u \xrightarrow[R]{*} v$?

rules	$a \to aa \quad bb \to b \quad aaaaaa \to bbbb$
words	start b end a
solution	there is no solution

The word problem for finite semi-Thue systems

WP: Given a finite list of rewriting rules R and two words u, v, $u \xrightarrow[R]{*} v$?

rules	$a \to aa \quad bb \to b \quad aaaaaa \to bbbb$
words	start b end a
solution	there is no solution

rules	$a \to aa \quad bb \to b \quad aaaaaa \to bbbb$
words	start aba endl ba
solution	$aba \to aaba \to aaaba \to aaaaba \to aaaaaba \to aaaaaaba$ $\to bbbbba \to bbbba \to bbba \to bba \to ba$

WP is undecidable

Theorem

$A_{TM} \leq_m \mathrm{WP}$

- Let $\langle M, x\rangle$ be an input to A_{TM}, $x \in \Sigma^*$ and $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$.

WP is undecidable

Theorem

$A_{TM} \leq_m \mathrm{WP}$

- Let $\langle M, x\rangle$ be an input to A_{TM}, $x \in \Sigma^*$ and $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$.
- From M and x, we construct the following input to WP with alphabet the union of Γ, Q and an additional symbol $\#$
  - The rewriting rules are

$$\begin{array}{ll} \multicolumn{2}{l}{\text{for } q, p \in Q \text{, with } q \neq q_F \text{ and for } a, b, c \in \Gamma} \\ (qa, bp) & \text{si } \delta(q, a) = (p, b, \mathtt{r}) \\ (cqa, pcb) & \text{si } \delta(q, a) = (p, b, \mathtt{l}) \\ (q\#, bp\#) & \text{si } \delta(q, \mathtt{b}) = (p, b, \mathtt{r}) \\ (cq\#, pcb\#) & \text{si } \delta(q, \mathtt{b}) = (p, b, \mathtt{l}) \\ \multicolumn{2}{l}{\text{for } a \in \Gamma,\ (aq_F, q_F) \text{ and } (q_F a, q_F)} \end{array}$$

  - The start word is $u = \blacktriangleright q_0 x\#$, and the final word is $v = q_F\#$

WP is undecidable

Theorem

$A_{TM} \leq_m \mathrm{WP}$

- Let $\langle M, x\rangle$ be an input to A_{TM}, $x \in \Sigma^*$ and $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$.
- From M and x, we construct the following input to WP with alphabet the union of Γ, Q and an additional symbol $\#$
  - The rewriting rules are

$$\begin{array}{l} \text{for } q, p \in Q, \text{ with } q \neq q_F \text{ and for } a, b, c \in \Gamma \\ \qquad \begin{array}{ll} (qa, bp) & \text{si } \delta(q, a) = (p, b, \mathtt{r}) \\ (cqa, pcb) & \text{si } \delta(q, a) = (p, b, \mathtt{l}) \\ (q\#, bp\#) & \text{si } \delta(q, \mathtt{b}) = (p, b, \mathtt{r}) \\ (cq\#, pcb\#) & \text{si } \delta(q, \mathtt{b}) = (p, b, \mathtt{l}) \end{array} \\ \text{for } a \in \Gamma, (aq_F, q_F) \text{ and } (q_F a, q_F) \end{array}$$

  - The start word is $u = \blacktriangleright q_0 x\#$, and the final word is $v = q_F\#$
- $x \in L(M)$ iff there is a finite computation $\blacktriangleright q_0 x, \ldots, \alpha q_F \beta$ iff $\blacktriangleright q_0 x\# \xrightarrow[R]{*} q_F\#$

Post Correspondence Problem

Introduced by by Emil Post in 1946.

PCP: Given two lists of words with the same length $A = (x_1, \ldots, x_n)$ and $B = (y_1, \ldots, y_n)$. Is there a finite sequence $(i_1, \ldots, i_r)$, $r \geq 1$ such that $x_{i_1} \cdots x_{i_r} = y_{i_1} \cdots y_{i_r}$?

Post Correspondence Problem

Introduced by by Emil Post in 1946.

PCP: Given two lists of words with the same length $A = (x_1, \ldots, x_n)$ and $B = (y_1, \ldots, y_n)$. Is there a finite sequence $(i_1, \ldots, i_r)$, $r \geq 1$ such that $x_{i_1} \cdots x_{i_r} = y_{i_1} \cdots y_{i_r}$?

	A	B
1	01	011
2	10	010
3	001	01
4	1011	10

Post Correspondence Problem

Introduced by by Emil Post in 1946.

PCP: Given two lists of words with the same length $A = (x_1, \ldots, x_n)$ and $B = (y_1, \ldots, y_n)$. Is there a finite sequence $(i_1, \ldots, i_r)$, $r \geq 1$ such that $x_{i_1} \cdots x_{i_r} = y_{i_1} \cdots y_{i_r}$?

	A	B
1	01	011
2	10	010
3	001	01
4	1011	10

Is a "yes" instance as $(1, 2, 2, 1, 3, 4, 3)$ gives 01101001001101 1001 in both systems.

Post Correspondence Problem

PCP: Given two lists of words with the same length $A = (x_1, \ldots, x_n)$ and $B = (y_1, \ldots, y_n)$. Is there a finite sequence $(i_1, \ldots, i_r)$, $r \geq 1$ such that $x_{i_1} \cdots x_{i_r} = y_{i_1} \cdots y_{i_r}$?

Post Correspondence Problem

PCP: Given two lists of words with the same length $A = (x_1, \ldots, x_n)$ and $B = (y_1, \ldots, y_n)$. Is there a finite sequence $(i_1, \ldots, i_r)$, $r \geq 1$ such that $x_{i_1} \cdots x_{i_r} = y_{i_1} \cdots y_{i_r}$?

	A	B
1	101	0100
2	100	10
3	0110	01
4	1010	10

Post Correspondence Problem

PCP: Given two lists of words with the same length $A = (x_1, \dots, x_n)$ and $B = (y_1, \dots, y_n)$. Is there a finite sequence $(i_1, \dots, i_r)$, $r \geq 1$ such that $x_{i_1} \cdots x_{i_r} = y_{i_1} \cdots y_{i_r}$?

	A	B
1	101	0100
2	100	10
3	0110	01
4	1010	10

Is a "no" instance, the number of ones in words 1,3,4 is higher in A than in B and equal for word 2.

Modified Post Correspondence Problem

MPCP: Given two lists of words with the same length $A = (x_1, \dots, x_n)$ and $B = (y_1, \dots, y_n)$. Is there a finite sequence $(1, i_2 \dots, i_r)$, $r \geq 1$ such that $x_1 x_{i_2} \cdots x_{i_r} = y_1 y_{i_2} \cdots y_{i_r}$?

Modified Post Correspondence Problem

MPCP: Given two lists of words with the same length $A = (x_1, \ldots, x_n)$ and $B = (y_1, \ldots, y_n)$. Is there a finite sequence $(1, i_2 \ldots, i_r)$, $r \geq 1$ such that $x_1 x_{i_2} \cdots x_{i_r} = y_1 y_{i_2} \cdots y_{i_r}$?

- The main difference is that we are forcing to start with the first word in A and B,
- MPCP is a subproblem of PCP,
- If PCP is decidable, MPCP is decidable.
- We wan to prove that MPCP is undecidable
- Then, extend this result to show that PCP is undecidable.

MPCP is undecidable

Theorem

$WP \leq_m MPCP$

MPCP is undecidable

Theorem

$\mathit{WP} \leq_m \mathit{MPCP}$

Proof

- Let (R, u, v) be an input to WP with $R = \{(u_1, v_1), \ldots, (u_n, v_n)\}$ over an alphabet Σ.

MPCP is undecidable

Theorem

$\mathit{WP} \leq_m \mathit{MPCP}$

Proof

- Let (R, u, v) be an input to WP with $R = \{(u_1, v_1), \ldots, (u_n, v_n)\}$ over an alphabet Σ.
- Let $\#$ be a symbol not in Σ.

MPCP is undecidable

Theorem

$\mathtt{WP} \leq_m \mathtt{MPCP}$

Proof

- Let (R, u, v) be an input to $\mathtt{WP}$ with $R = \{(u_1, v_1), \ldots, (u_n, v_n)\}$ over an alphabet Σ.
- Let $\#$ be a symbol not in Σ.
- Define (A, B) as

A	B	
$\#$	$\#u\#$	
u_i	v_i	$(u_i, v_i) \in R$
a	a	$a \in \Sigma$
$\#$	$\#$	
$v\#\#$	$\#$	

MPCP is undecidable

A	B	
$\#$	$\#u\#$	
u_i	v_i	$(u_i, v_i) \in R$
a	a	$a \in \Sigma$
$\#$	$\#$	
$v\#\#$	$\#$	

- When $(R, u, v) \in \mathtt{WP}$, there is a word $\#u\#x_1\# \dots \#x_m\#v\#$, where each intermediate word is obtained by applying a rule to the previous word.

MPCP is undecidable

A	B	
$\#$	$\#u\#$	
u_i	v_i	$(u_i, v_i) \in R$
a	a	$a \in \Sigma$
$\#$	$\#$	
$v\#\#$	$\#$	

- When $(R, u, v) \in \mathtt{WP}$, there is a word $\#u\#x_1\# \dots \#x_m\#v\#$, where each intermediate word is obtained by applying a rule to the previous word.
- We can reach this word starting from the first rule, and applying the corresponding rewriting rule. In B we go one word/letter in advance with respect to A. Using the last rule, we balance at the end.

MPCP is undecidable

A	B	
$\#$	$\#u\#$	
u_i	v_i	$(u_i, v_i) \in R$
a	a	$a \in \Sigma$
$\#$	$\#$	
$v\#\#$	$\#$	

- When $(R, u, v) \in \texttt{WP}$, there is a word $\#u\#x_1\# \dots \#x_m\#v\#$, where each intermediate word is obtained by applying a rule to the previous word.
- We can reach this word starting from the first rule, and applying the corresponding rewriting rule. In B we go one word/letter in advance with respect to A. Using the last rule, we balance at the end.
- So, $(A, B) \in \texttt{MCPC}$.

MPCP is undecidable

- Assume $(A, B) \in \texttt{MCPC}$.

A	B	
$\#$	$\#u\#$	
u_i	v_i	$(u_i, v_i) \in R$
a	a	$a \in \Sigma$
$\#$	$\#$	
$v\#\#$	$\#$	

MPCP is undecidable

A	B	
$\#$	$\#u\#$	
u_i	v_i	$(u_i, v_i) \in R$
a	a	$a \in \Sigma$
$\#$	$\#$	
$v\#\#$	$\#$	

- Assume $(A, B) \in \texttt{MCPC}$.
- The solution must start by the first rule and finalize with the last, so there is a common word $\#u\#x_1\#\dots\#x_m\#v\#$.

MPCP is undecidable

A	B
$\#$	$\#u\#$
u_i	v_i
a	a
$\#$	$\#$
$v\#\#$	$\#$

$(u_i, v_i) \in R$

$a \in \Sigma$

- Assume $(A, B) \in \texttt{MCPC}$.
- The solution must start by the first rule and finalize with the last, so there is a common word $\#u\#x_1\# \ldots \#x_m\#v\#$.
- As the correspondence copies or rewrites according to the Thue system, each intermediate word is obtained by applying a rule to the previous word.

MPCP is undecidable

A	B	
$\#$	$\#u\#$	
u_i	v_i	$(u_i, v_i) \in R$
a	a	$a \in \Sigma$
$\#$	$\#$	
$v\#\#$	$\#$	

- Assume $(A, B) \in \texttt{MCPC}$.
- The solution must start by the first rule and finalize with the last, so there is a common word $\#u\#x_1\# \ldots \#x_m\#v\#$.
- As the correspondence copies or rewrites according to the Thue system, each intermediate word is obtained by applying a rule to the previous word.
- So,$(R, u, v) \in \texttt{WP}$.

MPCP is undecidable

A	B	
$\#$	$\#u\#$	
u_i	v_i	$(u_i, v_i) \in R$
a	a	$a \in \Sigma$
$\#$	$\#$	
$v\#\#$	$\#$	

- Assume $(A, B) \in \texttt{MCPC}$.
- The solution must start by the first rule and finalize with the last, so there is a common word $\#u\#x_1\#\ldots\#x_m\#v\#$.
- As the correspondence copies or rewrites according to the Thue system, each intermediate word is obtained by applying a rule to the previous word.
- So,$(R, u, v) \in \texttt{WP}$.

End Proof