Université Paris Cité – LIPADE

Algorithmic Complexity

Turing Machines and Decidability

Jean-Guy Mailly (jean-guy.mailly@u-paris.fr)

2022

Some General Knowledge
Alan Mathison Turing

- UK, 1912 – 1954
- Mathematician, computer scientist, cryptanalyst
- Most famous works:
  - Computation model (« Turing Machines »)
  - Work on Enigma during WWII
  - Imitation Game (« Turing Test »)
- Father of theoretical CS and AI
- Turing Award $\simeq$ Nobel Award for CS

Outline

Turing Machines

Computing Machine [Turing 1936]

- Abstract model to compute any « calculable » decimal number i.e. any number which can be computed with a finite amount of resources
- Proposed years before the apparition of computer, but quite a faithful abstraction of modern machines
- Roughly speaking,
  - tape (sequence of squares) browsed by a reader/writer: memory of a computer
  - transition function: processor of a computer

Definition of Turing Machines

A Turing machine is a tuple $\langle Q, \Gamma, B, \Sigma, q_0, \delta, F\rangle$ with:

- $Q = \{q_0, q_1, \dots, q_m\}$, a finite set of *states*
- Γ, the finite set of symbols used by the machine (*vocabulary*)
- $B \in \Gamma$, a particular symbol called *blank*
- $\Sigma \subseteq \Gamma$, the *input vocabulary*
- $q_0 \in Q$, the *initial state* of the machine
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$, the *transition function*
- $F \subseteq Q$, the set of *final states*

Transition Function

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$$

- Input: a pair (current state, symbol on the tape) called *configuration*
- Output: a tuple (next state, symbol to write, move)

Example of Turing Machines (1/2)
Multiplying Integers by 2

Let us consider $\mathcal{M} = \langle Q, \Gamma, B, \Sigma, q_0, \delta, F \rangle$ with:

- $Q = \{in\_progress, done\}$
- $\Gamma = \{0, 1, B\}$
- $\Sigma = \{0, 1\}$
- $q_0 = in\_progress$
- δ as described in the table
- $F = \{done\}$

Current state	Current symbol	Next State	Write	Move
in_progress	0	*in_progress*	0	*R*
in_progress	1	*in_progress*	1	*R*
in_progress	*B*	*done*	0	*R*
done	STOP			

Example of Turing Machines (1/2)

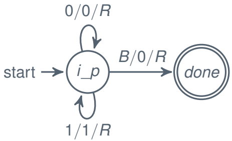
Multiplying Integers by 2

Let us consider $\mathcal{M} = \langle Q, \Gamma, B, \Sigma, q_0, \delta, F \rangle$ with:

- $Q = \{in\_progress, done\}$
- $\Gamma = \{0, 1, B\}$
- $\Sigma = \{0, 1\}$
- $q_0 = in\_progress$
- δ as described in the table
- $F = \{done\}$

Current state	Current symbol	Next State	Write	Move
in_progress	0	*in_progress*	0	*R*
in_progress	1	*in_progress*	1	*R*
in_progress	*B*	*done*	0	*R*
done	STOP			

Alternative notation of δ

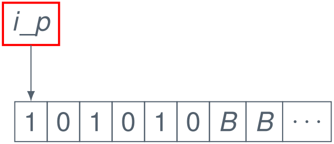


On each transition, $X/Y/Z$ is:

- X: the current symbol
- Y: the new symbol
- Z: the move

Example of Turing Machines (2/2)
Multiplying Integers by 2

- Read 1 in state *in_progress*: write 1, move right, state *in_progress*



Multiplying Integers by 2

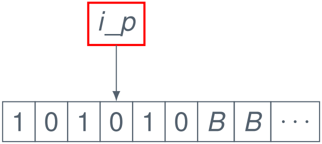
- Read 0 in state *in_progress*: write 0, move right, state *in_progress*

- Read 1 in state *in_progress*: write 1, move right, state *in_progress*

Example of Turing Machines (2/2)

Multiplying Integers by 2

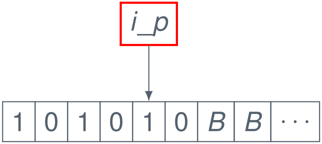
- Read 0 in state *in_progress*: write 0, move right, state *in_progress*



Example of Turing Machines (2/2)

Multiplying Integers by 2

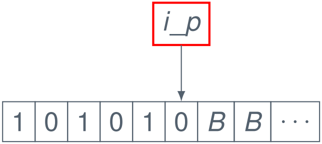
- Read 1 in state *in_progress*: write 1, move right, state *in_progress*



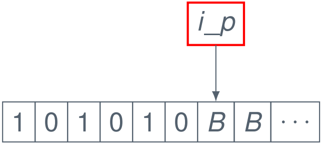
Example of Turing Machines (2/2)

Multiplying Integers by 2

- Read 0 in state *in_progress*: write 0, move right, state *in_progress*

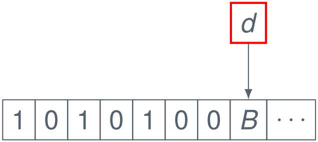


- Read *B* in state *in_progress*: write 0, move right, state *done*



Example of Turing Machines (2/2)
Multiplying Integers by 2

- State *done*: stop



Accept vs Reject vs Loop

We consider a special class of Turing machines, with three possible « results ». For an input x, we say that the Turing machine $\mathcal{M}$

- *accepts* x if $\mathcal{M}$ reaches the final state YES after a finite number of steps (i.e. transitions)
- *rejects* x if $\mathcal{M}$ reaches the final state NO after a finite number of steps
- *loops* on x if $\mathcal{M}$ never reaches a final state

Language of a Turing Machine

We call *language* of a Turing Machine $\mathcal{M}$ the set of all inputs which are accepted by $\mathcal{M}$

$$\mathcal{L}(\mathcal{M}) = \{x \in \Sigma^* \mid \mathcal{M} \text{ stops on } x \text{ and } \mathcal{M}(x) = \text{ YES}\}$$

Example

If $\mathcal{M}$ returns YES on inputs $P \in \mathbb{N}[X]$ with exactly one root, then $\mathcal{L}(\mathcal{M}) = \{P \in \mathbb{N}[X] \mid P \text{ is a polynomial with exactly one root } \}$

Language of a Turing Machine

We call *language* of a Turing Machine $\mathcal{M}$ the set of all inputs which are accepted by $\mathcal{M}$

$$\mathcal{L}(\mathcal{M}) = \{x \in \Sigma^* \mid \mathcal{M} \text{ stops on } x \text{ and } \mathcal{M}(x) = \text{ YES}\}$$

Example

If $\mathcal{M}$ returns YES on inputs $P \in \mathbb{N}[X]$ with exactly one root, then $\mathcal{L}(\mathcal{M}) = \{P \in \mathbb{N}[X] \mid P \text{ is a polynomial with exactly one root } \}$

Language $\simeq$ Problem

We associate the language $\mathcal{L}(\mathcal{M})$ of a Turing Machine $\mathcal{M}$ to a decision problem $\mathcal{P}$

$$x \in \mathcal{L}(\mathcal{M}) \text{ iff } x \text{ is a positive instance of } \mathcal{P}$$

(Non-)Deterministic Turing Machines

Definition

Given $\mathcal{M}$ a Turing machine and δ its transition function, $\mathcal{M}$ is *deterministic* (DTM) if δ is a mapping from any configuration (q, x) to (at most) a single image (q', x', m)
Otherwise, $\mathcal{M}$ is *non-deterministic* (NDTM)

(Non-)Deterministic Turing Machines

Definition

Given $\mathcal{M}$ a Turing machine and δ its transition function, $\mathcal{M}$ is *deterministic* (DTM) if δ is a mapping from any configuration (q, x) to (at most) a single image (q', x', m)
Otherwise, $\mathcal{M}$ is *non-deterministic* (NDTM)

Solving a Decision Problem

- A DTM solves a decision problem $\mathcal{P}$ if the sequence of transitions leads to the answer for any instance
- Not so easy for NDTM: when the transition function has several images, each of them must be checked
  The machine solves the problem if
  - for each positive instance, at least one sequence of transitions leads to a final state with the answer YES
  - for each negative instance, every possible sequence of transitions leads to a final state with the answer NO

Illustration: DTM vs NDTM

Deterministic

$$\begin{tikzcd}
(q_0, x_0) \arrow[d, "\delta"'] \\
(q_1, x_1) \arrow[d, "\delta"'] \\
\dots \arrow[d, "\delta"'] \\
(q_n, x_n)
\end{tikzcd}$$

Non-Deterministic

$$\begin{tikzcd}
 & & (q_0, x_0) \arrow[dl, "\delta"'] \arrow[d, "\delta"'] \arrow[dr, "\delta"] & \\
 & (q_1, x_1) \arrow[dl, "\delta"'] & \dots \arrow[d, "\delta"'] & (q_k, x_k) \arrow[d, "\delta"'] \\
\dots \arrow[d, "\delta"'] & & \dots \arrow[dl, "\delta"'] \arrow[d, "\delta"'] & \dots \arrow[d, "\delta"'] \\
(q_m, x_m) & (q_n, x_n) & (q_o, x_o) & (q_p, x_p)
\end{tikzcd}$$

Other Computing Models

Other models have been proposed

- Turing machines with several tapes/several dimensions/several readers-writers
- λ-calculus [Church 1936]
- Kolmogorov-Uspensky machines [Kolmogorov and Uspensky 1958]
- Schönhage machines [Schönhage 1980]
- ...

Power of Turing machines: the Church-Turing Thesis

Everything which can be computed with any of these models can be computed with a Turing machine

Other Computing Models

Other models have been proposed

- Turing machines with several tapes/several dimensions/several readers-writers
- λ-calculus [Church 1936]
- Kolmogorov-Uspensky machines [Kolmogorov and Uspensky 1958]
- Schönhage machines [Schönhage 1980]
- ...

Power of Turing machines: the Church-Turing Thesis

Everything which can be computed with any of these models can be computed with a Turing machine

Turing completeness

A system (computing model, programming language,...) is called *Turing complete* if it can compute every possible computable function

Outline

Recognizability

Recognizability of a Problem

A problem $\mathcal{P}$ is called *recognizable* if there exists a Turing machine $\mathcal{M}$ such that, for each instance *i* of $\mathcal{P}$, $\mathcal{M}(i)$ answers YES iff *i* is a positive instance of $\mathcal{P}$

Recognizability of a Language

A language $\mathcal{L}$ is called *recognizable* if there exists a Turing machine $\mathcal{M}$ such that $\mathcal{L} = \mathcal{L}(\mathcal{M})$

Recognizability

Recognizability of a Problem

A problem $\mathcal{P}$ is called *recognizable* if there exists a Turing machine $\mathcal{M}$ such that, for each instance i of $\mathcal{P}$, $\mathcal{M}(i)$ answers YES iff i is a positive instance of $\mathcal{P}$

Recognizability of a Language

A language $\mathcal{L}$ is called *recognizable* if there exists a Turing machine $\mathcal{M}$ such that $\mathcal{L} = \mathcal{L}(\mathcal{M})$

Prime Numbers

- Is $n \in \mathbb{N}$ a prime number?

This problem/language is recognizable: we can write an algorithm which answers YES when n is prime, and doesn't care of non-prime numbers

Decidability

Decidability of a Problem

A problem $\mathcal{P}$ is called *decidable* if there exists a Turing machine $\mathcal{M}$ such that, for each instance i of $\mathcal{P}$,

- $\mathcal{M}(i)$ answers YES when i is a positive instance of $\mathcal{P}$
- $\mathcal{M}(i)$ answers NO when i is a negative instance of $\mathcal{P}$

Decidability of a Problem

A language $\mathcal{L}$ is called *decidable* if there exists a Turing machine $\mathcal{M}$ which accepts each $x \in \mathcal{L}$ and rejects each $x \notin \mathcal{L}$
Equivalently: $\mathcal{L}$ is decidable iff both $\mathcal{L}$ and $\bar{\mathcal{L}}$ are recognizable

Decidability

Decidability of a Problem

A problem $\mathcal{P}$ is called *decidable* if there exists a Turing machine $\mathcal{M}$ such that, for each instance i of $\mathcal{P}$,

- $\mathcal{M}(i)$ answers YES when i is a positive instance of $\mathcal{P}$
- $\mathcal{M}(i)$ answers NO when i is a negative instance of $\mathcal{P}$

Decidability of a Problem

A language $\mathcal{L}$ is called *decidable* if there exists a Turing machine $\mathcal{M}$ which accepts each $x \in \mathcal{L}$ and rejects each $x \notin \mathcal{L}$
Equivalently: $\mathcal{L}$ is decidable iff both $\mathcal{L}$ and $\bar{\mathcal{L}}$ are recognizable

Prime Numbers

- Is $n \in \mathbb{N}$ a prime number?

This problem/language is decidable: we can write an algorithm which answers YES when n is prime, and NO otherwise

Complement of a Decision Problem

Definition

A problem $\mathcal{P}_1$ is called *the complement of* $\mathcal{P}_2$ $\mathcal{L}(\mathcal{P}_1) = \overline{\mathcal{L}(\mathcal{P}_2)}$.
Equivalently:

- $\mathcal{P}_1$ and $\mathcal{P}_2$ are defined on the same set of instances
- Positive instances of $\mathcal{P}_1$ are negative instances of $\mathcal{P}_2$
- Negative instances of $\mathcal{P}_1$ are positive instances of $\mathcal{P}_2$

Notation: $\mathcal{P}_1 = \overline{\mathcal{P}_2}$

Complement of a Decision Problem

Definition

A problem $\mathcal{P}_1$ is called *the complement of* $\mathcal{P}_2$ $\mathcal{L}(\mathcal{P}_1) = \overline{\mathcal{L}(\mathcal{P}_2)}$.
Equivalently:

- $\mathcal{P}_1$ and $\mathcal{P}_2$ are defined on the same set of instances
- Positive instances of $\mathcal{P}_1$ are negative instances of $\mathcal{P}_2$
- Negative instances of $\mathcal{P}_1$ are positive instances of $\mathcal{P}_2$

Notation: $\mathcal{P}_1 = \overline{\mathcal{P}_2}$

Prime Numbers

- Given $n \in \mathbb{N}$, is there $m \in \mathbb{N}$, $m \neq 1$, $m \neq n$, s.t. m divides n?

This problem is the complement of the prime number problem.

Halting Problem

Is there a Turing machine $\mathcal{H}$ which has two parameters:

- A Turing machine $\mathcal{M}$
- An input i of $\mathcal{M}$

and which returns

- YES when $\mathcal{M}$ stops on i
- NO when $\mathcal{M}$ loops on i

If there is such a machine, then Halting problem is decidable; otherwise, it is undecidable

Halting Problem

Is there a Turing machine $\mathcal{H}$ which has two parameters:

- A Turing machine $\mathcal{M}$
- An input *i* of $\mathcal{M}$

and which returns

- YES when $\mathcal{M}$ stops on *i*
- NO when $\mathcal{M}$ loops on *i*

If there is such a machine, then Halting problem is decidable; otherwise, it is undecidable

Result from [Turing 1936]

Halting is undecidable

Reducing a Problem to Another

Functional Reduction

A functional reduction f is a total computable function from a problem $\mathcal{P}_1$ to a problem $\mathcal{P}_2$ such that, for any instance i of $\mathcal{P}_1$, i is a positive instance iff $f(i)$ is a positive instance of $\mathcal{P}_2$
Notation:

$$\mathcal{P}_1 \leq_f \mathcal{P}_2$$

Reducing a Problem to Another

Functional Reduction

A functional reduction f is a total computable function from a problem $\mathcal{P}_1$ to a problem $\mathcal{P}_2$ such that, for any instance i of $\mathcal{P}_1$, i is a positive instance iff $f(i)$ is a positive instance of $\mathcal{P}_2$
Notation:

$$\mathcal{P}_1 \leq_f \mathcal{P}_2$$

Theorem

If $\mathcal{P}_1 \leq_f \mathcal{P}_2$ and $\mathcal{P}_1$ is undecidable, then $\mathcal{P}_2$ is undecidable

(Trivial) Example of Functional Reduction

- $\mathcal{P}_1$: Given $n \in \mathbb{N}$, is n even?
- $\mathcal{P}_2$: Given $n \in \mathbb{N}$, is n odd?
- $f : \mathbb{N} \to \mathbb{N}$ is defined by $f(n) = n + 1$

(Trivial) Example of Functional Reduction

- $\mathcal{P}_1$: Given $n \in \mathbb{N}$, is n even?
- $\mathcal{P}_2$: Given $n \in \mathbb{N}$, is n odd?
- $f : \mathbb{N} \rightarrow \mathbb{N}$ is defined by $f(n) = n + 1$

For a positive instance i of $\mathcal{P}_1$, $f(i)$ is a positive instance of $\mathcal{P}_2$.
For a negative instance i of $\mathcal{P}_1$, $f(i)$ is a negative instance of $\mathcal{P}_2$.

(Trivial) Example of Functional Reduction

- $\mathcal{P}_1$: Given $n \in \mathbb{N}$, is n even?
- $\mathcal{P}_2$: Given $n \in \mathbb{N}$, is n odd?
- $f : \mathbb{N} \rightarrow \mathbb{N}$ is defined by $f(n) = n + 1$

For a positive instance i of $\mathcal{P}_1$, $f(i)$ is a positive instance of $\mathcal{P}_2$.
For a negative instance i of $\mathcal{P}_1$, $f(i)$ is a negative instance of $\mathcal{P}_2$.
So $\mathcal{P}_1 \leq_f \mathcal{P}_2$

(Trivial) Example of Functional Reduction

- $\mathcal{P}_1$: Given $n \in \mathbb{N}$, is n even?
- $\mathcal{P}_2$: Given $n \in \mathbb{N}$, is n odd?
- $f : \mathbb{N} \to \mathbb{N}$ is defined by $f(n) = n + 1$

For a positive instance i of $\mathcal{P}_1$, $f(i)$ is a positive instance of $\mathcal{P}_2$.
For a negative instance i of $\mathcal{P}_1$, $f(i)$ is a negative instance of $\mathcal{P}_2$.
So $\mathcal{P}_1 \leq_f \mathcal{P}_2$
Here we also have $\mathcal{P}_2 \leq_f \mathcal{P}_1$; this is trivial since these are decidable problems.

A Less Trivial Example

Language

$$\mathcal{L}_1 = \{\mathcal{M} \mid \mathcal{M} \text{ halts on } \epsilon\}$$

with ϵ the empty word. Is $\mathcal{L}_1$ decidable?

A Less Trivial Example

Language

$$\mathcal{L}_1 = \{\mathcal{M} \mid \mathcal{M} \text{ halts on } \epsilon\}$$

with ϵ the empty word. Is $\mathcal{L}_1$ decidable?

Proof of Undecidability

We need a mapping $f : (\mathcal{M}, x) \mapsto \mathcal{M}'$ s.t. $(\mathcal{M}, x) \in$ Halting iff $\mathcal{M}' \in \mathcal{L}_1$

A Less Trivial Example

Language

$$\mathcal{L}_1 = \{\mathcal{M} \mid \mathcal{M} \text{ halts on } \epsilon\}$$

with ϵ the empty word. Is $\mathcal{L}_1$ decidable?

Proof of Undecidability

We need a mapping $f : (\mathcal{M}, x) \mapsto \mathcal{M}'$ s.t. $(\mathcal{M}, x) \in$ Halting iff $\mathcal{M}' \in \mathcal{L}_1$

- $\mathcal{M}'_x(y) = \mathcal{M}(x.y)$ with $x.y$ the concatenation of words
- $\mathcal{M}'_x(\epsilon) = \mathcal{M}(x)$, so obviously:
  - If $\mathcal{M}$ halts on x, then $\mathcal{M}'_x$ halts on ϵ
  - If $\mathcal{M}$ loops on x, then $\mathcal{M}'_x$ loops on ϵ

Other Method to Prove Undecidability

« Algorithmic » Proof

If $\mathcal{P}_1$ is known to be undecidable, we can prove that $\mathcal{P}_2$ is undecidable with an algorithm which computes $\mathcal{P}_1$ using only « simple » steps and calls to $\mathcal{P}_2$

Other Method to Prove Undecidability

« Algorithmic » Proof

If $\mathcal{P}_1$ is known to be undecidable, we can prove that $\mathcal{P}_2$ is undecidable with an algorithm which computes $\mathcal{P}_1$ using only « simple » steps and calls to $\mathcal{P}_2$

Remember: $\mathcal{L}_1 = \{\mathcal{M} \mid \mathcal{M}$ halts on $\epsilon\}$. Suppose that $\mathcal{M}_{\mathcal{L}_1}$ is a Turing Machine which decides $\mathcal{L}_1$.

Algorithm 2 Halting

Input: $\mathcal{M}, x = x_0 x_1 \dots x_n$
  if $x = \epsilon$ **then**
    return $\mathcal{M}_{\mathcal{L}_1}(\mathcal{M})$
  else
    $x' = x_1 \dots x_n$
    $\mathcal{M}' = \mathcal{M}$ with q_0 replaced by $\delta(q_0, x_0)$
    return Halting$(\mathcal{M}', x')$
  end if

Other Method to Prove Undecidability

« Algorithmic » Proof

If $\mathcal{P}_1$ is known to be undecidable, we can prove that $\mathcal{P}_2$ is undecidable with an algorithm which computes $\mathcal{P}_1$ using only « simple » steps and calls to $\mathcal{P}_2$

Remember: $\mathcal{L}_1 = \{\mathcal{M} \mid \mathcal{M}$ halts on $\epsilon\}$. Suppose that $\mathcal{M}_{\mathcal{L}_1}$ is a Turing Machine which decides $\mathcal{L}_1$.

Algorithm 3 Halting

Input: $\mathcal{M}, x = x_0 x_1 \dots x_n$
  if $x = \epsilon$ **then**
    return $\mathcal{M}_{\mathcal{L}_1}(\mathcal{M})$
  else
    $x' = x_1 \dots x_n$
    $\mathcal{M}' = \mathcal{M}$ with q_0 replaced by $\delta(q_0, x_0)$
    return Halting$(\mathcal{M}', x')$
  end if

Other Method to Prove Undecidability

« Algorithmic » Proof

If $\mathcal{P}_1$ is known to be undecidable, we can prove that $\mathcal{P}_2$ is undecidable with an algorithm which computes $\mathcal{P}_1$ using only « simple » steps and calls to $\mathcal{P}_2$

Remember: $\mathcal{L}_1 = \{\mathcal{M} \mid \mathcal{M}$ halts on $\epsilon\}$. Suppose that $\mathcal{M}_{\mathcal{L}_1}$ is a Turing Machine which decides $\mathcal{L}_1$.

Algorithm 4 Halting

Input: $\mathcal{M}, x = x_0 x_1 \dots x_n$
  if $x = \epsilon$ **then**
    return $\mathcal{M}_{\mathcal{L}_1}(\mathcal{M})$
  else
    $x' = x_1 \dots x_n$
    $\mathcal{M}' = \mathcal{M}$ with q_0 replaced by $\delta(q_0, x_0)$
    return Halting$(\mathcal{M}', x')$
  end if

How to Prove Decidability?

- Prove $\mathcal{P}_1 \leq_f \mathcal{P}_2$ with $\mathcal{P}_2$ decidable, then $\mathcal{P}_1$ is decidable

How to Prove Decidability?

- Prove $\mathcal{P}_1 \leq_f \mathcal{P}_2$ with $\mathcal{P}_2$ decidable, then $\mathcal{P}_1$ is decidable
- Write an algorithm which solves your problem with only « simple » steps!

Of course, the more complex is the algorithm, the less easy it is to be sure that it is correct :(

How to Prove Decidability?

- Prove $\mathcal{P}_1 \leq_f \mathcal{P}_2$ with $\mathcal{P}_2$ decidable, then $\mathcal{P}_1$ is decidable
- Write an algorithm which solves your problem with only « simple » steps!

Of course, the more complex is the algorithm, the less easy it is to be sure that it is correct :(

Example: Is $n \in \mathbb{N}$ a prime number?

Algorithm 7 Prime

Input: $n \in \mathbb{N}$

  for $x \in \{2, \ldots, \lfloor\sqrt{n}\rfloor\}$ **do**
    if $\exists y \in \mathbb{N}$ s.t. $n = x \times y$ **then**
      return false
    end if
  end for
  return true

[Turing 1936] A. M. Turing, *On computable numbers, with an application to the Entscheidungsproblem*. Proceedings of the London Mathematical Society, 1936.

[Church 1936] A. Church, *An unsolvable problem of elementary number theory*. American Journal of Mathematics, 1936.

[Kolmogorov and Uspensky 1958] A. N. Kolmogorov and V. Uspensky, *On the definition of an algorithm*. Uspekhi Mat. Naut, 1958.

[Schönhage 1980] A. Schönhage, *Storage modification machines*. SIAM Journal on Computing, 1980.