Computational complexity

lecture 9

Alternating machines

Definition of an alternating Turing machine (ATM):

- a configuration can have multiple successors (as in NTM)
- additionally states of the machine (and in effect its configurations) are divided to existential and universal ones

The set of <u>wining</u> configurations is defined as the smallest set s.t.:

- accepting configurations are winning
- every <u>existential</u> configuration, whose <u>some</u> successor is winning, is also winning
- every <u>universal</u> configuration, whose <u>all</u> successors are winning, is also winning

We accept a word w, if the initial configuration for this word is winning.

M works in time $T(n)$ / in space $S(n)$, if every computation fits in this time / space.

Observation:
NTM is a special case of an ATM – only existential states

Alternating machines

Classes **ATIME**($T(n)$), **ASPACE**($S(n)$),

AP=$\cup_k$ **ATIME**(n^k), **AL**=**ASPACE**($log\ n$)

<u>Theorem</u>

AL=**P**, **AP**=**PSPACE** (the same can be said more generally)

Alternating machines

Classes **ATIME**($T(n)$), **ASPACE**($S(n)$),

AP=$\cup_k$ **ATIME**(n^k), **AL**=**ASPACE**($log\ n$)

Theorem

AL=**P**, **AP**=**PSPACE** (the same can be said more generally)

Proof **AP**⊆**PSPACE**

Backtracking: we browse through all computations of the alternating machine (such a computation can be represented in polynomial space)

Alternating machines

Classes **ATIME**($T(n)$), **ASPACE**($S(n)$),

AP$=\cup_k$ **ATIME**(n^k), **AL**=**ASPACE**($log\ n$)

<u>Theorem</u>

AL=**P**, **AP**=**PSPACE** (the same can be said more generally)

<u>Proof</u> **AL**$\subseteq$**P**

We construct the graph containing all reachable configurations of the alternating machine – it is of polynomial size. Then in polynomial time we can find all winning configurations, by going backwards (starting from accepting configurations).

Alternating machines

Classes **ATIME**($T(n)$), **ASPACE**($S(n)$),

AP=$\cup_k$ **ATIME**(n^k), **AL**=**ASPACE**($\log n$)

<u>Theorem</u>

AL=**P**, **AP**=**PSPACE** (the same can be said more generally)

<u>Proof</u> **PSPACE**⊆**AP**

It is enough to prove that QBF∈**AP**, as QBF is **PSPACE**-complete.
This is almost obvious – player ∃ chooses values of variables quantified existentially, and player ∀ chooses values of variables quantified universally; at the end we deterministically compute the value of the formula.
Actually: the algorithm for **AP** is simpler than for **PSPACE**.

Alternating machines

Classes **ATIME**($T(n)$), **ASPACE**($S(n)$),

AP$=\cup_k$ **ATIME**(n^k), **AL**=**ASPACE**($\log n$)

<u>Theorem</u>

AL=**P**, **AP**=**PSPACE** (the same can be said more generally)

<u>Proof</u> **P**⊆**AL**

- For an algorithm in **P** there is an equivalent boolean circuit, and we can construct it in logarithmic space.
- It is easy to give an algorithm in **AL**, which computes the value of a circuit: players walk from the output gate, in OR gates player ∃ decides which predecessor is true, and in AND gates player ∀ decides which predecessor is supposed to be false.
- We do not generate the whole circuit, only particular fragments, „on demand”.

Alternating machines

Consider alternating machines which:

- work in polynomial time
- the initial state is existential (universal)
- every computation leads to at most k-1 changes between existential and universal states

Fact

Such machines recognize languages from Σ^p_k (Π^p_k)

(we skip the formal proof, although it is easy)

Probabilistic machines

Machines with a source of random bits (probabilistic machines):

- a deterministic machine
- an additional read-once tape (the head cannot move left along this tape)

Probabilistic machines

Machines with a source of random bits (probabilistic machines):

- a deterministic machine
- an additional read-once tape (the head cannot move left along this tape)

Notice that **NP** can be defined as follows: a language L is in **NP** iff there is a polynomial $p(n)$ and a machine M with a source of random bits, working in at most $p(n)$ steps, and such that:

- $w \in L \Rightarrow \exists s.\ (w,s) \in L_M$
- $w \notin L \Rightarrow \nexists s.\ (w,s) \in L_M$

(a word is in L iff some witness confirms this)

Probabilistic machines

Machines with a source of random bits (probabilistic machines):

- a deterministic machine
- an additional read-once tape (the head cannot move left along this tape)

Notice that **NP** can be defined as follows: a language L is in **NP** iff there is a polynomial $p(n)$ and a machine M with a source of random bits, working in at most $p(n)$ steps, and such that:

- $w \in L \Rightarrow \exists s.\ (w,s) \in L_M$
- $w \notin L \Rightarrow \nexists s.\ (w,s) \in L_M$

Class **RP** (randomized polynomial time): as above, but

- $w \in L \Rightarrow Pr_s[(w,s) \in L_M] \geq 0.5$
- $w \notin L \Rightarrow \nexists s.\ (w,s) \in L_M$

Intuition: a word is in L, if at least half of possible witnesses confirm this.

Probabilistic machines

Class **RP** (randomized polynomial time): a language L is in **RP** iff there is a polynomial $p(n)$ and a machine M with a source of random bits, working in at most $p(n)$ steps, and such that:

- $w \in L \Rightarrow Pr_s[(w,s) \in L_M] \geq 0.5$
- $w \notin L \Rightarrow \nexists s.\ (w,s) \in L_M$

As s we can take sequences of length $p(n)$, or infinite sequences, does not matter.

Intuition: a word is in L, if at least half of possible witnesses confirm this (but there are no witnesses for words not in L)

In other words: if a word is not in L, we will certainly reject;
if it is in L, then choosing transitions randomly, we will accept with probability at least 0.5

Probabilistic machines

Class **RP** (randomized polynomial time): a language L is in **RP** iff there is a polynomial $p(n)$ and a machine M with a source of random bits, working in at most $p(n)$ steps, and such that:

- $w \in L \Rightarrow Pr_s[(w,s) \in L_M] \geq 0.5$
- $w \notin L \Rightarrow \nexists s.\ (w,s) \in L_M$

Remark: Some machines does not accept any language in the sense of **RP**. It is undecidable whether a machine is correct in the sense of **RP**, even if we know the polynomial $p(n)$

For this reason we do not know any **RP**-complete problem.
Intuition: we cannot reduce from every machine recognizing a language from **RP**, because we do not know how such machines look like.

Probabilistic machines

Class **RP** (randomized polynomial time): a language L is in **RP** iff there is a polynomial $p(n)$ and a machine M with a source of random bits, working in at most $p(n)$ steps, and such that:

- $w \in L \Rightarrow Pr_s[(w,s) \in L_M] \geq 0.5$
- $w \notin L \Rightarrow \nexists s.\ (w,s) \in L_M$

Fact: **P**$\subseteq$**RP**$\subseteq$**NP** (both inclusions are obvious)

Probabilistic machines

Class **RP** (randomized polynomial time): a language L is in **RP** iff there is a polynomial $T(n)$ and a machine M with a source of random bits, working in at most $T(n)$ steps, and such that:

- $w \in L \Rightarrow Pr_s[(w,s) \in L_M] \geq 1-p=0.5$
- $w \notin L \Rightarrow \nexists s.\ (w,s) \in L_M$

<u>Fact</u> (amplification): in the definition of **RP** the number 0.5 can be changed to any number from the interval (0,1), and the class of defined languages will remain the same

Proof: Let $\mathbf{RP}_p$ be the class with <u>error</u> probability p

Obviously $\mathbf{RP}_p \subseteq \mathbf{RP}_q$ when $p \leq q$

We will now prove that $\mathbf{RP}_p \subseteq \mathbf{RP}_{p^2}$

- Out of a machine M with error p we construct a machine M', which on the same input chooses randomly two witnesses, and accepts if some of them is a correct witness
- The running time doubles, so it remains polynomial
- The error probability decreases to p^2 – M' is wrong only when M made a mistake twice

Probabilistic machines

Is this a realistic model?

- It is more realistic than nondeterministic or alternating machines: we can run a probabilistic machine, give it some sequence of bits as random bits, and obtain a result that is correct with some probability.
- We obtain a result that is correct with some probability (and due to amplification this probability can be arbitrarily high), but we cannot be sure.
- How to generate bits that are really random? There exist physical random number generators (basing e.g. on quantum effects). Problems: they are relatively slow, and can be biased (in particular after some time, when they start to be broken).
- In practice, we use pseudo-random generators, that generate "random" bits using some algorithm. In practice, this works well, as the generated sequence looks like a random one.
  But theoretically, we cannot be sure about the probability of correctness.

Examples of randomized algorithms

An <u>example</u> of an algorithm in **coRP** – primality testing

input: n

question: is n prime?

[So the converse: “is n composite?” is in **RP**.]

Examples of randomized algorithms

An <u>example</u> of an algorithm in **coRP** – primality testing

input: n

question: is n prime?

[So the converse: “is n composite?” is in **RP**.]

History:

- Clearly primality$\in$**coNP**: a nontrivial divisor is a witness, but it is difficult to find it.
- For years it was not known how to check that a number is prime (even before the era of computers, this was an interesting problem)

Examples of randomized algorithms

An <u>example</u> of an algorithm in **coRP** – primality testing

input: n

question: is n prime?

[So the converse: “is n composite?” is in **RP**.]

History:

- Clearly primality∈**coNP**: a nontrivial divisor is a witness, but it is difficult to find it.
- For years it was not known how to check that a number is prime (even before the era of computers, this was an interesting problem)
- Pratt 1975, primality∈**NP** (i.e., ∈**NP**∩**coNP**) – certificate for primality that can be checked in polynomial time
- probabilistic tests discovered, showing primality∈**coRP** (Solovay-Strassen test 1977, Miller-Rabin test 1976-1980)

Examples of randomized algorithms

An <u>example</u> of an algorithm in **coRP** – primality testing

input: n

question: is n prime?

[So the converse: “is n composite?” is in **RP**.]

History:

- Clearly primality∈**coNP**: a nontrivial divisor is a witness, but it is difficult to find it.
- For years it was not known how to check that a number is prime (even before the era of computers, this was an interesting problem)
- Pratt 1975, primality∈**NP** (i.e., ∈**NP**∩**coNP**) – certificate for primality that can be checked in polynomial time
- probabilistic tests discovered, showing primality∈**coRP** (Solovay-Strassen test 1977, Miller-Rabin test 1976-1980)
- Adleman-Pomerance-Rumely 1983: determin. alg., $|n|^{O(\ln \ln |n|)}$
- Adleman-Huang 1992: primality∈**RP**∩**coRP**
- Agrawal-Kayal-Saxena 2002: primality∈**P**, best known time: $O(|n|^6)$
- in practice, probabilistic tests are used (the determ. alg. is too slow)

Examples of randomized algorithms

An <u>example</u> of an algorithm in **coRP** – primality testing

input: n

question: is n prime?

<u>Miller-Rabin test:</u>

- let $n\text{-}1=2^s \cdot d$
- choose randomly $n \in \{1,\ldots,n\text{-}1\}$
- If $a^d \not\equiv 1 \ (mod\ n)$ and $a^{2^r d} \not\equiv \text{-}1 \ (mod\ n)$ for all $r \in \{0,\ldots,s\text{-}1\}$, say "composite", otherwise say "prime"

Examples of randomized algorithms

An <u>example</u> of an algorithm in **coRP** – primality testing

input: n

question: is n prime?

<u>Miller-Rabin test:</u>

- let $n\text{-}1=2^s \cdot d$
- choose randomly $n \in \{1,...,n\text{-}1\}$
- If $a^d \not\equiv 1 \ (mod\ n)$ and $a^{2^r d} \not\equiv \text{-}1 \ (mod\ n)$ for all $r \in \{0,...,s\text{-}1\}$, say "composite", otherwise say "prime"

➔ For prime numbers, the algorithm always says "prime"

➔ For composite numbers, the algorithm says "composite" with probability $\geq 3/4$ (probability of error $\leq 1/4$)

➔ We skip a proof

Examples of randomized algorithms

An example of an algorithm in **RP**: check that a polynomial (given in an implicit form) is nonzero?

Formally, we are given an arithmetic circuit, where we have gates +, *, –, and input gates corresponding to variables – notice that such a circuit always encodes a polynomial with integer coefficients. We ask whether there is a valuation of variables for which the result of the circuit is nonzero.
The problem can be considered over rational (real) numbers, or over some finite field.

Examples of randomized algorithms

An example of an algorithm in **RP**: check that a polynomial (given in an implicit form) is nonzero?

Formally, we are given an arithmetic circuit, where we have gates +, *, –, and input gates corresponding to variables – notice that such a circuit always encodes a polynomial with integer coefficients. We ask whether there is a valuation of variables for which the result of the circuit is nonzero.
The problem can be considered over rational (real) numbers, or over some finite field.

In both cases, we do not know whether the problem is in **P**.

An idea for a probabilistic algorithm:
take a random valuation of variables, and check the result

Does it makes sense?
Yes, if every nonzero polynomial is nonzero for a majority of valuations of variables.

Examples of randomized algorithms (✶)

Lemma (Schwartz-Zippel)

Let p be a nonzero polynomial of k variables, of total degree d, over a field F (finite or not), let S be a finite subset of this field. We pick $r_1,\ldots,r_k \in S$ randomly.

Then $Pr[p(r_1,\ldots,r_k)=0] \leq d \,/\, |S|$

Examples of randomized algorithms (⋆)

Lemma (Schwartz-Zippel)

Let p be a nonzero polynomial of k variables, of total degree d, over a field F (finite or not), let S be a finite subset of this field. We pick $r_1,\ldots,r_k \in S$ randomly.

Then $Pr[p(r_1,\ldots,r_k)=0] \leq d / |S|$

Proof: induction over k

$k=1$ → Bezout's theorem: a polynomial of degree d has $\leq d$ zeroes

Examples of randomized algorithms (✶)

<u>Lemma</u> (Schwartz-Zippel)

Let p be a nonzero polynomial of k variables, of total degree d, over a field F (finite or not), let S be a finite subset of this field. We pick $r_1,\ldots,r_k \in S$ randomly.

Then $Pr[p(r_1,\ldots,r_k)=0] \le d / |S|$

<u>Proof</u>: induction over k

$k=1 \rightarrow$ Bezout's theorem: a polynomial of degree d has $\le d$ zeroes

Induction step: write p as a polynomial of the variable x_1:

$$p(x_1,\ldots,x_k)=\sum_{i=0}^{d} x_1^i \cdot p_i(x_2,\ldots,x_k)$$

Take the greatest i, for which p_i is nonzero (exists, since $p \neq 0$).

The degree of p_i is $\le d-i$. From the induction assumption:

$Pr[p_i(r_2,\ldots,r_k)=0] \le (d-i) / |S|$

Examples of randomized algorithms (✶)

<u>Lemma</u> (Schwartz-Zippel)

Let p be a nonzero polynomial of k variables, of total degree d, over a field F (finite or not), let S be a finite subset of this field. We pick $r_1,\dots,r_k \in S$ randomly.

Then $Pr[p(r_1,\dots,r_k)=0] \le d / |S|$

<u>Proof</u>: induction over k

$k=1 \rightarrow$ Bezout's theorem: a polynomial of degree d has $\le d$ zeroes

Induction step: write p as a polynomial of the variable x_1:

$$p(x_1,\dots,x_k)=\sum_{i=0}^{d} x_1^i \cdot p_i(x_2,\dots,x_k)$$

Take the greatest i, for which p_i is nonzero (exists, since $p \neq 0$).

The degree of p_i is $\le d-i$. From the induction assumption:

$Pr[p_i(r_2,\dots,r_k)=0] \le (d-i) / |S|$

If $p_i(r_2,\dots,r_k) \neq 0$, then $p(x_1,r_2,\dots,r_k)$ is of degree i , so

$Pr[p(r_1,\dots,r_k)=0 \mid p_i(r_2,\dots,r_k) \neq 0] \le i / |S|$

This is enough, since $Pr[A] \le Pr[B] + Pr[A|B^c]$ for arbitrary events A,B

Examples of randomized algorithms (⋆)

The Schwartz-Zippel lemma shows that the question whether a polynomial is nonzero is in **RP**, when we consider it over a large finite field.

- We see that a circuit with n gates defines a polynomial of total degree at most 2^n. If there are k variables, it is enough to pick k random numbers from $0,\ldots,10\cdot 2^n$ (it requires $O(kn)$ bits), compute the value of the polynomial (i.e., simulate the circuit), and accept if the result is nonzero. We are wrong with probability ≤ 0.1.

Examples of randomized algorithms (⋆)

The Schwartz-Zippel lemma shows that the question whether a polynomial is nonzero is in **RP**, when we consider it over a large finite field.

- We see that a circuit with n gates defines a polynomial of total degree at most 2^n. If there are k variables, it is enough to pick k random numbers from $0,\ldots,10 \cdot 2^n$ (it requires $O(kn)$ bits), compute the value of the polynomial (i.e., simulate the circuit), and accept if the result is nonzero. We are wrong with probability ≤ 0.1.

Over the field of rationals (or if the considered finite field is too large) there is an additional problem: how to evaluate the circuit in polynomial time? Even if the final result is 0, intermediate results can be very long.

- Solution: pick a random number m from $2,\ldots,2^{2n}$, and compute everything modulo m

  Why this works well?

Examples of randomized algorithms (⋆)

- We take random m from $2,\ldots,2^{2n}$, and we compute modulo m
- If the value of a polynomial $Y=p(r_1,\ldots,r_k)$ is 0, then modulo m it is 0 as well.
- If the value is nonzero, we will prove that with probability $\geq 1/(10n)$ it does not divide by m (i.e., is nonzero modulo m)

Examples of randomized algorithms (✶)

- We take random m from $2,\ldots,2^{2n}$, and we compute modulo m
- If the value of a polynomial $Y=p(r_1,\ldots,r_k)$ is 0, then modulo m it is 0 as well.
- If the value is nonzero, we will prove that with probability $\geq 1/(10n)$ it does not divide by m (i.e., is nonzero modulo m)
- The number $\pi(N)$ of prime numbers smaller than N satisfies:

$$\lim_{N\to\infty}\frac{\pi(N)}{N/\ln(N)}=1$$

- Thus m is prime with probability at least $1/(5n)$ (for large enough n)

Examples of randomized algorithms (⋆)

- We take random m from $2,\dots,2^{2n}$, and we compute modulo m
- If the value of a polynomial $Y=p(r_1,\dots,r_k)$ is 0, then modulo m it is 0 as well.
- If the value is nonzero, we will prove that with probability $\geq 1/(10n)$ it does not divide by m (i.e., is nonzero modulo m)
- The number $\pi(N)$ of prime numbers smaller than N satisfies:

$$\lim_{N\to\infty} \frac{\pi(N)}{N/ln(N)}=1$$

- Thus m is prime with probability at least $1/(5n)$ (for large enough n)
- We have $Y\leq(10\cdot 2^n)^{2^n}$
- The number of prime divisors of Y equals is at most logarithmic, i.e., $\leq 5n2^n$
- A randomly chosen m is among these divisors with probability $\leq 5n2^n/2^{2n}<1/(10n)$
- Thus m is prime and is NOT a divisor of Y with probability $>1/(10n)$

Examples of randomized algorithms (✶)

- We take random m from $2,...,2^{2n}$, and we compute modulo m
- If the value of a polynomial $Y=p(r_1,...,r_k)$ is 0, then modulo m it is 0 as well.
- If the value is nonzero, then with probability $\geq 1/(10n)$ it does not divide by m (i.e., is nonzero modulo m)
- This is still not enough – our algorithm fails with prob. $\leq 1\text{-}1/(10n)$
- Let us repeat the whole algorithm n times. This is enough, since

$$\lim_{n\to\infty} (1-1/n)^n = 1/e$$

- For large n this is <0.5
- We have finitely many „small” n, where the error is a constant; we can decrease it using the standard amplification