Computational complexity

lecture 5

The parity language

PARITY – the language of those words $\{0,1\}$ in which the number of ones is even

Fact: PARITY$\in$**u-NC**^{1}

We count ones modulo 2 – circuit of tree-like shape.

Theorem (1986): PARITY$\notin$**AC**^{0}

AC^0 = circuits of polynomial size and constant depth (arbitrary fan-in)

- It is one of quite rare nontrivial proofs saying that some problem cannot be solved in some complexity class.
- (Mostly hardness theorems are relative – is a problem A is hard, then a problem B is hard, e.g. NP-completeness)

PARITY $\notin$ **AC**^{0}

General idea:

- Every circuit of small depth can be approximated by a proper polynomial of low degree (Lemma 1 – previous lecture)
- The parity function cannot be approximated by a polynomial of low degree (Lemma 2 – now)

Proof of Lemma 2 (∗)

Lemma 2. For large enough n every polynomial of n variables and total degree $\leq\sqrt{n}$ differs from the parity function on at least $\frac{1}{100}2^n$ inputs.

A general idea:

- We assume that there exists a polynomial of low degree which agrees with the parity function on a large set S of inputs.
- Using this polynomial, for every function we will construct a polynomial of low degree which agrees with this function on the same set S.
- There are many functions, but significantly less polynomials.
- Thus the set S cannot be too large.

Proof of Lemma 2 (∗)

<u>Lemma 2.</u> For large enough n every polynomial of n variables and total degree $\leq\sqrt{n}$ differs from the parity function on at least $\frac{1}{100}2^n$ inputs.

- Let $PAR(x_1,\ldots,x_n)$ denote the parity function
- Consider the „shifted" parity function $PAR':\{-1,1\}^n\rightarrow\{-1,1\}$ $PAR'(x_1,\ldots,x_n)=PAR(x_1-1,\ldots,x_n-1)+1=x_1\cdot x_2\cdot\ldots\cdot x_n$

Proof of Lemma 2 (∗)

Lemma 2. For large enough n every polynomial of n variables and total degree $\leq\sqrt{n}$ differs from the parity function on at least $\frac{1}{100}2^n$ inputs.

- Let $PAR(x_1,\ldots,x_n)$ denote the parity function
- Consider the „shifted" parity function $PAR':\{-1,1\}^n\to\{-1,1\}$ $PAR'(x_1,\ldots,x_n)=PAR(x_1-1,\ldots,x_n-1)+1=x_1\cdot x_2\cdot\ldots\cdot x_n$
- If there exists a polynomial which agrees with PAR on some set of inputs, then there exists a polynomial of the same degree, which agrees with PAR' on the same set
- Thus take a polynomial p of degree $\leq\sqrt{n}$ approximating PAR' Let $S\subseteq\{-1,1\}^n$ be the set of those inputs in which p agrees with PAR'.

Proof of Lemma 2 (✶)

- A polynomial p of degree $\leq\sqrt{n}$ agrees with PAR' on a set $S\subseteq\{-1,1\}^n$.
- Take any function $f{:}S\rightarrow\mathbb{Z}_3$
- We can always represent f as a polynomial:

$$p_f(x_1,\ldots,x_n)=\sum_{(y_1,\ldots,y_n)\in S} f(y_1,\ldots,y_n)\cdot(2-x_1y_1)\cdot\ldots\cdot(2-x_ny_n)$$

- This polynomial has degree n, too large for us
- We will correct it so that the degree will be $\leq n/2+\sqrt{n}$

Proof of Lemma 2 (∗)

- A polynomial p of degree $\leq\sqrt{n}$ agrees with PAR' on a set $S\subseteq\{-1,1\}^n$.
- Take any function $f{:}S\rightarrow\mathbb{Z}_3$
- We can always represent f as a polynomial:

$$p_f(x_1,\ldots,x_n)=\sum_{(y_1,\ldots,y_n)\in S} f(y_1,\ldots,y_n)\cdot(2-x_1y_1)\cdot\ldots\cdot(2-x_ny_n)$$

- This polynomial has degree n, too large for us
- We will correct it so that the degree will be $\leq n/2+\sqrt{n}$
- To this end, in p_f we replace every monomial $\prod_{i\in T}x_i$ of degree $|T|>n/2$ by $p(x_1,\ldots,x_n)\cdot\prod_{i\notin T}x_i$

Proof of Lemma 2 (∗)

- A polynomial p of degree $\leq\sqrt{n}$ agrees with PAR' on a set $S\subseteq\{-1,1\}^n$.
- Take any function $f:S\rightarrow\mathbb{Z}_3$
- We can always represent f as a polynomial:

$$p_f(x_1,\ldots,x_n)=\sum_{(y_1,\ldots,y_n)\in S} f(y_1,\ldots,y_n)\cdot(2-x_1y_1)\cdot\ldots\cdot(2-x_ny_n)$$

- This polynomial has degree n, too large for us
- We will correct it so that the degree will be $\leq n/2+\sqrt{n}$
- To this end, in p_f we replace every monomial $\prod_{i\in T}x_i$ of degree $|T|>n/2$ by $p(x_1,\ldots,x_n)\cdot\prod_{i\notin T}x_i$
- This modification does not change the result, as for $(x_1,\ldots,x_n)\in S$ we have $p(x_1,\ldots,x_n)=x_1\cdot\ldots\cdot x_n$ and $(x_1)^2=1$
- Now the degree is indeed $\leq n/2+\sqrt{n}$

Proof of Lemma 2 (∗)

- A polynomial p of degree $\leq\sqrt{n}$ agrees with PAR' on a set $S\subseteq\{-1,1\}^n$.
- Take any function $f{:}S\to\mathbb{Z}_3$
- We can always represent f as a polynomial:

$$p_f(x_1,\ldots,x_n)=\sum_{(y_1,\ldots,y_n)\in S} f(y_1,\ldots,y_n)\cdot(2-x_1y_1)\cdot\ldots\cdot(2-x_ny_n)$$

- This polynomial has degree n, too large for us
- We will correct it so that the degree will be $\leq n/2+\sqrt{n}$
- To this end, in p_f we replace every monomial $\prod_{i\in T}x_i$ of degree $|T|>n/2$ by $p(x_1,\ldots,x_n)\cdot\prod_{i\notin T}x_i$
- This modification does not change the result, as for $(x_1,\ldots,x_n)\in S$ we have $p(x_1,\ldots,x_n)=x_1\cdot\ldots\cdot x_n$ and $(x_1)^2=1$
- Now the degree is indeed $\leq n/2+\sqrt{n}$
- Thus (using the hypothetical polynomial p) for every function $f{:}S\to\mathbb{Z}_3$ we have constructed a polynomial of degree $\leq n/2+\sqrt{n}$, which on S gives the same values as f

Proof of Lemma 2 (⋆)

- A polynomial p of degree $\leq\sqrt{n}$ agrees with PAR' on a set $S \subseteq \{-1,1\}^n$.
- For every function $f: S \rightarrow \mathbb{Z}_3$ we have constructed a polynomial of degree $\leq n/2+\sqrt{n}$, which on S gives the same values as f
- For inputs in $\{-1,1\}^n$ we have that $x^2=1$, so we can assume that in f there are no exponents greater than 1.

Proof of Lemma 2 (∗)

- A polynomial p of degree $\leq\sqrt{n}$ agrees with PAR' on a set $S\subseteq\{-1,1\}^n$.
- For every function $f\colon S\rightarrow\mathbb{Z}_3$ we have constructed a polynomial of degree $\leq n/2+\sqrt{n}$, which on S gives the same values as f
- For inputs in $\{-1,1\}^n$ we have that $x^2=1$, so we can assume that in the polynomial there are no exponents greater than 1.

Let us compute the number of such polynomials:

- For large enough n, there are $\leq 0.99\cdot 2^n$ monomials of n variables and degree $\leq n/2+\sqrt{n}$, using every variable at most once (next slide)
- Thus the number of polynomials is $\leq 3^{0.99\cdot 2^n}$
- The number of functions $f\colon S\rightarrow\mathbb{Z}_3$ is $3^{|S|}$, to each of them we have assigned a different polynomial
- Thus $|S|\leq 0.99\cdot 2^n$

Proof of Lemma 2 (∗)

Why the number of monomials (using variables $x_1,\ldots,x_n$, each of them either with exponent 0 or 1) of degree $\leq n/2+\sqrt{n}$ is $\leq 0.99\cdot 2^n$, for large enough n?

- Choose a monomial in random
- Let X_i=(does x_i appear in the monomial)
- Random variables X_i are independent and $P(X_i=0)=P(X_i=1)=0.5$
- Central limit theorem: for every $z\in\mathbb{R}$, $P(Z_n\leq z) \xrightarrow[n\to\infty]{} \Phi(z)$ where

$$Z_n = \frac{\sum_{i=1}^{n}(X_i-\mu)}{\sqrt{n}\sigma}$$

  and $\mu=EX_i=0.5$, $\sigma=sd(X_i)=0.5$, and Φ is the cumulative distribution function of the normal distribution $N(0,1)$
- Notice that $X_1+\ldots+X_n\leq n/2+\sqrt{n} \Leftrightarrow Z_n\leq 2$, and $\Phi(2)\approx 0{,}97725$
- Thus for large enough n, the probability that the degree is $\leq n/2+\sqrt{n}$ i.e., $P(Z_n\leq 2)$ is at most $0{,}99$

[THE END OF THE PROOF OF LEMMA 2]

Extensions of **AC**^{0}

Consider circuits like in **AC**^{0}, where additionally we can use the XOR gate. Then we can recognize PARITY.

Is it enough to recognize, e.g., all regular languages?

Extensions of **AC**^{0}

Consider circuits like in **AC**^{0}, where additionally we can use the XOR gate. Then we can recognize PARITY.
Is it enough to recognize, e.g., all regular languages?

- Class **AC**$^0[m]$ – like **AC**^{0}, but where we can additionally use gates counting the number of ones modulo m
- It is known that: if p,q are different prime numbers, then **AC**$^0[p]$ cannot count modulo q
- An open problem: we cannot show any language, even from **NP**, which cannot be recognized in **AC**$^0[6]$
  (gates „mod 6” ⇔ gates „mod 2” i gates „mod 3”)

Overview

Already finished:

- Deterministic Turing machines – basic facts
- Boolean circuits

Next topic:

- Nondeterministic Turing machines, reductions

Later:

- Probabilistic computations
- Fixed parameter tractability (FPT)
- Interactive proofs
- Alternating Turing machines
- Probabilistically checkable proofs (PCP)
- ...

Nondeterministic Turing machines

We introduce the following changes to the definition of Turing machines:

- a transition relation instead of a transition function:

$$\delta \subseteq Q \times \Gamma^k \times Q \times \Gamma^k \times \{L, R, Z\}^k$$

- there is no rejecting state (it is useless)

Nondeterministic Turing machines

We introduce the following changes to the definition of Turing machines:

- a transition <u>relation</u> instead of a transition function:

  $$\delta \subseteq Q \times \Gamma^k \times Q \times \Gamma^k \times \{L,R,Z\}^k$$

- there is no rejecting state (it is useless)
- ⇒ a transition relation on configurations
- a run of a machine: any sequence of configuration which respects the transition relation
- a machine accepts a word w if there <u>exists</u> an accepting run over this word

Nondeterministic Turing machines

We introduce the following changes to the definition of Turing machines:

- a transition <u>relation</u> instead of a transition function:

  $$\delta \subseteq Q \times \Gamma^k \times Q \times \Gamma^k \times \{L,R,Z\}^k$$

- there is no rejecting state (it is useless)
- ⇒ a transition relation on configurations
- a run of a machine: any sequence of configuration which respects the transition relation
- a machine accepts a word w if there <u>exists</u> an accepting run over this word
- A machine *works in time* $T(n)$ if <u>every</u> run (not only the accepting one) halts after at most $T(n)$ steps
- A machine *works in space* $S(n)$ if <u>every</u> run (not only the accepting one) uses at most $S(n)$ tape cells and <u>halts</u>

Nondeterministic Turing machines

A machine *works in space* $S(n)$ if every run (not only the accepting one) uses at most $S(n)$ tape cells and halts

- for a deterministic machine there was Sipser's theorem, saying that the halting property can be introduced without increasing memory usage (⇒ we could remove the condition “and halts” from the above definition)
- for a nondeterministic machine the Sipser's construction (simulating the computation backwards) does not work

Nondeterministic Turing machines

A machine *works in space* $S(n)$ if <u>every</u> run (not only the accepting one) uses at most $S(n)$ tape cells and <u>halts</u>

- for a deterministic machine there was Sipser's theorem, saying that the halting property can be introduced without increasing memory usage (⇒ we could remove the condition "and halts" from the above definition)
- for a nondeterministic machine the Sipser's construction (simulating the computation backwards) does not work
- but the construction with a counter of steps does work (if the number of steps has exceeded the maximal number of configurations for the current memory usage, then the machine entered a loop)
- this construction does not increase memery usage as soon as $S(n) \geq log(n)$
- thus the condition "and halts" is not so important

Nondeterministic Turing machines

- A machine *works in time* $T(n)$ if <u>every</u> run (not only the accepting one) halts after at most $T(n)$ steps
- A machine *works in space* $S(n)$ if <u>every</u> run (not only the accepting one) uses at most $S(n)$ tape cells and <u>halts</u>
- **NTIME**($T(n)$) – languages recognizable in time $O(T(n))$ on a nondeterministic machine
- **NSPACE**($S(n)$) – languages recognizable in space $O(S(n))$ on a nondeterministic machine

Nondeterministic Turing machines

- A machine *works in time* $T(n)$ if every run (not only the accepting one) halts after at most $T(n)$ steps
- A machine *works in space* $S(n)$ if every run (not only the accepting one) uses at most $S(n)$ tape cells and halts
- **NTIME**$(T(n))$ – languages recognizable in time $O(T(n))$ on a nondeterministic machine
- **NSPACE**$(S(n))$ – languages recognizable in space $O(S(n))$ on a nondeterministic machine
- **NL**=**NSPACE**$(\log n)$
- **NP**=$\bigcup_{k\in\mathbb{N}}$**NTIME**(n^k)
- **NPSPACE**=$\bigcup_{k\in\mathbb{N}}$**NSPACE**(n^k)
- itp.

Nondeterministic Turing machines

An example of a language in **NP** – the language of (codes of) these graphs in which there exists a Hamiltonian cycle

How do we recognize it?

- walk in the graph, arbitrarily choosing the next node to visit – remember visited nodes, and ensure that every node is visited at most once;
- if every node was visited (exactly once), and there is an edge to the starting node, then accept

A model with witnesses

An alternative definition of **NP** – using witnesses:

- A <u>relation</u> R is defined as the language of words of the form $v\$w$ (where $v,w\in\Sigma^*$ and $\$\notin\Sigma$)
- A relation R is called <u>polynomial</u> if:
  - ➔ $R\in$ **P** and
  - ➔ there exists a polynomial p such that $v\$w\in R$ implies $|w|\leq p(|v|)$
- The <u>projection</u> of a relation R is defined as $\exists R=\{v : \exists w.\ v\$w\in R\}$

A model with witnesses

An alternative definition of **NP** – using witnesses:

- A <u>relation</u> R is defined as the language of words of the form $v\$w$ (where $v,w\in\Sigma^*$ and $\$\notin\Sigma$)
- A relation R is called <u>polynomial</u> if:
  - ➔ $R\in$**P** and
  - ➔ there exists a polynomial p such that $v\$w\in R$ implies $|w|\leq p(|v|)$
- The <u>projection</u> of a relation R is defined as $\exists R=\{v : \exists w.\ v\$w\in R\}$

An example of a language in **NP** – the language of (codes of) these graphs in which there exists a Hamiltonian cycle

- it is of the form $\exists R$ for
  $R=\{$graph $\$$ consecutive nodes on a Hamiltonian cycle in this graph$\}$
- it is easy to recognize R in polynomial time
- the second part (a cycle) is no longer than the first one (a graph)

A model with witnesses

<u>Theorem</u>

$L \in$ **NP** $\Leftrightarrow$ there exists a polynomial relation R such that $L = \exists R$

A model with witnesses

<u>Theorem</u>

$L \in$ **NP** $\Leftrightarrow$ there exists a polynomial relation R such that $L = \exists R$

<u>Proof</u>

$\Leftarrow$ By definition, the length of witnesses is bounded by some polynomial p. We create a machine M, which after the input word (nondeterministically) writes an arbitrary word of length $\leq p(n)$ (in particular M counts the length of the word that it writes, and finishes writing it, if it gets longer than $p(n)$); then M executes the (deterministic) machine recognizing R.

A model with witnesses

Theorem

$L \in \mathbf{NP} \Leftrightarrow$ there exists a polynomial relation R such that $L = \exists R$

Proof

$\Leftarrow$ By definition, the length of witnesses is bounded by some polynomial p. We create a machine M, which after the input word (nondeterministically) writes an arbitrary word of length $\leq p(n)$ (in particular M counts the length of the word that it writes, and finishes writing it, if it gets longer than $p(n)$); then M executes the (deterministic) machine recognizing R.

$\Rightarrow$ L is recognized by a nondeterministic machine M in time $p(n)$. Then on every accepted word v there exists a sequence of transitions of M performed in consecutive steps of an accepting run; this sequence has length $\leq p(|v|)$. To R we take input words together with codes of accepting runs. This relation is polynomial; in particular, it can be recognized by a deterministic machine in polynomial time (remark: notice that a “transition” comes from a set of constant size)

A model with witnesses

<u>Theorem</u>

$L \in$ **NP** $\Leftrightarrow$ there exists a polynomial relation R such that $L = \exists R$

Similarily we can define another time-complexity classes, e.g., languages from **NEXPTIME** are projections of relations such that:

- ➔ can be recognized in **P**
- ➔ there exists an exponential function f such that $v\$w \in R$ implies $|w| \leq f(|v|)$

A model with witnesses

Theorem

$L \in$ **NP** $\Leftrightarrow$ there exists a polynomial relation R such that $L = \exists R$

Similarily we can define another time-complexity classes, e.g., languages from **NEXPTIME** are projections of relations such that:

- can be recognized in **P**
- there exists an exponential function f such that $v\$w \in R$ implies $|w| \leq f(|v|)$

What about space-complexity classes, e.g., **NL**?

A model with witnesses

Theorem

$L \in$ **NP** $\Leftrightarrow$ there exists a polynomial relation R such that $L = \exists R$

Similarily we can define another time-complexity classes, e.g., languages from **NEXPTIME** are projections of relations such that:

- ➔ can be recognized in **P**
- ➔ there exists an exponential function f such that $v\$w \in R$ implies $|w| \leq f(|v|)$

What about space-complexity classes, e.g., **NL**?

- a witness of logarithmic length?

A model with witnesses

Theorem

$L \in$ **NP** $\Leftrightarrow$ there exists a polynomial relation R such that $L = \exists R$

Similarily we can define another time-complexity classes, e.g., languages from **NEXPTIME** are projections of relations such that:

- can be recognized in **P**
- there exists an exponential function f such that $v\$w \in R$ implies $|w| \leq f(|v|)$

What about space-complexity classes, e.g., **NL**?

- a witness of logarithmic length? – too short
- a witness of polynomial length, recognizing in **L**?

A model with witnesses

Theorem

$L \in$ **NP** $\Leftrightarrow$ there exists a polynomial relation R such that $L = \exists R$

Similarily we can define another time-complexity classes, e.g., languages from **NEXPTIME** are projections of relations such that:

- ➔ can be recognized in **P**
- ➔ there exists an exponential function f such that $v\$w \in R$ implies $|w| \leq f(|v|)$

What about space-complexity classes, e.g., **NL**?

- a witness of logarithmic length? – too short
- a witness of polynomial length, recognizing in **L**?
  – too much: gives the whole **NP**
- a witness of polynomial length, which can be read only once (the head does not move left), recognizing in **L** – OK

Classes of complements

For every class C, the class **co**C consists of complements of languages from C.

- for trivial reasons, deterministic classes are equal to its co-classes, e.g., **P**=**coP**
- for nondeterministic classes this is not clear
- e.g., the language of graph, in which there DOES NOT exist an Hamiltonian cycle
  - ➔ belongs to **coNP**
  - ➔ but is it in **NP**? – what can be taken as a witness?

Classes of complements

For every class C, the class **co**C consists of complements of languages from C.

- for trivial reasons, deterministic classes are equal to its co-classes, e.g., **P**=**coP**
- for nondeterministic classes this is not clear
- e.g., the language of graph, in which there DOES NOT exist an Hamiltonian cycle
  - ➔ belongs to **coNP**
  - ➔ but is it in **NP**? – what can be taken as a witness?
- An open problem: does **NP**≠**coNP**?
  (if **NP**≠**coNP** then also **NP**≠**P**)
- Another open problem: does **NP**∩**coNP**=**P**?
  We don't have too many problems, for which we know that they are in **NP**∩**coNP**, but we do not know whether they are in **P.**

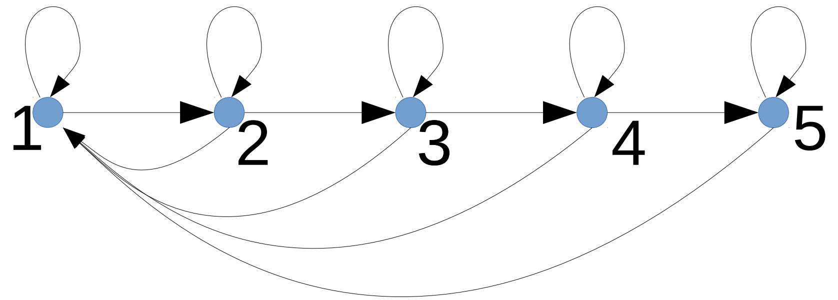
NP∩coNP

We don't have too many problems, for which we know that they are in **NP∩coNP**, but we do not know whether they are in **P:**

- For a long time checking that a number is prime was a problem with this property, but now we know that it is in **P**
- Example: factoring ∈ **NP∩coNP** (decision variant of factoring: does n have a prime factor $<k$?) – prime factorization is a witness in both directions.
  This suggests that **NP∩coNP≠P**, as we believe that factoring cannot be done in polynomial time.
- Another example: some game problems, e.g. parity_games ∈ **NP∩coNP** (next slide)

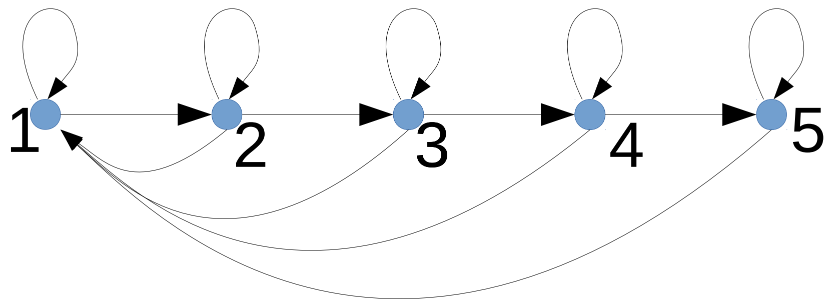
Parity games

- We are given a directed graph, with nodes labeled by numbers
- Players alternatingly move (one, common) pawn along edges of the graph – ad infinitum
- We look for the greatest number appearing infinitely often – if it is odd, then player 1 wins; if it is even, player 2 wins



Parity games

- We are given a directed graph, with nodes labeled by numbers
- Players alternatingly move (one, common) pawn along edges of the graph – ad infinitum
- We look for the greatest number appearing infinitely often – if it is odd, then player 1 wins; if it is even, player 2 wins
- Alternatively: we play only to the first repetition of a pair (node, player_number) and we look for the greatest number on the created cycle
- Question: does player 1 wins (has a wining strategy)?
- It is in **NP**: a strategy of player 1 is a polynomial size witness, which can be verified in polynomial time
- It is in **coNP** as well – a strategy of player 2 is ...
- not known to be in **P**
- can be solved in $O(n^{c+\log n})$



Determinization

<u>Theorem</u>

DTIME($f(n)$)$\subseteq$**NTIME**($f(n)$), **DSPACE**($f(n)$)$\subseteq$**NSPACE**($f(n)$)

<u>Proof</u>

Trivial, since a deterministic machine is a special case of a nondeterministic machine.

Determinization

<u>Theorem</u>

NTIME($f(n)$)$\subseteq$**DSPACE**($f(n)$)

<u>Proof</u>

- We have a nondetermin. machine M working in time $g(n)=O(f(n))$. We want to check whether it has an accepting run on a given input.

Determinization

Theorem

NTIME($f(n)$)$\subseteq$**DSPACE**($f(n)$)

Proof

- We have a nondetermin. machine M working in time $g(n)=O(f(n))$. We want to check whether it has an accepting run on a given input.
- Allocate space $g(n)$ and generate there all possible words w of this length, one after another (assume for a moment that $g(n)$ is space constructible)
- For every generated word w simulate M on the input word, treating w is a sequence of consecutive choices of M (the input word should not be destroyed)

Determinization

Theorem

NTIME($f(n)$)$\subseteq$**DSPACE**($f(n)$)

Proof

- We have a nondetermin. machine M working in time $g(n)=O(f(n))$. We want to check whether it has an accepting run on a given input.
- Allocate space $g(n)$ and generate there all possible words w of this length, one after another (assume for a moment that $g(n)$ is space constructible)
- For every generated word w simulate M on the input word, treating w is a sequence of consecutive choices of M (the input word should not be destroyed)
- We need space $g(n)$ for the sequences of choices, and at most $g(n)$ for the memory of M
- We can succeed also without assuming that $g(n)$ is space constructible: we start from short sequences of choices; if during the simulation of M we see that the sequence is too short, we make it longer.