Layerwise computable mappings and computable Lovasz local lemma

following Lovasz, Moser, Tardos, Hoyrup, Rojas, Levin, Fortnow, Miller, K. Makarychev, Rumyantsev,...

Philosophy

Philosophy

- Probabilistic existence proofs: we show that some property is true for a random object with positive probability, and conclude that objects with this property do exist. Randomized algorithms, exhaustive search.

Philosophy

- Probabilistic existence proofs: we show that some property is true for a random object with positive probability, and conclude that objects with this property do exist. Randomized algorithms, exhaustive search.
- Constructive proofs: explicit construction, (fast) algorithms,...

Probabilistic proof: uniform matrices

Probabilistic proof: uniform matrices

- $0/1$ $n \times n$ matrices

Probabilistic proof: uniform matrices

- 0/1 $n \times n$ matrices
- $k \times k$ minors: k rows and k columns selected

Probabilistic proof: uniform matrices

- 0/1 $n \times n$ matrices
- $k \times k$ minors: k rows and k columns selected
- uniform minor: all zeros or all ones

Probabilistic proof: uniform matrices

- 0/1 $n \times n$ matrices
- $k \times k$ minors: k rows and k columns selected
- uniform minor: all zeros or all ones
- for $k = O(\log n)$ there exists $n \times n$ matrix without uniform $k \times k$ minors

Probabilistic proof: uniform matrices

- $0/1$ $n \times n$ matrices
- $k \times k$ minors: k rows and k columns selected
- uniform minor: all zeros or all ones
- for $k = O(\log n)$ there exists $n \times n$ matrix without uniform $k \times k$ minors
- Why? Matrices with uniform minors are compressible, so they appear with small probability.

Probabilistic proof: max-cut

Probabilistic proof: max-cut

- In a graph with E edges one can color vertices in two colors obtaining at least $E/2$ bicolored edges.

Probabilistic proof: max-cut

- In a graph with E edges one can color vertices in two colors obtaining at least $E/2$ bicolored edges.
- Proof: expected number of bicolored edges is $E/2$ (linearity of expectation)

Probabilistic proof: at least $7/8$ satisfied clauses in 3-CNF

Probabilistic proof: at least 7/8 satisfied clauses in 3-CNF

- $(\neg p \vee q \vee r) \wedge (p \vee \neg r \vee \neg s) \wedge \ldots$

Probabilistic proof: at least $7/8$ satisfied clauses in 3-CNF

- $(\neg p \vee q \vee r) \wedge (p \vee \neg r \vee \neg s) \wedge \ldots$
- each clause has exactly 3 literals

Probabilistic proof: at least $7/8$ satisfied clauses in 3-CNF

- $(\neg p \vee q \vee r) \wedge (p \vee \neg r \vee \neg s) \wedge \ldots$
- each clause has exactly 3 literals
- For each 3-CNF there is an assignment that satisfies at least $7/8$ of the clauses

Derandomization

Derandomization

- How to convert probabilistic proof into an explicit construction?

Derandomization

- How to convert probabilistic proof into an explicit construction?
- Conditional expectations: fix sequentially the values of the variables so that conditional expectation does not decrease, until all the variables are fixed

Derandomization

- How to convert probabilistic proof into an explicit construction?
- Conditional expectations: fix sequentially the values of the variables so that conditional expectation does not decrease, until all the variables are fixed
  (possible if we can compute the conditional expectation)

Derandomization

- How to convert probabilistic proof into an explicit construction?
- Conditional expectations: fix sequentially the values of the variables so that conditional expectation does not decrease, until all the variables are fixed
  (possible if we can compute the conditional expectation)
- Big machinery: pseudo-randomness, expanders, extractors,...

Infinite case

Infinite case

- Random process (a machine with random bit generator)

Infinite case

- Random process (a machine with random bit generator)
- generates a sequence of output bits

Infinite case

- Random process (a machine with random bit generator)
- generates a sequence of output bits
- we prove that the probability to get a “good” (infinite) sequence is positive

Infinite case

- Random process (a machine with random bit generator)
- generates a sequence of output bits
- we prove that the probability to get a “good” (infinite) sequence is positive
- conclusion: good sequences exist

Infinite case

- Random process (a machine with random bit generator)
- generates a sequence of output bits
- we prove that the probability to get a “good” (infinite) sequence is positive
- conclusion: good sequences exist
- “Derandomization”: can we prove that *computable* good sequence exist?

Two simple derandomization tools

Two simple derandomization tools

- (Singleton) Let ω be a bit sequence. If the probability to get ω by a randomized algorithm is *positive*, then ω is computable.

Two simple derandomization tools

- (Singleton) Let ω be a bit sequence. If the probability to get ω by a randomized algorithm is *positive*, then ω is computable.
- (Closed set) Let S be a *closed* set in the Cantor space. If a randomized algorithm produces an element in S *with probability* 1, then A has a computable element.

Two simple derandomization tools

- (Singleton) Let ω be a bit sequence. If the probability to get ω by a randomized algorithm is *positive*, then ω is computable.
- (Closed set) Let S be a *closed* set in the Cantor space. If a randomized algorithm produces an element in S *with probability* 1, then A has a computable element.

First seem to be useless; the second will be used, but more general class of randomized algorithms is needed

Randomized algorithm and its output distribution

Randomized algorithm and its output distribution

- Machine M has access to fair coin

Randomized algorithm and its output distribution

- Machine M has access to fair coin
- has write-only output tape filled bit by bit

Randomized algorithm and its output distribution

- Machine M has access to fair coin
- has write-only output tape filled bit by bit
- output sequence can be finite or infinite

Randomized algorithm and its output distribution

- Machine M has access to fair coin
- has write-only output tape filled bit by bit
- output sequence can be finite or infinite
- we are interested in infinite sequences, but the probability to get an infinite sequence may be < 1

Randomized algorithm and its output distribution

- Machine M has access to fair coin
- has write-only output tape filled bit by bit
- output sequence can be finite or infinite
- we are interested in infinite sequences, but the probability to get an infinite sequence may be < 1
- function $m(x) =$ probability to get x or some extension

Randomized algorithm and its output distribution

- Machine M has access to fair coin
- has write-only output tape filled bit by bit
- output sequence can be finite or infinite
- we are interested in infinite sequences, but the probability to get an infinite sequence may be < 1
- function $m(x) =$ probability to get x or some extension
- $m(x)$ is lower semicomputable

Randomized algorithm and its output distribution

- Machine M has access to fair coin
- has write-only output tape filled bit by bit
- output sequence can be finite or infinite
- we are interested in infinite sequences, but the probability to get an infinite sequence may be < 1
- function $m(x) =$ probability to get x or some extension
- $m(x)$ is lower semicomputable
- $m(\Lambda) = 1$

Randomized algorithm and its output distribution

- Machine M has access to fair coin
- has write-only output tape filled bit by bit
- output sequence can be finite or infinite
- we are interested in infinite sequences, but the probability to get an infinite sequence may be < 1
- function $m(x) =$ probability to get x or some extension
- $m(x)$ is lower semicomputable
- $m(\Lambda) = 1$
- $m(x) \geq m(x0) + m(x1)$ for all binary strings x

Randomized algorithm and its output distribution

- Machine M has access to fair coin
- has write-only output tape filled bit by bit
- output sequence can be finite or infinite
- we are interested in infinite sequences, but the probability to get an infinite sequence may be < 1
- function $m(x) =$ probability to get x or some extension
- $m(x)$ is lower semicomputable
- $m(\Lambda) = 1$
- $m(x) \geq m(x0) + m(x1)$ for all binary strings x
- every m with these properties corresponds to some M

Randomized algorithm and its output distribution

- Machine M has access to fair coin
- has write-only output tape filled bit by bit
- output sequence can be finite or infinite
- we are interested in infinite sequences, but the probability to get an infinite sequence may be < 1
- function $m(x) =$ probability to get x or some extension
- $m(x)$ is lower semicomputable
- $m(\Lambda) = 1$
- $m(x) \geq m(x0) + m(x1)$ for all binary strings x
- every m with these properties corresponds to some M
- measures $m(x) = m(x0) + m(x1)$ correspond to machines that generate infinite sequences almost surely

Existence of computable objects

Existence of computable objects

(de Leeuw, Moore, Shannon, Shapiro): if a single sequence is generated by some randomized algorithm with positive probability, it is computable

Existence of computable objects

(de Leeuw, Moore, Shannon, Shapiro): if a single sequence is generated by some randomized algorithm with positive probability, it is computable

Proof:

Existence of computable objects

(de Leeuw, Moore, Shannon, Shapiro): if a single sequence is generated by some randomized algorithm with positive probability, it is computable

Proof:

- assume that probability of $\{\omega\}$ is greater than some $\varepsilon > 0$

Existence of computable objects

(de Leeuw, Moore, Shannon, Shapiro): if a single sequence is generated by some randomized algorithm with positive probability, it is computable

Proof:

- assume that probability of $\{\omega\}$ is greater than some $\varepsilon > 0$
- consider maximal set of incomparable strings x such that $m(x) > \varepsilon$

Existence of computable objects

(de Leeuw, Moore, Shannon, Shapiro): if a single sequence is generated by some randomized algorithm with positive probability, it is computable

Proof:

- assume that probability of $\{\omega\}$ is greater than some $\varepsilon > 0$
- consider maximal set of incomparable strings x such that $m(x) > \varepsilon$
- each element of this set can be extended uniquely (or cannot be extended at all)

Existence of computable objects

(de Leeuw, Moore, Shannon, Shapiro): if a single sequence is generated by some randomized algorithm with positive probability, it is computable

Proof:

- assume that probability of $\{\omega\}$ is greater than some $\varepsilon > 0$
- consider maximal set of incomparable strings x such that $m(x) > \varepsilon$
- each element of this set can be extended uniquely (or cannot be extended at all)
- ω can be reconstructed starting from its prefix in the set

Existence of computable objects

(de Leeuw, Moore, Shannon, Shapiro): if a single sequence is generated by some randomized algorithm with positive probability, it is computable

Proof:

- assume that probability of $\{\omega\}$ is greater than some $\varepsilon > 0$
- consider maximal set of incomparable strings x such that $m(x) > \varepsilon$
- each element of this set can be extended uniquely (or cannot be extended at all)
- ω can be reconstructed starting from its prefix in the set

Probably not very useful in proving the existence of computable objects

Existence of computable objects II

Existence of computable objects II

- closed set in the Cantor space

Existence of computable objects II

- closed set in the Cantor space
- = defined by a family of conditions, each dealing with finitely many bits

Existence of computable objects II

- closed set in the Cantor space
- = defined by a family of conditions, each dealing with finitely many bits
- example: square-free

Existence of computable objects II

- closed set in the Cantor space
- = defined by a family of conditions, each dealing with finitely many bits
- example: square-free
- If some randomized machine M with probability 1 generates a sequence in some closed set S, then S contains a computable element

Existence of computable objects II

- closed set in the Cantor space
- = defined by a family of conditions, each dealing with finitely many bits
- example: square-free
- If some randomized machine M with probability 1 generates a sequence in some closed set S, then S contains a computable element
- proof: construct ω bit by bit in such a way that each prefix of ω has positive probability

Existence of computable objects II

- closed set in the Cantor space
- = defined by a family of conditions, each dealing with finitely many bits
- example: square-free
- If some randomized machine M with probability 1 generates a sequence in some closed set S, then S contains a computable element
- proof: construct ω bit by bit in such a way that each prefix of ω has positive probability

This will be used but some more general machines are needed

Lovasz local lemma (special case)

Lovasz local lemma (special case)

- CNF: $(a \vee \neg b \vee c) \wedge (\neg a \vee d \vee \neg e) \wedge \dots$

Lovasz local lemma (special case)

- CNF: $(a \vee \neg b \vee c) \wedge (\neg a \vee d \vee \neg e) \wedge \ldots$
- each clause excludes some combination of variables appearing in it

Lovasz local lemma (special case)

- CNF: $(a \vee \neg b \vee c) \wedge (\neg a \vee d \vee \neg e) \wedge \ldots$
- each clause excludes some combination of variables appearing in it
- assume each clause has exactly m variables

Lovasz local lemma (special case)

- CNF: $(a \vee \neg b \vee c) \wedge (\neg a \vee d \vee \neg e) \wedge \dots$
- each clause excludes some combination of variables appearing in it
- assume each clause has exactly m variables
- if there are less than 2^m clauses then CNF is satisfiable

Lovasz local lemma (special case)

- CNF: $(a \vee \neg b \vee c) \wedge (\neg a \vee d \vee \neg e) \wedge \dots$
- each clause excludes some combination of variables appearing in it
- assume each clause has exactly m variables
- if there are less than 2^m clauses then CNF is satisfiable
- LLL: if each clause has at most 2^{m-3} neighbors, then CNF is satisfiable

Lovasz local lemma (special case)

- CNF: $(a \vee \neg b \vee c) \wedge (\neg a \vee d \vee \neg e) \wedge \ldots$
- each clause excludes some combination of variables appearing in it
- assume each clause has exactly m variables
- if there are less than 2^m clauses then CNF is satisfiable
- LLL: if each clause has at most 2^{m-3} neighbors, then CNF is satisfiable
- neighbors: clauses that have common variables

Lovasz local lemma (special case)

- CNF: $(a \vee \neg b \vee c) \wedge (\neg a \vee d \vee \neg e) \wedge \ldots$
- each clause excludes some combination of variables appearing in it
- assume each clause has exactly m variables
- if there are less than 2^m clauses then CNF is satisfiable
- LLL: if each clause has at most 2^{m-3} neighbors, then CNF is satisfiable
- neighbors: clauses that have common variables
- compactness: finite case is enough

Lovasz local lemma (special case)

- CNF: $(a \vee \neg b \vee c) \wedge (\neg a \vee d \vee \neg e) \wedge \ldots$
- each clause excludes some combination of variables appearing in it
- assume each clause has exactly m variables
- if there are less than 2^m clauses then CNF is satisfiable
- LLL: if each clause has at most 2^{m-3} neighbors, then CNF is satisfiable
- neighbors: clauses that have common variables
- compactness: finite case is enough
- classical proof uses induction to prove some bound on conditional probabilities

Lovasz local lemma (special case)

- CNF: $(a \vee \neg b \vee c) \wedge (\neg a \vee d \vee \neg e) \wedge \ldots$
- each clause excludes some combination of variables appearing in it
- assume each clause has exactly m variables
- if there are less than 2^m clauses then CNF is satisfiable
- LLL: if each clause has at most 2^{m-3} neighbors, then CNF is satisfiable
- neighbors: clauses that have common variables
- compactness: finite case is enough
- classical proof uses induction to prove some bound on conditional probabilities
- Moser's proof that uses Kolmogorov complexity

Infinite Lovasz local lemma

Infinite Lovasz local lemma

- countably many variables

Infinite Lovasz local lemma

- countably many variables
- each clause involves m of them

Infinite Lovasz local lemma

- countably many variables
- each clause involves m of them
- and has at most 2^{m-3} neighbors

Infinite Lovasz local lemma

- countably many variables
- each clause involves *m* of them
- and has at most 2^{m-3} neighbors
- computable CNF: variables and clauses are indexed by integers

Infinite Lovasz local lemma

- countably many variables
- each clause involves *m* of them
- and has at most 2^{m-3} neighbors
- computable CNF: variables and clauses are indexed by integers
- algorithm writes down *i*-th clause given *i*

Infinite Lovasz local lemma

- countably many variables
- each clause involves m of them
- and has at most 2^{m-3} neighbors
- computable CNF: variables and clauses are indexed by integers
- algorithm writes down i-th clause given i
- and lists all clauses that involve j-th variable given j

Infinite Lovasz local lemma

- countably many variables
- each clause involves m of them
- and has at most 2^{m-3} neighbors
- computable CNF: variables and clauses are indexed by integers
- algorithm writes down i-th clause given i
- and lists all clauses that involve j-th variable given j
- Computable LLL: such a CNF has a computable satisfying assignment

Infinite Lovasz local lemma

- countably many variables
- each clause involves *m* of them
- and has at most 2^{m-3} neighbors
- computable CNF: variables and clauses are indexed by integers
- algorithm writes down *i*-th clause given *i*
- and lists all clauses that involve *j*-th variable given *j*
- Computable LLL: such a CNF has a computable satisfying assignment

Proof: CNF determines a closed set;

Infinite Lovasz local lemma

- countably many variables
- each clause involves *m* of them
- and has at most 2^{m-3} neighbors
- computable CNF: variables and clauses are indexed by integers
- algorithm writes down *i*-th clause given *i*
- and lists all clauses that involve *j*-th variable given *j*
- Computable LLL: such a CNF has a computable satisfying assignment

Proof: CNF determines a closed set; it is enough to construct a machine that generates satisfying assignments with probability 1;

Infinite Lovasz local lemma

- countably many variables
- each clause involves *m* of them
- and has at most 2^{m-3} neighbors
- computable CNF: variables and clauses are indexed by integers
- algorithm writes down *i*-th clause given *i*
- and lists all clauses that involve *j*-th variable given *j*
- Computable LLL: such a CNF has a computable satisfying assignment

Proof: CNF determines a closed set; it is enough to construct a machine that generates satisfying assignments with probability 1; such a machine can be extracted from Moser–Tardos algorithm for finding a solution for finite LLL;

Infinite Lovasz local lemma

- countably many variables
- each clause involves *m* of them
- and has at most 2^{m-3} neighbors
- computable CNF: variables and clauses are indexed by integers
- algorithm writes down *i*-th clause given *i*
- and lists all clauses that involve *j*-th variable given *j*
- Computable LLL: such a CNF has a computable satisfying assignment

Proof: CNF determines a closed set; it is enough to construct a machine that generates satisfying assignments with probability 1; such a machine can be extracted from Moser–Tardos algorithm for finding a solution for finite LLL; but this is *rewriting* machine

Rewriting machines

Rewriting machines

- Machine has a random bit generator and rewritable output tape

Rewriting machines

- Machine has a random bit generator and rewritable output tape
- restriction: each output bit stabilizes (to 0 or to 1) with probability 1

Rewriting machines

- Machine has a random bit generator and rewritable output tape
- restriction: each output bit stabilizes (to 0 or to 1) with probability 1
- Defines an almost everywhere defined mapping

Rewriting machines

- Machine has a random bit generator and rewritable output tape
- restriction: each output bit stabilizes (to 0 or to 1) with probability 1
- Defines an almost everywhere defined mapping
- stronger condition: for each bit position i and every $\varepsilon > 0$ we can compute $N(i, \varepsilon)$ such that change in i-th bit after $N(i, \varepsilon)$ steps has probability less than ε

Rewriting machines

- Machine has a random bit generator and rewritable output tape
- restriction: each output bit stabilizes (to 0 or to 1) with probability 1
- Defines an almost everywhere defined mapping
- stronger condition: for each bit position i and every $\varepsilon > 0$ we can compute $N(i, \varepsilon)$ such that change in i-th bit after $N(i, \varepsilon)$ steps has probability less than ε
- mappings defined in this way are layerwise computable

Rewriting machines

- Machine has a random bit generator and rewritable output tape
- restriction: each output bit stabilizes (to 0 or to 1) with probability 1
- Defines an almost everywhere defined mapping
- stronger condition: for each bit position i and every $\varepsilon > 0$ we can compute $N(i, \varepsilon)$ such that change in i-th bit after $N(i, \varepsilon)$ steps has probability less than ε
- mappings defined in this way are layerwise computable
- output distribution is still computable: $m(x)$ = the probability that output starts with x, can be computed with arbitrary precision

Rewriting machines

- Machine has a random bit generator and rewritable output tape
- restriction: each output bit stabilizes (to 0 or to 1) with probability 1
- Defines an almost everywhere defined mapping
- stronger condition: for each bit position i and every $\varepsilon > 0$ we can compute $N(i, \varepsilon)$ such that change in i-th bit after $N(i, \varepsilon)$ steps has probability less than ε
- mappings defined in this way are layerwise computable
- output distribution is still computable: $m(x)$ = the probability that output starts with x, can be computed with arbitrary precision
- paradox: the same class of distributions

so it is enough to construct a rewriting machine that solves LLL with probability 1

Moser–Tardos probabilistic machine

Moser–Tardos probabilistic machine

- finds an assignment for infinite computable CNF

Moser–Tardos probabilistic machine

- finds an assignment for infinite computable CNF
- (assuming all clauses have m variables and at most 2^{m-2} neighbors)

Moser–Tardos probabilistic machine

- finds an assignment for infinite computable CNF
- (assuming all clauses have m variables and at most 2^{m-2} neighbors)
- enumerate all clauses, rank = maximal variable number

Moser–Tardos probabilistic machine

- finds an assignment for infinite computable CNF
- (assuming all clauses have m variables and at most 2^{m-2} neighbors)
- enumerate all clauses, rank = maximal variable number
- start with random values

Moser–Tardos probabilistic machine

- finds an assignment for infinite computable CNF
- (assuming all clauses have m variables and at most 2^{m-2} neighbors)
- enumerate all clauses, rank = maximal variable number
- start with random values
- find first unsatisfied clause and resample it

Moser–Tardos probabilistic machine

- finds an assignment for infinite computable CNF
- (assuming all clauses have m variables and at most 2^{m-2} neighbors)
- enumerate all clauses, rank = maximal variable number
- start with random values
- find first unsatisfied clause and resample it
- Moser–Tardos: this converges with probability 1

Moser–Tardos probabilistic machine

- finds an assignment for infinite computable CNF
- (assuming all clauses have m variables and at most 2^{m-2} neighbors)
- enumerate all clauses, rank = maximal variable number
- start with random values
- find first unsatisfied clause and resample it
- Moser–Tardos: this converges with probability 1
- they give an estimate for convergence speed

Moser–Tardos probabilistic machine

- finds an assignment for infinite computable CNF
- (assuming all clauses have m variables and at most 2^{m-2} neighbors)
- enumerate all clauses, rank = maximal variable number
- start with random values
- find first unsatisfied clause and resample it
- Moser–Tardos: this converges with probability 1
- they give an estimate for convergence speed
- so $N(i, \varepsilon)$ can be computed

Moser–Tardos probabilistic machine

- finds an assignment for infinite computable CNF
- (assuming all clauses have m variables and at most 2^{m-2} neighbors)
- enumerate all clauses, rank = maximal variable number
- start with random values
- find first unsatisfied clause and resample it
- Moser–Tardos: this converges with probability 1
- they give an estimate for convergence speed
- so $N(i, \varepsilon)$ can be computed
- Q.E.D.

Forbidden substrings

Forbidden substrings

- Let F be a set of strings ("forbidden strings"); assume that F contains at most $2^{\alpha n}$ strings of length n, where $\alpha < 1$ is a constant. Then there exists a constant c and a sequence ω that does not contain forbidden substrings of length greater than n.

Forbidden substrings

- Let F be a set of strings ("forbidden strings"); assume that F contains at most $2^{\alpha n}$ strings of length n, where $\alpha < 1$ is a constant. Then there exists a constant c and a sequence ω that does not contain forbidden substrings of length greater than n.
- (Combinatorial translation of Levin's lemma: for every $\alpha < 1$ there exists an everywhere α-complex sequence where all substrings y have complexity at least $\alpha|y| - O(1)$.)

Forbidden substrings

- Let F be a set of strings ("forbidden strings"); assume that F contains at most $2^{\alpha n}$ strings of length n, where $\alpha < 1$ is a constant. Then there exists a constant c and a sequence ω that does not contain forbidden substrings of length greater than n.
- (Combinatorial translation of Levin's lemma: for every $\alpha < 1$ there exists an everywhere α-complex sequence where all substrings y have complexity at least $\alpha|y| - O(1)$.)
- Computable version: let F be a computable set of forbidden strings...there exists a computable sequence ω...

Forbidden substrings

- Let F be a set of strings ("forbidden strings"); assume that F contains at most $2^{\alpha n}$ strings of length n, where $\alpha < 1$ is a constant. Then there exists a constant c and a sequence ω that does not contain forbidden substrings of length greater than n.
- (Combinatorial translation of Levin's lemma: for every $\alpha < 1$ there exists an everywhere α-complex sequence where all substrings y have complexity at least $\alpha|y| - O(1)$.)
- Computable version: let F be a computable set of forbidden strings...there exists a computable sequence ω...
- J. Miller's proof ("modified conditional expectations")

Forbidden substrings

- Let F be a set of strings ("forbidden strings"); assume that F contains at most $2^{\alpha n}$ strings of length n, where $\alpha < 1$ is a constant. Then there exists a constant c and a sequence ω that does not contain forbidden substrings of length greater than n.
- (Combinatorial translation of Levin's lemma: for every $\alpha < 1$ there exists an everywhere α-complex sequence where all substrings y have complexity at least $\alpha|y| - O(1)$.)
- Computable version: let F be a computable set of forbidden strings...there exists a computable sequence ω...
- J. Miller's proof ("modified conditional expectations")
- more complicated for bidirectional sequences

Forbidden substrings

- Let F be a set of strings ("forbidden strings"); assume that F contains at most $2^{\alpha n}$ strings of length n, where $\alpha < 1$ is a constant. Then there exists a constant c and a sequence ω that does not contain forbidden substrings of length greater than n.
- (Combinatorial translation of Levin's lemma: for every $\alpha < 1$ there exists an everywhere α-complex sequence where all substrings y have complexity at least $\alpha|y| - O(1)$.)
- Computable version: let F be a computable set of forbidden strings...there exists a computable sequence ω...
- J. Miller's proof ("modified conditional expectations")
- more complicated for bidirectional sequences
- for 2D sequences and $2^{\alpha S}$ forbidden rectangular patterns of area S: Lovasz local lemma is needed

Remarks

Remarks

- Breakthrough: Moser–Tardos algorithm

Remarks

- Breakthrough: Moser–Tardos algorithm
- better name: Moser–Tardos proof for trivial algorithm

Remarks

- Breakthrough: Moser–Tardos algorithm
- better name: Moser–Tardos proof for trivial algorithm
- layerwise computable mappings = almost everywhere defined mappings that correspond to rewriting machines with effective convergence

Remarks

- Breakthrough: Moser–Tardos algorithm
- better name: Moser–Tardos proof for trivial algorithm
- layerwise computable mappings = almost everywhere defined mappings that correspond to rewriting machines with effective convergence
- algorithmic randomness approach: layerwise computable mapping can be computed given the sequence and an upper bound for its randomness deficiency (Hoyrup, Rojas)

Remarks

- Breakthrough: Moser–Tardos algorithm
- better name: Moser–Tardos proof for trivial algorithm
- layerwise computable mappings = almost everywhere defined mappings that correspond to rewriting machines with effective convergence
- algorithmic randomness approach: layerwise computable mapping can be computed given the sequence and an upper bound for its randomness deficiency (Hoyrup, Rojas)
- computable points in a suitable metric space

Remarks

- Breakthrough: Moser–Tardos algorithm
- better name: Moser–Tardos proof for trivial algorithm
- layerwise computable mappings = almost everywhere defined mappings that correspond to rewriting machines with effective convergence
- algorithmic randomness approach: layerwise computable mapping can be computed given the sequence and an upper bound for its randomness deficiency (Hoyrup, Rojas)
- computable points in a suitable metric space
- using computable sequence outside a Schnorr null set as a pseudorandom sequence