RANDOMIZED COMPUTATION

- Monte Carlo algorithms and random walks
- Randomized complexity classes: $\mathbf{RP}, \mathbf{ZPP}, \mathbf{PP}, \mathbf{BPP}$
- Perfect and slightly random sources
- Classes δ-$\mathbf{RP}$, δ-$\mathbf{BPP}$
- Circuit complexity

(C. Papadimitriou: *Computational complexity*, Chapter 11)

© 2007 TKK, Laboratory for Theoretical Computer Science

RANDOMIZED COMPUTATION

A randomized algorithm: flip unbiased coins

Example.

- Consider perfect matching:

  INSTANCE: Bipartite graph $B = (U, V, E)$, where $U = \{u_1, \ldots, u_n\}$, $V = \{v_1, \ldots, v_n\}$, $E \subseteq U \times V$.

  QUESTION: Is there a set $M \subseteq E$ of n edges such that for any two edges $(u, v), (u', v') \in M$, $u \neq u'$ and $v \neq v'$
  (is there a *perfect matching*)?

© 2007 TKK, Laboratory for Theoretical Computer Science

Perfect matching—cont'd

- This is related to computing determinants of a matrix:
  Given a graph G, construct an $n \times n$ matrix A_G where the element i, j is a variable x_{ij} iff $(u_i, v_j) \in E$ otherwise 0.

$$\det A^G = \sum_{\pi} \sigma(\pi) \prod_{i=1}^{n} A^g_{i,\pi(i)}$$

  where π ranges over permutations of n elements and each terms is of the form

$$\sigma(\pi) a_{1,\pi(1)} \cdots a_{n,\pi(n)}$$

- Hence, G has a perfect matching iff $\det A^G$ is not identically 0.
- Testing whether $\det A^G$ is identically 0 can be done using a randomized algorithm.

© 2007 TKK, Laboratory for Theoretical Computer Science

Randomized algorithm for perfect matching

Given a matrix $A^G(x_1, \ldots, x_m)$ with m variables:

  Choose m random integers $i_1, \ldots, i_m$ (between 0 and $M = 2m$)
  Compute $\det A^G(i_1, \ldots, i_m)$ (by Gaussian elimination).
  If $\det A^G(i_1, \ldots, i_m) \neq 0$, then return "$G$ has a perfect matching"
  If $\det A^G(i_1, \ldots, i_m) = 0$, then return "$G$ probably has no perfect matching"

Properties:

- No false positives (if "yes" is returned, this is correct).
- False negatives possible (if "no" is returned, this might be wrong).

© 2007 TKK, Laboratory for Theoretical Computer Science

Monte Carlo algorithm

- Polynomial randomized algorithm
- No false positives
- The probability of false negatives no more than $\frac{1}{2}$.

☞ The previous algorithm is a Monte Carlo algorithm for perfect matching: it can be shown that the probability of false negatives is no more than $\frac{1}{2}$ when the integers are randomly selected between 0 and $2m$.

☞ If the probability of false negatives is $\varepsilon > \frac{1}{2}$, we can perform k *independent* experiments and the probability of false negatives is reduced to ε^k (and running times remains polynomial).

© 2007 TKK, Laboratory for Theoretical Computer Science

Random Walks

- A randomized walk algorithm for SAT:

  Take any truth assignment T and repeat r times:

  If there is not unsatisfiable clauses, return "satisfiable"
  Otherwise take any unsatisfiable clause
  Pick any of its literals at random and flip it in T.

  After r repetition return "probably unsatisfiable"

- Is this a Monte Carlo algorithm?

  No false positives but the probability of false negatives is high! (An exponential number of repetitions r is needed to achieve low probability for classes of 3SAT problems).

© 2007 TKK, Laboratory for Theoretical Computer Science

Random Walks—cont'd

- For 2SAT a Monte Carlo algorithm is obtained by setting $r = 2n^2$.
- Then the probability of false negatives is less than $\frac{1}{2}$
- The following lemma plays an important role:

Lemma. If x is a random variable taking non-negative integer values, then for any $k > 0$, $\mathbf{prob}[x \geq k \cdot \mathcal{E}(x)] \leq \frac{1}{k}$

where $\mathcal{E}(x)$ is the expected value of x.

© 2007 TKK, Laboratory for Theoretical Computer Science

Monte Carlo algorithm for composite

- Fermat's Theorem: For a prime N, for all $0 < a < N$, $a^{N-1} = 1 \bmod N$.
- Fermat test for COMPOSITE:

  Pick random residue a modulo N.
  If $a^{N-1} \neq 1 \bmod N$, then return "N is composite"
  Otherwise answer "N is probably prime"

- Monte Carlo algorithm?
  - By Fermat's Theorem no false positive.
  - But is it the case that for a composite, for at least half of its nonzero residues a, $a^{N-1} \neq 1 \bmod N$?
    (No, Carmichael numbers are exceptions)

© 2007 TKK, Laboratory for Theoretical Computer Science

Monte Carlo algorithm for composite—cont'd

➤ A refined algorithm for testing compositeness of N

Generate a random integer M between 2 and $N-1$;
If $(M,N) > 1$ then return "N is a composite"
else
  if $(M|N) \neq M^{\frac{N-1}{2}}$ mod N then return "N is a composite"
  else return "N is probably a prime".

where (M,N) is the greatest common divisor of M and N and $(M|N)$ is the Jacobi symbol.

➤ This is a Monte Carlo algorithm:
(M,N) and $(M|N)$ can be computed in polynomial time, no false positives and the probability of false negative at most $\frac{1}{2}$.

© 2007 TKK, Laboratory for Theoretical Computer Science

RANDOMIZED COMPLEXITY CLASSES

➤ Randomized algorithms (such as Monte Carlo ones) can be analyzed using nondeterministic Turing machines but *with a different interpretation of what it means for such a machine to accept its input.*

➤ No coin-flipping is needed in the Turing machine!

© 2007 TKK, Laboratory for Theoretical Computer Science

The class RP

Definition. Let L be a language. A polynomial time *Monte Carlo Turing machine* for L is a nondeterministic Turing machine N
(i) which is precise having exactly two nondet. choices at each step;
(ii) the number of steps in each computation for an input of length n is $p(n)$, a polynomial and
(iii) for each input x:

- If $x \in L$, then at least half of the $2^{p(|x|)}$ computations of N on x halt with "yes".
- If $x \notin L$, then all the $2^{p(|x|)}$ computations halt with "no".

The class of all languages with polynomial time Monte Carlo Turing machines is denoted by **RP** (randomized polynomial time).

© 2007 TKK, Laboratory for Theoretical Computer Science

The class RP—cont'd

Monte Carlo algorithms are captured by **RP**:

➤ All nondeterministic steps are "coin flips".

➤ There are no false positive answers.

➤ All computations equiprobable (with probability $2^{-p(|x|)}$).

➤ The probability of a false negative is at most $\frac{1}{2}$:

- Given a Monte Carlo Turing machine N for a language L: a false negative answer is given if N halts with "no" on $x \in L$.
- This happens in less than half of the $2^{p(|x|)}$ computations each having a probability of $2^{-p(|x|)}$.
- Hence, the probability of a false negative is at most $\frac{1}{2} \cdot 2^{p(|x|)} \cdot 2^{-p(|x|)} = \frac{1}{2}$

© 2007 TKK, Laboratory for Theoretical Computer Science

The class RP—cont'd

The power of **RP** would not be affected if the probability of acceptance were not $\frac{1}{2}$ but any number $0 < \varepsilon < 1$:

- If $\varepsilon < \frac{1}{2}$, "repeat" the algorithm k times and accept iff at least one of the k computations accepts otherwise reject.
- Now the probability of false negative is at most $(1-\varepsilon)^k$.
- By taking $k = \lceil -\frac{1}{\log(1-\varepsilon)} \rceil$, the probability of false negative is at most $\frac{1}{2}$.
- The running time is k times the original.
- As $-\frac{1}{\log(1-\varepsilon)} \approx \frac{1}{\varepsilon}$, ε could even be of the form $\frac{1}{p(n)}$ where $p(n)$ is a polynomial and the overall algorithm would remain polynomial.

© 2007 TKK, Laboratory for Theoretical Computer Science

The class RP—cont'd

- $\mathbf{P} \subseteq \mathbf{RP} \subseteq \mathbf{NP}$
- Given a Turing machine, it is not easy to determine whether it is a Monte Carlo machine (for all inputs either rejects "unanimously" or accept "by majority").
  - ☞ A semantic class (like $\mathbf{NP} \cap \mathbf{coNP}$ and **TFNP**).
  - ☞ No known complete problem

For example, **P** and **NP** are syntactic classes with complete problems.

© 2007 TKK, Laboratory for Theoretical Computer Science

The class ZPP

- **coRP**: the languages having Monte Carlo machines with no false negatives and a limited number of false positives.
- PRIMES in **coRP**
- $\mathbf{ZPP} = \mathbf{RP} \cap \mathbf{coRP}$ is the class of languages with *Las Vegas algorithms* (polynomial randomized algorithms with zero probability of error).
- A Las Vegas algorithm = two Monte Carlo algorithms: one for the language and one for its complement.
- Running k independent experiments with both algorithms:
  (i) sooner or later a definite answer will come: either a positive answer from the algorithm with no false positives or a negative one from the algorithm with no false negatives.
  (ii) probability of a definite answer is at least $1-2^{-k}$.
- PRIMES in **RP** and thus in **ZPP**.

© 2007 TKK, Laboratory for Theoretical Computer Science

The class PP

- Consider the problem MAJSAT:
  Given a Boolean expression, is it true that the majority of the 2^n truth assignments to its n variables satisfy it.
- It is not clear that MAJSAT is in **NP** (and thus less likely in **RP**).
- **PP** is the class of languages L having a nondeterministic polynomially bounded Turing machine N (precise and with two choices each step) such that for all inputs x, $x \in L$ iff more than half of the computations of N on input x end up accepting.

Theorem. MAJSAT is **PP**-complete.

Theorem. $\mathbf{NP} \subseteq \mathbf{PP}$.

PP is closed under complement.

© 2007 TKK, Laboratory for Theoretical Computer Science

The class PP–cont'd

- $\mathbf{ZPP} \subseteq \mathbf{RP} \subseteq \mathbf{NP} \subseteq \mathbf{PP}$
- $\mathbf{ZPP}, \mathbf{RP}$ are plausible notions of efficient randomized computations (but $\mathbf{PP}$ is not).
- $\mathbf{PP}$ cannot be used algorithmically because acceptance by majority is too fragile: the acceptance probability can be $\frac{1}{2} + 2^{-p(|x|)}$ and there is no plausible efficient experimentation that can detect such accepting behaviour (see below).

© 2007 TKK, Laboratory for Theoretical Computer Science

Detecting the more likely side of a bias coin

- To understand this consider the following problem:
  You have a biased coin with one side having probability $\frac{1}{2} + \varepsilon$ and the other $\frac{1}{2} - \varepsilon$. How to detect which side is more likely?
  Solution: Flip the coin many times and pick the side that appeared the most times. But how many times?
- *The Chernoff bound:*
  Suppose that $x_1, \ldots, x_n$ are independent random variables taking the values 1 and 0 with probabilities p and $p-1$, respectively, and consider their sum $X = \sum_{i=1}^{n} x_i$. Then for all $0 \leq \theta \leq 1$,
  $\mathbf{prob}[X \geq (1+\theta)pn] \leq e^{-\frac{\theta^2}{3}pn}$.
- The probability that X deviates from its expected value (pn) decreases exponentially with the deviations.

© 2007 TKK, Laboratory for Theoretical Computer Science

Detecting the more likely side of a bias coin

- Corollary:
  If $p = \frac{1}{2} + \varepsilon$ for some $\varepsilon > 0$, then $\mathbf{prob}[\sum_{i=1}^{n} x_i \geq \frac{n}{2}] \leq e^{-\frac{\varepsilon^2}{6}n}$.
- A bias of ε can be detected with reasonable confidence by taking a majority of about $\frac{1}{\varepsilon^2}$ experiments ($e^{-\frac{\varepsilon^2}{6\varepsilon^2}} = 0.85$).
- For a $\mathbf{PP}$ problem the bias ε can be as small as $2^{-p(|x|)}$: an exponential number of repetitions of the algorithm is required to determine the correct answer with reasonable confidence.
- Is there some plausible notion of realistic computation between $\mathbf{RP}$ and $\mathbf{PP}$?
  ☞ $\mathbf{BPP}$

© 2007 TKK, Laboratory for Theoretical Computer Science

The class BPP

- $\mathbf{BPP}$ is the class of languages L having a nondeterministic polynomially bounded Turing machine N (precise and with two choices each step) such that for all inputs x,
  if $x \in L$, then at least $\frac{3}{4}$ of the computations of N on x accept;
  if $x \notin L$, then at least $\frac{3}{4}$ of the computations of N on x reject;
  (bounded probability of error)
- $\mathbf{RP} \subseteq \mathbf{BPP} \subseteq \mathbf{PP}$.
- Open: $\mathbf{BPP} \subseteq \mathbf{NP}$.
- $\mathbf{BPP}$ is closed under complement.
- Semantic class
- No known complete problem

© 2007 TKK, Laboratory for Theoretical Computer Science

RANDOM SOURCES

- In order to implement randomized algorithms (e.g., those for **RP** and **BPP**), we need a *source of random bits*.
- *A perfect random source* is a random variable with values that are infinite sequences $(x_1, x_2, \ldots)$ of bits such that for all $n > 0$ and for all $(y_1, y_2, \ldots, y_n) \in \{0,1\}^n$

$$\mathbf{prob}[x_i = y_i, i = 1, \ldots, n] = 2^{-n}$$

- A Monte Carlo algorithm could be implemented using a random source by generating a sequence $(x_1, x_2, \ldots)$ of bits and choosing the transition at each step i according to the bit x_i.

© 2007 TKK, Laboratory for Theoretical Computer Science

Random sources—cont'd

- A problem: where to find a perfect random source?
- A perfect random source should be
  - *independent*: the probability that $x_i = 1$ does not depend on the previous or future outcomes
  - *fair*: the probability that $x_i = 1$ should be exactly $\frac{1}{2}$.
- The important requirement is independence:

  Any independent but unfair random sequence of bits can be turned into a fair one as follows:

  (i) Break the sequence in pairs and
  (ii) interpret: $01 \rightsquigarrow 0$, $10 \rightsquigarrow 1$ (ignoring 00 and 11).

© 2007 TKK, Laboratory for Theoretical Computer Science

Random sources—cont'd

- Now if the probability of 1 is p, the probability of 10 is $p(1-p)$ which equals to that of 01.
- To get a perfect random sequence of length n we need a sequence of expected length $\frac{2n}{1-c}$ where is $c = p^2 + (1-p)^2$ is the coincidence probability of the source.
- The real problem of physically implementing perfect random sources is that any physical process tends to be affected by its previous outcomes (and circumstances leading to it).
- Randomness in mathematical or computational process: *pseudorandom number generators*

  Typical congruential approach ($x_{i+1} = ax_i + b \bmod c$) is terrible (easy to predict bits and even deduce "secret" parameters).

© 2007 TKK, Laboratory for Theoretical Computer Science

Slightly random sources

- Perfect random sources seem to be hard to implement physically.
- A weaker concept: *δ-random source*

  Let δ be a number $0 < \delta \leq \frac{1}{2}$ and p any function $\{0,1\}^* \mapsto [\delta, 1-\delta]$ (a highly complex function unknown to us).

  The δ-random source S_p is a random variable with infinite bit sequences as values where the probability that the first n bits have the values $y_1, y_2, \ldots, y_n$ is

$$\prod_{i=1}^{n} (y_i p(y_1 \ldots y_{i-1}) + (1-y_i)(1-p(y_1 \ldots y_{i-1})))$$

  (Notice: the probability that the ith bit is 1 is $p(y_1 \ldots y_{i-1})$, a number between δ and $1-\delta$ that depends in an arbitrary way on all previous outcomes $y_1 \ldots y_{i-1}$).

© 2007 TKK, Laboratory for Theoretical Computer Science

Slightly random sources—cont'd

- Now $\delta \leq p(y_1 \ldots y_{i-1}) \leq 1-\delta$
- A $\frac{1}{2}$-random source is a perfect random source.
- A δ-random source with $\delta < \frac{1}{2}$ is a *slightly random source*.
- Slightly random sources: Geiger counters, Zehner diodes, coins

© 2007 TKK, Laboratory for Theoretical Computer Science

Slightly random sources—cont'd

- In the worst case slightly random sources appear to be useless for running randomized algorithms.
- Suppose a Monte Carlo algorithm is driven by a random bits generated by a δ-random source with δ is much smaller that $\frac{1}{2}$.
- In the worst case the algorithm can make choices which very often lead to a false negative outcome:

  consider an *adversary* who knows the algorithm and monitors its executions including the random choices and sets the values of p on the basis of this.

© 2007 TKK, Laboratory for Theoretical Computer Science

The classes δ-RP and δ-BPP

- Let N be a precise, polynomially bounded nondeterministic Turing Machine with exactly two choices per step.
- On input x the computation $N(x)$ is in effect a full binary tree of depth $n = p(|x|)$ (having $2^{n+1}-1$ nodes of which 2^n are leaves and 2^n-1 internal).
- Let δ be a number $0 < \delta < \frac{1}{2}$. A *δ-assignment F* is a mapping from the set of edges of $N(x)$ to the interval $[\delta, 1-\delta]$ such that the two edges leaving each internal node are assigned numbers adding up to one (p is precisely F on 1-choices).

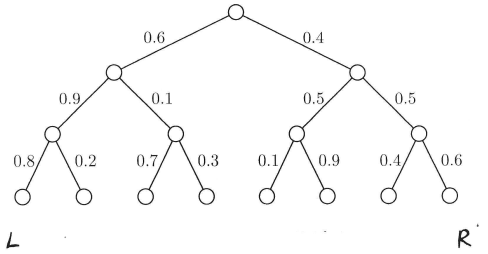
© 2007 TKK, Laboratory for Theoretical Computer Science

The classes δ-RP and δ-BPP—cont'd

- Given a δ-assignment F for each leaf l, $\mathbf{prob}[l] = \prod_{a \in P(l)} F(a)$ where $P(l)$ is the path from the root to leaf l.
- $\mathbf{prob}[N(x) = \text{“yes”} \,|\, F]$ is the sum of $\mathbf{prob}[l]$ for all “yes” leaves l of $N(x)$.
- We say that a language L is in **δ-RP** if there is a nondeterministic machine N, standardized as above, such that if $x \in L$, then $\mathbf{prob}[N(x) = \text{“yes”} \,|\, F] \geq \frac{1}{2}$ and if $x \notin L$, then $\mathbf{prob}[M(x) = \text{“yes”} \,|\, F] = 0$ for *all δ-assignments F*.
- A language L is in **δ-BPP** if there is a nondeterministic machine N such that if $x \in L$, then $\mathbf{prob}[N(x) = \text{“yes”} \,|\, F] \geq \frac{3}{4}$ and if $x \notin L$, then $\mathbf{prob}[N(x) = \text{“no”} \,|\, F] \geq \frac{3}{4}$ for *all δ-assignments F*.

© 2007 TKK, Laboratory for Theoretical Computer Science

Example. Consider the computation tree and 0.1-assignment [Papadimitriou, 1994]



- For the left most leaf L
  $\textbf{prob}[L] = \prod_{a \in P(L)} F(a) = 0.6 \cdot 0.9 \cdot 0.8 = 0.432$
- and for the right most leaf R
  $\textbf{prob}[R] = \prod_{a \in P(R)} F(a) = 0.4 \cdot 0.5 \cdot 0.6 = 0.120$

© 2007 TKK, Laboratory for Theoretical Computer Science

The classes δ-RP and δ-BPP—cont'd

- 0-$\mathbf{RP} = 0$-$\mathbf{BPP} = \mathbf{P}$
- $\frac{1}{2}$-$\mathbf{RP} = \mathbf{RP}$, $\frac{1}{2}$-$\mathbf{BPP} = \mathbf{BPP}$
- What about intermediate values of δ?
- A slightly random source can be used to simulate any randomized algorithm with polynomial loss of efficiency.
- This holds even if we assume that a hypothetical "adversary" can bias the slightly random source in arbitrary ways to lead the random algorithm to false answers.

Theorem. For any $\delta > 0$, δ-$\mathbf{BPP} = \mathbf{BPP}$.

Proof. δ-$\mathbf{BPP} \subseteq \mathbf{BPP}$ clear; $\mathbf{BPP} \subseteq \delta$-$\mathbf{BPP}$ tricky (see below).

© 2007 TKK, Laboratory for Theoretical Computer Science

Simulating a randomized algorithm

- Assume that $L \in \mathbf{BPP}$, i.e., L is decided by a NTM N by clear majority.
- Construct a machine N' deciding L by clear majority when driven by *any* slightly random source.
- The basic idea: confuse the "adversary" by shattering the slightly random bits using inner products.
- Inner product of two sequences of bits $\kappa = (\kappa_1, \ldots, \kappa_k)$ and $\lambda = (\lambda_1, \ldots, \lambda_k)$ is the bit obtained by $\kappa \cdot \lambda = \sum_{i=1}^{k} \kappa_i \lambda_i \mod 2$.

© 2007 TKK, Laboratory for Theoretical Computer Science

Simulating machine N'

- On input x, let $n = p(|x|)$ be the length of N's computation on x and let k be an integer (a parameter depending on n and δ).
- Generate n sequences of bits (blocks) $\beta_1, \ldots, \beta_n$ using a δ-random source where each β_i contains k bits.
- Do 2^k parallel simulations of N with the sequences of choices

$$\begin{aligned} T &= \{(\beta_1 \cdot \kappa, \ldots, \beta_n \cdot \kappa) : \kappa = 0, 1, \ldots, 2^k - 1\} \\ &= \{(\beta_1 \cdot 0, \ldots, \beta_n \cdot 0), \ldots, (\beta_1 \cdot (2^k - 1), \ldots, \beta_n \cdot (2^k - 1))\} \end{aligned}$$

- Of the 2^k answers, adopt the majoritanian one as the answer of N'.

© 2007 TKK, Laboratory for Theoretical Computer Science

Simulating machine N' — cont'd

To reduce the probability of a false answer by N' to at most $1/4$:

- Assume that probability of wrong answer by N is $1/32$ (instead of $1/4$).
- Set $k = \lceil \frac{\log n + 5}{2\delta - 2\delta^2} \rceil$
- Now N' works within time $O(n2^k) = O(nn^{\frac{1}{2\delta-2\delta^2}}) = O(p(|x|)^{1+\frac{1}{2\delta-2\delta^2}})$, i.e. in polynomial time.

Corollary. For any $\delta > 0$, δ-**RP** $=$ **RP**.

© 2007 TKK, Laboratory for Theoretical Computer Science

CIRCUIT COMPLEXITY

- A Boolean circuit with n inputs accepts a string $x = x_1 \ldots x_n$ of length n in $\{0,1\}^*$ iff the output of the circuit is **true** given x as its input (i.e., input i is **true** if x_i is 1 and **false** otherwise).
- To relate circuits to strings of arbitrary length families of circuits are introduced.
- The size of a circuit is the number of gates in it.
- A family of circuits is an infinite sequence $C = (C_0, C_1, \ldots)$ of Boolean circuits where C_n has n input variables.

© 2007 TKK, Laboratory for Theoretical Computer Science

Languages with polynomial circuits

- A language $L \subseteq \{0,1\}^*$ has *polynomial circuits* if there is a family $C = (C_0, C_1, \ldots)$ such that

  (i) the size of C_n is at most $p(n)$ for some fixed polynomial p;

  (ii) for all $x \in \{0,1\}^*$, $x \in L$ iff the output of $C_{|x|}$ is **true** when x is given as input to $C_{|x|}$.

Proposition. All languages in **P** have polynomial circuits.

Proof. By the **P**-completeness proof of CIRCUIT VALUE:
Given a Turing machine M, its input x, and running time $p(|x|)$, a variable-free polynomial size circuit $C(M,x,p)$ is constructed such that the output of $C(M,x,p)$ is **true** iff M accepts x.

It is easy to modify input gates of $C(M,x,p)$ to variable gates reflecting the symbols in x.

© 2007 TKK, Laboratory for Theoretical Computer Science

Languages with polynomial circuits

Proposition. There are undecidable languages that have polynomial circuits.

Proof.

- Let $L \subseteq \{0,1\}^*$ be an undecidable language and let $U \subseteq \{1\}^*$ be $U = \{1^n :$ the binary expansion of n is in $L\}$.
- U is undecidable.
- U has polynomial circuits $(C_0, C_1, \ldots)$ where C_n consists of n input gates and
  - n-1 AND-gates (conjunction of all inputs) if $1^n \in U$ and
  - **false** output gate if $1^n \notin U$.

© 2007 TKK, Laboratory for Theoretical Computer Science

Uniformly polynomial circuits

- A family $C = (C_0, C_1, \ldots)$ of circuits is said to be *uniform* if there is a $\log n$-space bounded Turing machine which on input 1^n outputs C_n.
- A language $L \subseteq \{0,1\}^*$ has *uniformly polynomial circuits* if there is a uniform family of polynomial circuits $C = (C_0, C_1, \ldots)$ that decides L.

Theorem. A language L has uniformly polynomial circuits iff $L \in \mathbf{P}$.

Proof. $(\Rightarrow)$ $x \in L$ can be decided in polynomial time by constructing $C_{|x|}$ in $\log |x|$ space (and hence in polynomial time) and evaluating it for input x (in polynomial time).
$(\Leftarrow)$ By the $\mathbf{P}$-completeness proof of CIRCUIT VALUE:
Given a Turing machine M, its input x, and running time $p(|x|)$, the circuit $C(M, x, p)$ can be constructed in $\log |x|$ space.

© 2007 TKK, Laboratory for Theoretical Computer Science

Polynomial circuits and P vs NP

- $\mathbf{P} \neq \mathbf{NP}$ equivalent to
  Conjecture A: $\mathbf{NP}$-complete problems have no uniformly polynomial circuits.
- **Conjecture B:** $\mathbf{NP}$-complete problems have no polynomial circuits, uniform or not.
- Most Boolean functions do not have small circuits.
- An approach to establish $\mathbf{P} \neq \mathbf{NP}$: show Conjecture B for some $\mathbf{NP}$-complete problem.
- Cannot be used for establishing $\mathbf{P} \neq \mathbf{BPP}$

Theorem. All languages in $\mathbf{BPP}$ have polynomial circuits.

© 2007 TKK, Laboratory for Theoretical Computer Science

Learning Objectives

- The concepts of Monte Carlo and Las Vegas algorithms
- The key randomized complexity classes: $\mathbf{RP}, \mathbf{ZPP}, \mathbf{PP}, \mathbf{BPP}$
- The concepts of perfect and slightly random sources
- Basic concepts of circuit complexity: languages with polynomial circuits and uniformly polynomial circuits.

© 2007 TKK, Laboratory for Theoretical Computer Science